

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

АНАЛИЗ ЭФФЕКТИВНОСТИ АЛГОРИТМОВ АКТИВНОГО
УПРАВЛЕНИЯ ОЧЕРЕДЬЮ В СЕТЕВЫХ УСТРОЙСТВАХ

Автор Балакишина Юлия Андреевна _____
(Фамилия, Имя, Отчество) (Подпись)
Направление подготовки 09.04.01 «Информатика и
(специальность) вычислительная техника»
Квалификация магистр
Руководитель Соснин Владимир Валерьевич,
к.т.н _____
(Фамилия, И., О., ученое звание, степень) (Подпись)

К защите допустить

Зав. кафедрой Алиев Т.И., профессор, д.т.н. _____
(Фамилия, И., О., ученое звание, степень) (Подпись)
« ____ » _____ 20 ____ г.

Санкт-Петербург, 20 18 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 ОБЗОР ЛИТЕРАТУРЫ И ПОСТАНОВКА ЗАДАЧИ	12
1.1 Проблема излишней сетевой буферизации	12
1.2 Терминологические несоответствия	14
1.3 Алгоритмы управления очередью	16
1.3.1 Алгоритм раннего произвольного обслуживания	18
1.3.2 Алгоритм управляемой задержки	19
1.3.3 Расширенный пропорционально-интегральный регулятор	20
1.4 Современное состояние проблемы исследования	21
1.5 Постановка задачи исследования	23
1.6 Выводы по главе 1	26
2 ТЕХНИЧЕСКОЕ ОПИСАНИЕ ИМИТАЦИОННОЙ МОДЕЛИ И ЭКСПЕРИМЕНТА.....	28
2.1 Обоснование выбора метода и инструментов исследования	28
2.2 Архитектура имитационной модели	28
2.3 Описание реализованной функциональности	30
2.4 Планирование эксперимента	32
2.5 Проведение эксперимента.....	35
2.6 Выводы по главе 2	37
3 АНАЛИЗ РЕЗУЛЬТАТОВ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ.....	39
3.1 Верификация модели.....	39
3.2 Анализ полученных результатов.....	41
3.3 Рекомендации по выбору алгоритма активного управления очередью ...	45
3.4 Выводы по главе 3	45
ЗАКЛЮЧЕНИЕ.....	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	50

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ	54
ПРИЛОЖЕНИЕ А. Код разработанной имитационной модели	55
ПРИЛОЖЕНИЕ Б. Скрипт для автоматизации запуска модели	61

ВВЕДЕНИЕ

Актуальность области исследования. Активное развитие информационных технологий привело к тому, что большая часть информации во всех сферах жизни человека оцифровывается для дальнейшего эффективного хранения и использования. При этом возникает необходимость передачи такой информации, представляющей собой разнородные формы: текстовые сообщения, изображения, видеоматериалы. Важно отметить, что зачастую требуется обеспечить высокую скорость и надёжность передаваемой информации. Логичным и правильным решением для такого запроса являются современные компьютерные сети.

Одной из основных задач любой компьютерной сети является обеспечение требуемого качества обслуживания (*англ.* QoS – Quality of Service), заключающееся в высокой скорости и надёжности передаваемых данных. С одной стороны, простым решением выглядит приобретение более быстрых и дорогих каналов связи и сетевых устройств. Но с другой стороны, правильнее и эффективнее сначала осуществить точную настройку имеющегося коммутационного оборудования.

Для такого рода настроек используются различные механизмы управления трафиком, ключевыми из которых являются дисциплины обслуживания и алгоритмы управления очередью. Алгоритмы активного управления очередью будут рассмотрены подробно в данной магистерской диссертации.

Объектом исследования являются алгоритмы активного управления очередью.

Предметом исследования является влияние современных алгоритмов активного управления очередью на среднюю скорость передачи фонового трафика, склонного к возникновению всплесков.

Цель исследования заключается в увеличении эффективности передачи фонового трафика при использовании современных алгоритмов активного управления очередью.

Для достижения цели исследования решались следующие задачи:

1. Провести аналитический обзор исследований в области активного управления очередью.
2. Изучить терминологический базис в исследуемой области. Определить соответствие терминологических понятий в русско- и англоязычной литературе.
3. Реализовать имитационную модель, которая позволит выполнить требуемые для исследования эксперименты.
4. Провести верификацию работы реализованной модели.
5. Составить план проведения экспериментов для достижения максимальной точности измерений при минимальном количестве опытов и сохранении статистической достоверности результатов
6. Провести имитационные эксперименты для исследования влияния современных алгоритмов активного управления очередью на среднюю скорость передачи фонового трафика.
7. На основании результатов экспериментов сформулировать рекомендации по выбору алгоритмов управления очередью в условиях передачи данных различного типа.

Теоретическая и методологическая основа исследования. Средствами исследования являются среда имитационного моделирования ns-3, табличный процессор Microsoft Excel 2015. В качестве методов исследования используются теория массового обслуживания, имитационное моделирование, методы математической статистики.

Актуальность темы исследования. Удешевление компьютерной памяти и желание производителей использовать в сетевых устройствах буферы «с запасом», то есть больше рекомендуемого размера, привело к возникновению такой проблемы, как излишняя сетевая буферизация.

Излишняя сетевая буферизация – это явление, при котором буферизация слишком большого количества данных вызывает нежелательную задержку пакетов в сети и снижение производительности передачи данных. Для борьбы с переполнением

буфера используют специальные механизмы, называемые алгоритмами управления очередью.

В современном сетевом оборудовании используют алгоритмы активного управления очередью. Данные алгоритмы осуществляют отбрасывание пакетов из очереди сетевого устройства до того, как она заполнится полностью. Это позволяет обеспечить корректную работу алгоритмов управления перегрузкой, так как отбрасываемые пакеты являются для них маркером загруженности канала. Кроме того, это позволяет поддерживать невысокие значения задержки пакетов в очереди, что особенно критично для интерактивных видов трафика.

Однако использование алгоритмов активного управления очередью может оказывать негативное влияние на передачу трафика компьютерных данных. Зачастую такой трафик характеризуется неравномерностью передачи данных, или так называемыми всплесками. Возникновение всплеска приводит к быстрому заполнению очереди буфера и отбрасыванию отправленных пакетов, что вызывает повторные передачи, уменьшение пропускной способности канала связи и увеличение времени передачи данных.

Таким образом, для корректного выбора алгоритма активного управления очередью требуется знать какой именно алгоритм оказывает наименьшее отрицательное влияние на трафик, наиболее часто передаваемый в настраиваемой сети. Такая информация может быть получена с помощью имитационного моделирования, чему и посвящена предлагаемая работа.

Степень теоретической разработанности темы. Первый алгоритм активного управления очередью RED был разработан С. Флойдом и В. Якобсоном в 1993 году, его основным предназначением являлось уведомление о перегрузке сети [1]. Концепция «излишней сетевой буферизации» была впервые предложена только в 2010 году Джимом Геттисом в его личном блоге [2]. Для борьбы с этой проблемой в 2012 и 2013 годах были предложены новые алгоритмы AQM, основной идеей которых стало использование в качестве метрики величины задержки, а не длины очереди

пакетов [3–4]. В дальнейшем исследования современных алгоритмов AQM были представлены в различных статьях. Кроме того, на сегодняшний день в сети интернет существует проект, целью которого является координация усилий экспертов для борьбы с излишней сетевой буферизацией [5].

Информационная база исследования. Для изучения принципов работы современных алгоритмов AQM использованы RFC-документы, а также статьи разработчиков данных алгоритмов [3–4; 6–7]. При описании русскоязычной терминологии за основу взято учебное пособие Т.И. Алиева «Основы моделирования дискретных систем» [8]. В качестве источников для изучения англоязычной литературы использован ряд статей [9–11]. Также выполнен обзор различных англоязычных статей в данной предметной области [3; 12–14].

Научная новизна исследования

1. Разработана имитационная модель, позволяющая исследовать влияние современных алгоритмов AQM на характеристики передачи различных видов трафика.
2. Выявлена зависимость качества работы алгоритмов AQM от параметров передаваемого трафика.
3. Сформированы рекомендации по выбору алгоритма AQM в зависимости от параметров передаваемого в конфигурируемой сети трафика.

Практическая значимость исследования. Разработанная имитационная модель может быть использован системными администраторами, а также студентами в образовательном процессе как инструмент для исследования алгоритмов AQM. Сформированные в ходе исследования рекомендации позволят грамотно выбрать алгоритм AQM, не оказывающий значительного негативного влияния на скорость передачи данных.

Апробация результатов исследования. Основные результаты были представлены на VI и VII Конгрессах молодых ученых, VIII Научно-практической конференции молодых ученых «Вычислительные системы и сети (Майоровские

чтения)». По теме исследования опубликованы две статьи, включая одну в журнале из перечня ВАК [15–16], и два тезиса докладов конференций [17–18].

Объем и структура работы. Выпускная квалификационная работа состоит из введения, трёх глав, заключения и библиографического списка. Общий объем работы составляет 63 страницы.

В первой главе описываются теоретические основы исследуемой области: проблема излишней сетевой буферизации, алгоритмы активного и пассивного управления очередью. Поднимается проблема терминологических несоответствий в русской англоязычной литературе. Выполняется обзор аналогичных исследований, и ставится задача исследования.

Во второй главе приведена архитектура разработанной модели и описаны основные этапы ее реализации. Представлен план проведения экспериментов.

В третьей главе описывается верификация разработанной функциональности, приводятся условия проведения экспериментов и полученные результаты, а также сформированные на их основании рекомендации.

Список используемой литературы состоит из 33 наименований. В работе присутствует 10 рисунков и 6 таблиц.

1 ОБЗОР ЛИТЕРАТУРЫ И ПОСТАНОВКА ЗАДАЧИ

1.1 Проблема излишней сетевой буферизации

В современном мире активно встает проблема излишней сетевой буферизации, то есть явления, при котором буферизация слишком большого количества данных вызывает нежелательную задержку пакетов в сети и снижение производительности передачи данных. Причиной возникновения данной проблемы является удешевление компьютерной памяти, которое позволило производителям сетевого оборудования снабжать устройства буферами большого размера.

Использование буферов в пакетной сети передачи данных необходимо, поскольку позволяет сгладить всплески трафика и улучшить общую производительность сети. При передаче данных между отправителем и получателем, пакет с большой долей вероятности проходит через несколько физических сетей, которые различаются канальной емкостью и уровнем загрузки. Из-за разницы в скоростях, пакеты полученные из более быстрой сети вынуждены ожидать пока очередной пакет будет отправлен через более медленную сеть. Для этого и используются буферы памяти.

Можно предположить, что размер буфера должен быть соизмерим с размером максимального окна передачи данных – в этом случае, при возникновении «всплеска», не произойдет потери данных. Рекомендованный размер буфера – $RTT \cdot BW$, где BW – пропускная способность исходящего канала, а в качестве RTT принято выбирать значение 100 мс – величину задержки трансатлантической передачи [19]. Однако на практике подобрать идеальный размер буфера невозможно. Вариация параметров отдельных потоков данных, мультиплексируемых в канале слишком велика и динамична для статически заданного параметра. Кроме того, параметры самой физической среды могут меняться, например, изменение местоположение ноутбука или планшета может на порядок изменить пропускную способность в беспроводной сети. Вследствие этого даже местоположение «бутылочного горлышка» может

изменяться. Таким образом, невозможность подбора идеального размера буфера, а также неполное понимание принципов работы очереди привело производителей к желанию использовать буферы «с запасом», то есть больше рекомендуемого размера. А удешевление компьютерной памяти сделало это возможным.

Проблема использования буферов большого размера заключается в следующем: в случае возникновения затора, пакеты начинают скапливаться в буфере, из-за этого потерь пакетов не происходит, и, соответственно, сигнал источнику трафика о необходимости снижения скорости передачи данных не отправляется. В результате алгоритм управления перегрузкой (TCP-congestion control) не способен своевременно адаптировать скорость передачи данных к доступной канальной емкости, что вызывает задержки, потери и повторные передачи, значительно уменьшающие реальную пропускную способность. Проблема излишней сетевой буферизации особенно критична для интерактивных видов трафика, поскольку они наиболее чувствительны к задержкам.

На практике проблема возникает при высокой загрузке сети, и выражается в виде долгого отображения веб-страниц (несколько секунд, или даже минут) и невозможности работать с приложениями, чувствительными к задержкам.

Актуальность проблемы подтверждают данные, представленные на рисунке 1. С помощью специального приложения пользователями по всему миру было произведено более 130000 тестов. В каждом тесте увеличивалась скорость передачи данных до тех пор, пока буферы не оказывались заполненными. Исследователи осуществили анализ распределения полученных пакетов и сделали вывод о реальной пропускной способности каналов связи и возможном размере буферов. На рисунке 1, каждая точка соответствует проведенному тесту, на горизонтальной оси указаны наиболее распространенные размеры буферов, а на вертикальной – скорости передачи данных. Диагональные линии показывают дополнительную задержку, порожденную буферизацией [20].

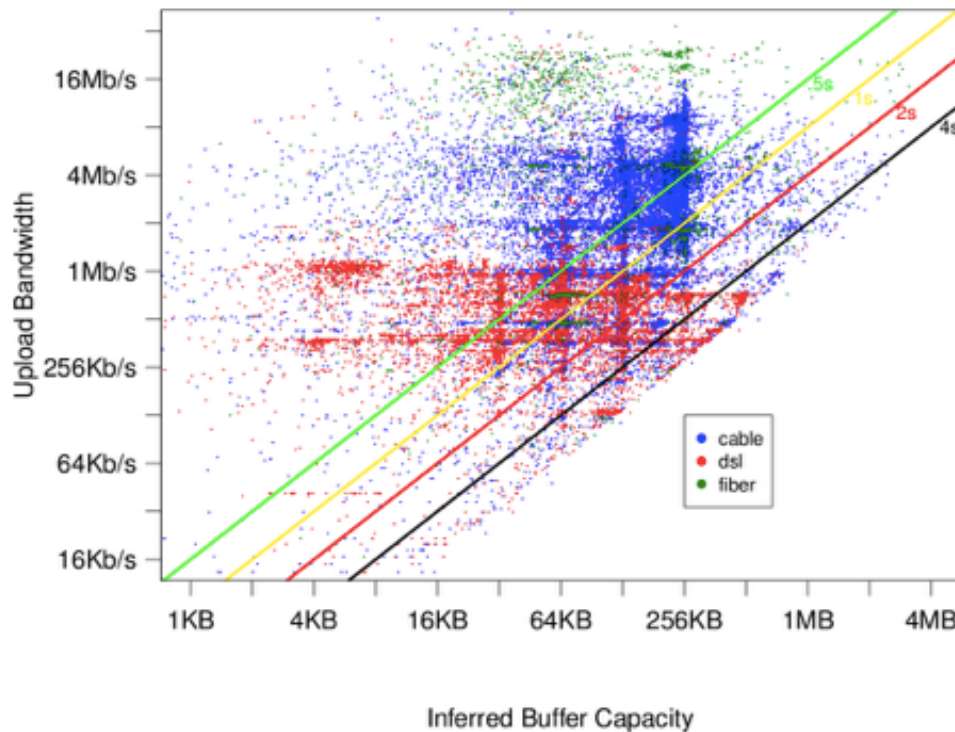


Рисунок 1 – Результаты исследований, подтверждающие актуальность проблемы излишней сетевой буферизации

Для борьбы с излишней сетевой буферизацией используют специальные механизмы, называемые алгоритмами управления очередью.

1.2 Терминологические несоответствия

В связи с тем, что большинство используемых литературных источников являются англоязычными, при написании данной работы возникли проблемы, связанные с терминологическими неточностями при переводе.

В таблице 1 представлены используемые термины, их встречаемый перевод и соответствующие русскоязычные аналоги.

Таблица 1 – Таблица соответствия терминологии

Англоязычный термин	Встречаемые в литературе переводы	Соответствующий русскоязычный термин
Queueing discipline	Дисциплина обслуживания, алгоритм планирования очереди, алгоритм управления очередью	—

Англоязычный термин	Встречаемые в литературе переводы	Соответствующий русскоязычный термин
Scheduling algorithm	Алгоритм планирования, Планировщик, Алгоритм распределения ресурсов, Дисциплина обслуживания	Дисциплина обслуживания
Queue management algorithm	Алгоритм управления очередью, Дисциплина обслуживания	Алгоритм управления очередью

Основная проблема заключается в том, что в различных переводах англоязычных статей можно встретить различный перевод одного и того же термина. Так, многие литературные источники переводят термин «Queueing discipline» как «Дисциплина обслуживания», «Алгоритм планирования очереди» или «Алгоритм управления очередью». Однако, в русскоязычной литературе данные термины могут иметь различное значение, а некоторые могут не использоваться вовсе. Далее рассматриваются определения наиболее спорных терминов, представленных в русско- и англоязычной литературе.

Термины, используемые в англоязычной литературе:

Queueing discipline (QDisc) – это механизм управления очередью в сетевом устройстве. QDisc может быть представлен в виде Scheduling algorithm, Queue management algorithm или их комбинации.

Scheduling algorithm – это алгоритм представляющий собой правило выбора сетевого пакета из очереди на обслуживание. Примерами данных алгоритмов являются алгоритмы FIFO, FQ, AQ, PQ, WFQ.

Queue management algorithm – это алгоритм, позволяющий ограничивать длину очереди сетевого устройства. Примерами данных алгоритмов являются TailDrop, RED, CoDel, PIE.

Также существуют QDisc, которые содержат в себе функции Scheduling и Queue management алгоритмов. Примером может послужить алгоритм fq_codel.

Термины, используемые в русскоязычной литературе:

Дисциплина обслуживания – правило, по которому выбирается сетевой пакет из очереди на обслуживание.

Алгоритм управления очередью – механизм ограничения длины очереди сетевого устройства.

Таким образом, можно видеть, что термин «Дисциплина обслуживания» в русскоязычной литературе соответствует термину «Scheduling algorithm» в англоязычной, а термин «Алгоритм управления очередью» соответствует термину «Queue management algorithm».

Стоит отметить, что в русскоязычной литературе термины «Дисциплина обслуживания» и «Алгоритм управления очередью» рассматриваются как самостоятельные и никакого обобщающего термина, как «QDisc» в англоязычной литературе, для них не используется.

1.3 Алгоритмы управления очередью

Алгоритмы управления очередью – алгоритмы, реализующие отбрасывание сетевых пакетов, когда буфер заполнен или приближается к заполнению. Разделяются на активные и пассивные [21].

Пассивное управление очередью. Алгоритмы пассивного управления очередью осуществляют отбрасывание пакетов, когда очередь заполняется полностью (достигает указанного размера в пакетах или байтах). Данные алгоритмы имеют два рабочих состояния: состояние, в котором пакеты не отбрасываются и отбрасывание всех поступивших пакетов.

Достоинством данных алгоритмов является простота реализации с невысокими вычислительными накладными расходами.

Недостатки данных алгоритмов:

- Раннее предупреждение отправителей о перегрузке отсутствует. Пакеты начинают отбрасываться в момент, когда очереди уже заполнены. Это приводит к неправильной работе алгоритма управления перегрузкой.
- При использовании данных алгоритмов необходим поиск компромисса между размером буфера и качеством обслуживания, поскольку это единственный способ сократить проблему излишней сетевой буферизации.
- В очереди может быть большое количество пакетов в течение длительного времени, что будет вызывать нежелательные задержки.
- Возможность возникновения блокировки (lock-out): ситуации, когда один или несколько потоков пакетов монополизируют пространство буфера и предотвращают попадание в очередь других потоков. Это приводит к проблемам справедливого распределения потоков.

Примерами алгоритмов пассивного управления очередями являются алгоритм отбрасывания хвоста (Tail-Drop), отбрасывания головы (Drop-Head) и случайного отбрасывания пакетов (Random drop).

Активное управление очередью. Алгоритмы активного управления очередью (active queue management, AQM) осуществляют отбрасывание пакетов до того, как очередь заполнится полностью. При этом, при увеличении загрузки канала связи, увеличивается и количество отбрасываемых пакетов. Этот механизм обеспечивает корректную работу алгоритмов управления перегрузкой, так как отбрасываемые пакеты являются для них маркером загрузки канала. Алгоритмы AQM преследуют следующие цели:

- Сократить нежелательную задержку, возникающую из-за длительного нахождения пакетов в очереди.
- Избежать возможности возникновения блокировки, справедливо распределяя пропускную способность между несколькими потоками данных.

Наиболее распространенными алгоритмами AQM на сегодняшний день являются алгоритмы CoDel, PIE, RED и его модификации [22].

1.3.1 Алгоритм раннего произвольного обслуживания

Алгоритм раннего произвольного обслуживания (Random Early Detection, RED) был разработан в 1993 году и стал первым алгоритмом AQM. Основным предназначением алгоритма RED является раннее предупреждение источников трафика о возникновении перегрузки сети и необходимости снижения интенсивности передачи информации. На сегодняшний день RED реализован в большинстве сетевых устройств, однако в связи с тем, что он требует тщательной настройки параметров для различных условий работы сети, многие сетевые администраторы не используют данный алгоритм [1].

В качестве показателя перегрузки описываемый алгоритм использует переменную avg , которая представляет собой экспоненциально сглаженную длину очереди и вычисляется по следующей формуле:

$$avg = (1-wq) \cdot avg + wq \cdot q \quad (1)$$

где wq – коэффициент сглаживания, q – текущая длина очереди.

По умолчанию коэффициент сглаживания равен 0,002, но при желании его можно изменить с помощью параметра wq .

Алгоритм RED работает следующим образом:

1. Происходит расчет сглаженной длины очереди и его сравнение с минимальным порогом длины очереди. Минимальный порог измеряется в пакетах и по умолчанию равен 5. Эта величина является настраиваемой и при желании ее можно изменить с помощью параметра $minth$.
2. Если сглаженная длина очереди меньше, чем минимальный порог, поступающие пакеты не отбрасываются.
3. Если сглаженная длина очереди находится между минимальным и максимальным порогом, RED определяет начинающуюся перегрузку в сети и

отбрасывает пакеты случайным образом. Вероятность отбрасывания пропорциональна сглаженной длине очереди и изменяется линейно между 0 и максимальной вероятностью отбрасывания пакетов. По умолчанию данная вероятность равна 0,1, но ее можно изменить с помощью параметра `maxp`. Максимальный порог длины очереди измеряется в пакетах и по умолчанию равен 50. Эта величина так же является настраиваемой и при желании ее можно изменить с помощью параметра `maxth`.

4. Если сглаженная длина очереди превышает максимальный порог, то RED отбрасывает все прибывающие пакеты.

1.3.2 Алгоритм управляемой задержки

Алгоритм управляемой задержки (Controlled Delay, CoDel) был разработан в 2012 году. Основным предназначением данного алгоритма, как и алгоритма RED, является раннее предупреждение отправителя о перегрузке сети. Однако в качестве метрики данный алгоритм использует не длину очереди, а минимальное значение задержки пакетов в буфере. Это позволяет CoDel напрямую бороться с нежелательной задержкой в буфере, не прибегая к косвенным метрикам для ее оценки. Также данный подход позволяет избежать потерь пакетов при всплесках трафика, вызывающих возникновение очередей, которые затем быстро уменьшаются и не оказывают существенного влияния на производительность сети. Кроме того, алгоритм CoDel является более простым в настройке по сравнению с алгоритмом RED [3; 6].

Описываем алгоритм позволяет поддерживать минимальную величину задержки пакетов в буфере ниже установленного параметра. По умолчанию этот параметр равен 5 мс, однако при желании его можно изменить, используя параметр дисциплины `_target_`.

Алгоритм CoDel работает следующим образом:

1. В течение определенного интервала времени происходит измерение значений задержек пакетов в буфере. По умолчанию этот интервал равен 100 мс, но при желании его можно изменить с помощью параметра `_interval_`.
2. Если за это время некоторая минимальная задержка превысит значение установленного параметра (по умолчанию 5 мс), пакет, находящийся в очереди дольше всего, отбросится, а интервал уменьшится. Интервал уменьшается посредством деления значения интервала на квадратный корень из числа, соответствующего количеству последовательных интервалов в которых происходило отбрасывание пакетов из-за превышения установленного параметра задержки. По умолчанию, значения будут следующими: $100, \frac{100}{\sqrt{2}}, \frac{100}{\sqrt{3}}, \frac{100}{\sqrt{4}} \dots$
3. Как только минимальная задержка опять достигнет заданного значения, отбрасывание пакетов прекратится и интервал примет свое первоначальное значение.

1.3.3 Расширенный пропорционально-интегральный регулятор

Расширенный пропорционально-интегральный регулятор (Proportional Integral controller Enhanced, PIE) был разработан в 2013 году. Он базируется на пропорционально-интегральном регуляторе, но его основная цель – управление задержкой в очереди буфера. Данный алгоритм сочетает в себе преимущества CoDel и RED алгоритмов: в качестве показателя перегрузки он использует задержку пакетов в очереди, как алгоритм CoDel, при этом он отбрасывает пакеты случайным образом при обнаружении перегрузки, как алгоритм RED [4; 7].

Алгоритм PIE работает следующим образом:

1. Рассчитывается скорость выхода пакетов из очереди `depart_rate`. Для этого измеряется длина очереди `qlen` и сравнивается с минимальным порогом длины очереди `dq_threshold` (по умолчанию параметр равен 16 KB). Дальнейшие вычисления проводятся только при условии, что длина очереди превышает

минимальный порог. Это делается для того, чтобы исключить влияние небольших непостоянных всплесков трафика, которые могут приводить к быстрому опустошению очереди. Далее, при выполнении условия, происходит расчет количества байтов, выходящих из очереди до тех пор, пока это значение не превысит минимального порога `dq_treshhold`. Как только это происходит, производится деление вышедшего числа байтов на затраченное время. Затем значение обнуляется и расчет повторяется циклически.

2. Используя полученную скорость выхода пакетов из очереди рассчитывается текущая задержка пакетов в очереди. Для этого текущая длина очереди `qlen` делится на скорость выхода пакетов из очереди `depart_rate`.
3. На основе полученного значения текущей задержки рассчитывается вероятность отбрасывания пакетов p . Для этого используется следующая формула:

$$p = p + \alpha \cdot (est_del - target_del) + \beta \cdot (est_del - est_del_old), \quad (2)$$

где p – вероятность отбрасывания пакетов, est_del – текущая задержка пакетов в очереди, est_del_old – задержка пакетов в очереди на предыдущем цикле расчетов, $target_del$ – установленная параметром задержка, α и β – коэффициенты, позволяющие управлять ростом вероятности отбрасывания пакетов. α определяет насколько отклонение текущей задержки от установленной параметром влияет на изменения вероятности отбрасывания пакетов. А β оказывает дополнительные корректировки на влияние изменения задержки на вероятность отбрасывания пакетов.

4. Таким образом, отбрасывание пакетов происходит случайным образом в соответствии с вероятностью, которая зависит от задержки пакетов в очереди.

1.4 Современное состояние проблемы исследования

Исследуемая предметная область представлена в различных статьях. В каждой из них рассматриваются различные аспекты современных алгоритмов AQM. Многие исследователи изучают эффективность существующих алгоритмов AQM с целью

поиска наилучшего. Наиболее популярные алгоритмы исследуются с целью определения их рабочих диапазонов и способов настройки для различных условий сети. Далее представлен краткий обзор некоторых работ из данной области.

В работе [12] авторы сравнивают алгоритмы RED и CoDel с точки зрения их возможности уменьшать задержку пакетов для различных вариаций протокола TCP. В рамках исследования проводилось моделирование работы данных алгоритмов, измерялись задержки пакетов и время передачи 10 Мб данных FTP-трафика при параллельной передаче фоновых трафика. В данных условиях исследователи получили следующие результаты: CoDel уменьшает задержку на 87 %, RED на 75 %. При этом время передачи данных при использовании CoDel на 42% дольше, чем при использовании RED. Кроме того, авторы обращают внимание на то, что RED является более изученным алгоритмом, имеющим различные рекомендации по настройке. К недостатку данной работы стоит отнести то, что исследователи не обосновывают выбор данных, используемых для моделирования. Так, FTP-трафик является малочувствительным к задержкам, и алгоритмы активного управления очередью для улучшения качества передачи такого рода данных обычно не используются. Также непонятно почему выбран именно такой размер данных для основного потока, и потока фоновых данных.

В работе [13] авторы исследуют рабочие диапазоны, настраиваемость и производительность алгоритмов активного управления очередью CoDel и PIE. В статье отражены результаты моделирования для различных условий сети, представлены рабочие диапазоны данных алгоритмов с учетом загрузки сети и задержки распространения сигнала в прямом и обратном направлении, исследованы способы настройки данных алгоритмов. Данная работа позволяет получить развернутое представление о работе алгоритмов CoDel и PIE в различных условиях сети и способах их настройки, единственным недостатком может послужить отсутствие практических рекомендаций по настройке алгоритмов и установлению определенных значений параметров при работе в сетях, находящихся за пределами

рабочих диапазонов алгоритмов, то есть тех диапазонов в которых алгоритмы работают менее эффективно со значениями параметров по умолчанию.

В [14] описаны различные алгоритмы AQM, продемонстрированы их слабые и сильные стороны. Также авторы заявляют о том, что они произвели сравнение описываемых алгоритмов. Однако стоит отметить, что в работе не представлены показатели, по которым данные алгоритмы сравниваются. Кроме того, авторы не проводят самостоятельных экспериментов, позволяющих получить данные для сравнения, а только используют информацию из открытых источников.

При изучении алгоритма AQM CoDel наибольший интерес представляет статья разработчиков данного алгоритма К. Николс и В. Якобсона [3]. В ней исследователи подробно описывают проблемы излишней сетевой буферизации и принципы работы нового алгоритма, способного их решить. Также описываются проведенные имитационные эксперименты и их результаты. Алгоритм CoDel сравнивается с алгоритмами RED и DropTail по различным показателям. Стоит отметить, что В. Якобсон также являлся одним из разработчиков алгоритма RED, а также одним из основных разработчиков стека протоколов TCP/IP. Данная работа вносит большой вклад в исследуемую предметную область, поскольку позволяет получить полную информацию о работе алгоритма непосредственно от его разработчиков.

Таким образом, можно отметить, что исследуемая предметная область представляет интерес для научного сообщества. Используемые сегодня алгоритмы активного управления очередью были разработаны 5-6 лет назад, и, следовательно, не являются досконально изученными. Данная магистерская диссертация преследует цель анализа эффективности и сравнения двух наиболее популярных на сегодняшний день алгоритмов: CoDel и PIE.

1.5 Постановка задачи исследования

При исследовании алгоритмов AQM многие авторы используют минимизацию среднего значения задержки пакетов в очереди в качестве критерия эффективности.

Это связано с тем, что одна из основных целей алгоритмов AQM – борьба с нежелательными задержками при излишней сетевой буферизации. Поэтому такого рода исследования позволяют получить объективные данные об эффективности применения алгоритма. Однако при исследовании алгоритмов CoDel и PIE данный подход не является рациональным. Основной задачей данных алгоритмов AQM является поддержание минимального значения задержки в очереди ниже определенного значения. Это приводит к тому, что измерение значений задержки при изменении условий сети не позволит получить наглядных результатов. Поэтому при исследовании данных алгоритмов требуется выбрать такой критерий, который позволит оценить эффективность алгоритма, но не будет привязан к величине задержки пакетов в очереди.

Следует отметить, что разработчики алгоритмов AQM, которые используют в качестве основной метрики задержку, утверждают, что данные алгоритмы являются менее чувствительными к всплескам трафика, в отличие от алгоритмов, которые используют в качестве метрики длину очереди. Это происходит потому, что возникновение всплеска вызывает быстрое заполнение очереди, которое приводит к немедленному отбрасыванию пакетов в алгоритмах старого поколения. Современные алгоритмы не привязаны к длине очереди и позволяют разрешить всплеск без потерь. Однако численные данные, позволяющие оценить, с какой величиной всплеска трафика может справиться тот или иной алгоритм, отсутствуют.

Для трафика компьютерных данных проблема всплесков или так называемой неравномерности данных встает особенно остро. Это происходит потому, что зачастую такой трафик обладает невысокими требованиями к качеству обслуживания, и приложения, его генерирующие, не обязаны поддерживать выдачу пакетов таким образом, чтобы обеспечить минимальную вариацию задержки, а также минимальный разброс размеров пакетов. Также из-за невысоких требований к качеству обслуживания обычно данный тип трафика используется в исследованиях в качестве фонового.

При исследовании работы алгоритмов AQM со смешанными видами трафика, большинство исследователей делает упор в изучении на приоритетные виды трафика, такие как VoIP, поскольку они наиболее чувствительны к задержкам. Характеристики же фонового трафика, который чаще всего представляет собой трафик компьютерных данных, обычно в статьях не представляются.

В данной работе предлагается исследовать влияние современных алгоритмов AQM на характеристики передачи фонового трафика, склонного к возникновению всплесков. Исследование предполагает создание имитационной модели, в которой будет производиться передача одного VoIP-потока и одного потока фонового трафика (трафика компьютерных данных). Пакеты данных из этих потоков будут помещаться в общую очередь, с которой работает изучаемый алгоритм AQM. При проведении экспериментов предполагается изменение параметров фонового трафика, загрузки канала, а также размера буфера сетевого устройства. В результате проведенных экспериментов будут собраны данные о средней скорости передачи фонового трафика при использовании различных алгоритмов AQM. Это позволит сделать вывод о том, какие алгоритмы справляются наилучшим образом с фоновым трафиком, обладающим определенным набором параметров. Таким образом, в качестве критерия эффективности при исследовании алгоритмов AQM предлагается использовать максимизацию средней скорости передачи фонового трафика.

Исходя из поставленной задачи, к разрабатываемой модели предъявляются следующие требования:

1. Модель должна поддерживать работу с современными алгоритмами AQM.
2. Модель должна поддерживать работу TCP и UDP протоколов, поскольку это основные протоколы передачи компьютерных и интерактивных видов трафика.
3. В модели должна быть доступна возможность задания функциональных, нагрузочных и структурных параметров сетевого устройства.
4. В ходе проведения эксперимента должна рассчитываться такая сетевая характеристика, как средняя скорость передачи данных исследуемого потока.

После построения данной имитационной модели предлагается произвести ее верификацию, провести ряд экспериментов согласно предварительно составленному плану экспериментов, проанализировать результаты и составить рекомендации по выбору алгоритма AQM.

1.6 Выводы по главе 1

1. Удешевление компьютерной памяти и неполное понимание принципов работы очереди производителями сетевого оборудования привело к возникновению проблемы излишней сетевой буферизации, то есть явления, при котором буферизация слишком большого количества данных вызывает нежелательную задержку пакетов в сети и снижение производительности передачи данных.
2. Для борьбы с переполнением буфера используют специальные механизмы, называемые алгоритмами управления очередью. Существуют активные и пассивные алгоритмы управления очередью. В современном сетевом оборудовании используют алгоритмы активного управления очередью. Данные алгоритмы осуществляют отбрасывание пакетов из очереди сетевого устройства до того, как она заполнится полностью.
3. Наиболее перспективными считаются алгоритмы AQM, которые в качестве метрики для начала отбрасывания пакетов используют не длину очереди, а минимальное значение задержки пакетов в буфере. Это позволяет напрямую бороться с нежелательной задержкой, не прибегая к косвенным метрикам для ее оценки.
4. Существуют терминологические несоответствия в англо- и русскоязычных литературных источниках. Стоит обратить внимание, что в русскоязычной литературе термины «Дисциплина обслуживания» и «Алгоритм управления очередью» рассматриваются как самостоятельные и в отличие от англоязычной литературы не имеют обобщающего термина, такого как «Queueing discipline».

5. Обзор исследований в данной области показал недостаток структурированной информации, позволяющей однозначно оценить и выбрать алгоритм AQM в соответствии с параметрами настраиваемой сети.
6. При исследовании алгоритмов AQM многие авторы используют минимизацию среднего значения задержки пакетов в очереди в качестве критерия эффективности. Однако при исследовании алгоритмов, целью которых является поддержание минимального значения задержки в очереди ниже определенного значения данный метод не подходит. В данной работе в качестве критерия эффективности предлагается использовать максимизацию средней скорости передачи фонового трафика. Полученные данные позволят сделать вывод о том, какие алгоритмы AQM справляются наилучшим образом с фоновым трафиком, обладающим определенным набором параметров.

2 ТЕХНИЧЕСКОЕ ОПИСАНИЕ ИМИТАЦИОННОЙ МОДЕЛИ И ЭКСПЕРИМЕНТА

2.1 Обоснование выбора метода и инструментов исследования

Для исследования AQM было принято решение использовать методы имитационного моделирования. Данные методы позволяют проводить исследования менее трудоемко, чем при работе с реализованными алгоритмами на реальных или виртуальных машинах. Имитационная модель позволит сравнивать различные AQM, исследовать их характеристики при различных параметрах как трафика, так и самих алгоритмов.

В качестве среды имитационного моделирования был выбран сетевой симулятор ns-3, так как он обладает следующим рядом преимуществ [23].

1. Открытый исходный код всех компонентов. Это позволяет детально изучить логику работы сетевого симулятора ns-3 и упрощает модификацию его базовых возможностей.
2. Распространение программы по лицензии свободного программного обеспечения. То есть все компоненты симулятора доступны бесплатно.
3. Работоспособность в свободно распространяемой операционной системе Linux.
4. Существующая база разработанных моделей в области исследования. Библиотека ns-3 содержит в себе базу различных алгоритмов управления очередью, включая TailDrop, RED, CoDel и PIE [24].

2.2 Архитектура имитационной модели

В архитектуре имитационной модели можно выделить следующие основные составляющие (рисунок 2):

1. Входные данные. Задаются пользователем с помощью командной строки или модификации скрипта в командной оболочке `shell` (описание скрипта представлено в разделе 2.5), включают в себя структурные, функциональные, и нагрузочные параметры сетевого устройства.

2. Топология исследуемой сети. Реализованная топология представлена на рисунке
3. Данный вид топологии рекомендуется использовать для исследования производительности TCP, а также алгоритмов AQM [25–26].
3. Реализованные в системе ns-3 алгоритмы AQM: PIE и CoDel.
4. Генераторы трафика, позволяющие создавать потоки данных с заданными параметрами.
5. Модуль расчета сетевых характеристик. Набор функций, реализующих логику расчета требуемых сетевых характеристик на основании полученных трасс данных.

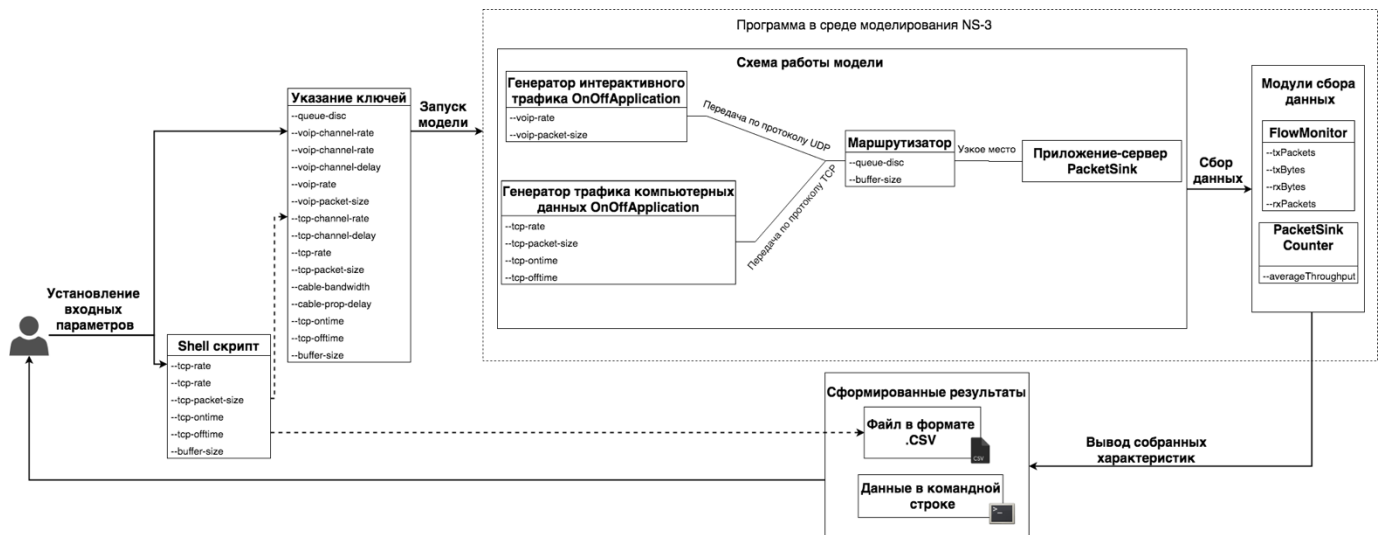


Рисунок 2 – Архитектура разработанной модели

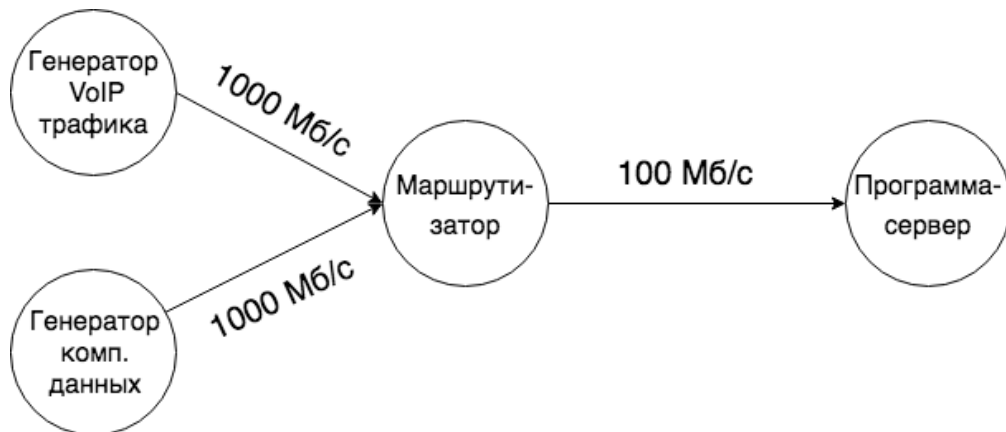
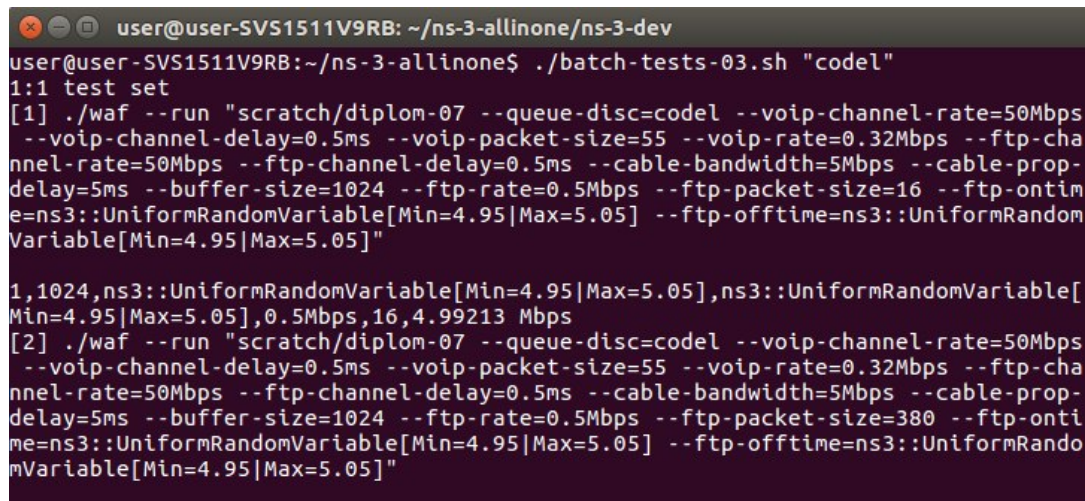


Рисунок 3 – Реализованная топология

2.3 Описание реализованной функциональности

Имитационная модель была разработана в среде моделирования ns-3 с помощью языка программирования C++. Код программы представлен в приложении 1.

Входные параметры и рассчитанные характеристики. На рисунке 4 представлен результат работы имитационной модели. Так как запуск модели осуществляется с помощью командной строки, было принято решение реализовать задание входных параметров и отображение рассчитанных характеристик также с ее помощью.



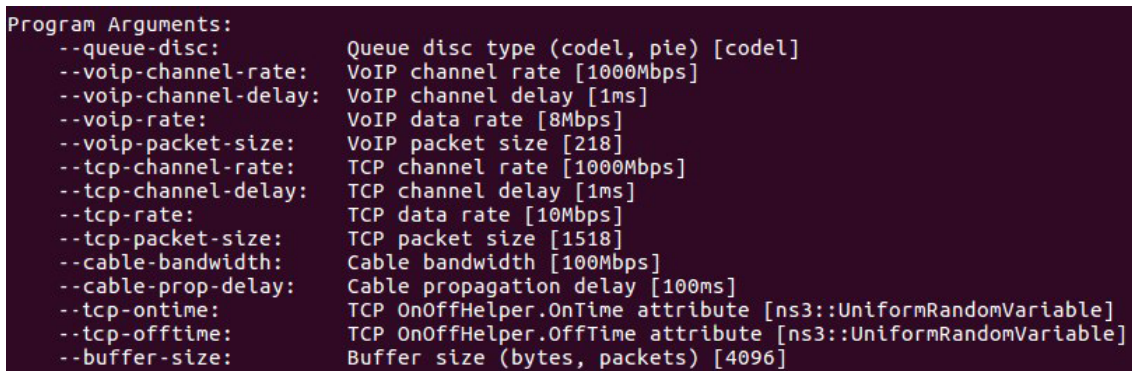
```

user@user-SVS1511V9RB: ~/ns-3-allinone/ns-3-dev
user@user-SVS1511V9RB:~/ns-3-allinone$ ./batch-tests-03.sh "codel"
1:1 test set
[1] ./waf --run "scratch/diplom-07 --queue-disc=codel --voip-channel-rate=50Mbps
--voip-channel-delay=0.5ms --voip-packet-size=55 --voip-rate=0.32Mbps --ftp-channel-rate=50Mbps --ftp-channel-delay=0.5ms --cable-bandwidth=5Mbps --cable-prop-delay=5ms --buffer-size=1024 --ftp-rate=0.5Mbps --ftp-packet-size=16 --ftp-ontime=ns3::UniformRandomVariable[Min=4.95|Max=5.05] --ftp-offtime=ns3::UniformRandomVariable[Min=4.95|Max=5.05]"
1,1024,ns3::UniformRandomVariable[Min=4.95|Max=5.05],ns3::UniformRandomVariable[Min=4.95|Max=5.05],0.5Mbps,16,4.99213 Mbps
[2] ./waf --run "scratch/diplom-07 --queue-disc=codel --voip-channel-rate=50Mbps
--voip-channel-delay=0.5ms --voip-packet-size=55 --voip-rate=0.32Mbps --ftp-channel-rate=50Mbps --ftp-channel-delay=0.5ms --cable-bandwidth=5Mbps --cable-prop-delay=5ms --buffer-size=1024 --ftp-rate=0.5Mbps --ftp-packet-size=380 --ftp-ontime=ns3::UniformRandomVariable[Min=4.95|Max=5.05] --ftp-offtime=ns3::UniformRandomVariable[Min=4.95|Max=5.05]"

```

Рисунок 4 – Результат работы имитационной модели

Для задания необходимых параметров, при запуске модели на исполнение необходимо указать требуемый параметр и его значение в следующем формате «--parameter=x», где parameter – изменяемый параметр, а x – задаваемое значение. На рисунке 5 приведен список изменяемых параметров.



```

Program Arguments:
--queue-disc:           Queue disc type (codel, pie) [codel]
--voip-channel-rate:    VoIP channel rate [1000Mbps]
--voip-channel-delay:   VoIP channel delay [1ms]
--voip-rate:            VoIP data rate [8Mbps]
--voip-packet-size:     VoIP packet size [218]
--tcp-channel-rate:     TCP channel rate [1000Mbps]
--tcp-channel-delay:    TCP channel delay [1ms]
--tcp-rate:             TCP data rate [10Mbps]
--tcp-packet-size:      TCP packet size [1518]
--cable-bandwidth:      Cable bandwidth [100Mbps]
--cable-prop-delay:     Cable propagation delay [100ms]
--tcp-ontime:           TCP OnOffHelper.OnTime attribute [ns3::UniformRandomVariable]
--tcp-offtime:          TCP OnOffHelper.OffTime attribute [ns3::UniformRandomVariable]
--buffer-size:          Buffer size (bytes, packets) [4096]

```

Рисунок 5 – Список изменяемых в модели параметров

Топология сети и используемые алгоритмы. Реализованная топология представлена в разделе 2.2. При создании каналов связи была учтена задержка распространения (установлено значение 100 мс, равное величине задержки трансатлантической передачи), а также задержка выхода пакета в канал, которая рассчитывается как «размер пакета / пропускная способность». После создания топологии исследуемый алгоритм AQM был подключен на канал связи между узлами R и Dest.

Генерация трафика. Для генерации трафика двух типов было использовано приложение OnOffApplication. Данное приложение позволяет задавать настройки, с помощью которых можно имитировать передачу трафика разных классов. Кроме того, оно позволяет с легкостью имитировать возникновение всплесков, то есть искусственно создавать неравномерность передаваемого трафика. Принцип работы OnOffApplication заключается в следующем: генерация трафика осуществляется по шаблону включен/выключен (англ. On/Off), то есть после вызова данного приложения происходит чередование состояний «включен» и «выключен». Продолжительность каждого из этих состояний задается с помощью переменных onTime и offTime соответственно. Во время состояния «выключен» трафик не генерируется. Во время состояния «включен» генерируется трафик, характеризующийся заданной скоростью передачи данных и размером пакета. Также OnOffApplication позволяет имитировать передачу данных по UDP или TCP протоколу [27].

Сбор характеристик. Для сбора характеристик был подключен модуль DynamicCast<PacketSink>. Он позволяет получить информацию о количестве полученных узлом-приемником полезных байтов определенного потока. Для получения средней скорости передачи данных полученное значение делилось на время симуляции

2.4 Планирование эксперимента

Для достижения максимальной точности измерений при минимальном количестве опытов и сохранении статистической достоверности результатов было осуществлено планирование эксперимента. Планирование эксперимента представляет собой комплекс мероприятий, заключающийся в выполнении ряда этапов [28].

1. Установление цели эксперимента. В рамках представленной работы целью проведения эксперимента является анализ эффективности алгоритмов AQM, и сравнение наиболее актуальных алгоритмов между собой.
2. Уточнение условий проведения эксперимента. Для проведения эксперимента используется имитационная модель, разработанная в рамках данной выпускной квалификационной работы. Данная модель реализована в свободно распространяемой системе моделирования ns-3, и предполагает работу в операционной системе Linux, которая также распространяется по лицензии свободного программного обеспечения. Это позволяет использовать имитационную модель бесплатно и в рамках ресурсов, предоставляемых обычным персональным компьютером.
3. Выявление и выбор входных и выходных параметров. Входные параметры включают в себя структурные (исследуемый алгоритм со своим набором параметров), функциональные (размер буфера), и нагрузочные параметры сетевого устройства (характеристики трафика: размер пакетов, on/off интервалы, скорость генерации трафика). Выходные параметры: количество отправленных пакетов, количество отправленных байтов, пропускная способность потока. Далее представлена таблица 2 со значениями параметров, которые предлагается варьировать в ходе проведения эксперимента.

Таблица 2 – Варьируемые параметры

Название параметра	Минимальное значение	Максимальное значение
Размер пакетов фоновой трафика	64 байта	1518 байт
Скорость генерации фоновой трафика	10 Мб/с	80 Мб/с
Пульсация onTime для фоновой трафика	1%	99%
Пульсация offTime для фоновой трафика	1%	99%
Размер буфера	4 кибибайта	16 мебибайт

В качестве минимального значения пакета фоновой трафика было выбрано 64 байта, поскольку данное значение представляет собой минимальную длину Ethernet-кадра. В качестве максимального значения было выбрано значение 1518 байт, что соответствует максимальной длине Ethernet-кадра [29]. Скорости генерации трафика были выбраны таким образом, чтобы обеспечить низкую и высокую загрузку канала связи. Пульсация времен onTime и offTime позволяет регулировать всплески трафика, в рамках исследования предполагается исследовать как трафик с большим количеством всплесков, так и с малым, это объясняет выбор 1% и 99% пульсации. Минимальному и максимальному значениям размера буфера соответствуют значения буфера в ОС Linux.

В таблице 3 представлены параметры, которые фиксировались в ходе эксперимента.

Таблица 3 – Фиксированные параметры

Параметр	Значение
Пропускная способность канала между узлами R и Dest	100 Мб/с
Пропускная способность канала VoIP	1000 Мб/с
Пропускная способность канала TCP	1000 Мб/с
Размер пакета VoIP	218 байт
Скорость генерации трафика VoIP	8 Мб/с

В качестве значения пропускной способности исследуемого канала было выбрано значение 100 Мб/с, поскольку такую скорость предлагают интернет-провайдеры в качестве стандартной для домашнего подключения [30]. Пропускные способности каналов связи для TCP и VoIP потоков были выбраны таким образом, чтобы они не являлись узким местом в исследуемой сети. В качестве размера VoIP-пакета было выбрано значение в 218 байт, что соответствует среднему размеру VoIP-пакета[31]. Скорость генерации трафика была выбрана в соответствии с требованиями для проведения видеоконференции на 7+ человек [32].

4. Составление плана эксперимента. На результаты эксперимента оказывает влияние набор различных факторов, таких как всплески трафика, размер буфера сетевого устройства и др. При этом, каждый из этих факторов может принимать различные значения. Это приводит к необходимости проведения большого количества экспериментов. Для того, чтобы рассмотреть все возможные ситуации и при этом не проводить количество опытов превышающих необходимое значение, было принято решение использовать метод факторного плана эксперимента. Данный метод позволяет реализовать все возможные неповторяющиеся комбинации уровней факторов или независимых переменных, причем каждый фактор принудительно варьируется на заранее выбранном количестве уровней. Для реализации данного метода была построена матрица планирования, которая представлена в таблице 3.

Таблица 3 – Матрица планирования

Номер опыта	Размер буфера	Пульсация onTime	Пульсация offTime	Скорость генерации	Размер пакетов	Вычисленная скорость
1	0	0	0	0	0	y ₁
2	0	0	0	0	1	y ₂
3	0	0	0	1	0	y ₃
.
32	1	1	1	1	1	y ₃₂

В данной таблице значение «0» соответствует минимальному значению аналогичного параметра, представленного в таблице 2, а значение «1» соответствует максимальному значению рассматриваемого параметра. Вычисленная скорость является результатом выполнения эксперимента. Стоит отметить, что на результаты экспериментов может оказывать влияние такой параметр трафика, как соотношение периодов onTime и OffTime. Однако данное соотношение имеет смысл рассматривать в трех вариантах: 1:10, 10:1, 1:1. Так как это не вписывается в рамки двухфакторного плана экспериментов, предлагается использовать приведенный выше план трижды: для каждого из возможных соотношений onTime и offTime.

Описание проведения эксперимента см. В пункте 2.5.

5. Обработка результатов эксперимента и объяснение полученных результатов представлены в главе 3.

2.5 Проведение эксперимента

Постановка задачи эксперимента. Пользователь одновременно выполняет скачивание файла большого объема и совершает видео-звонок (рисунок 6).

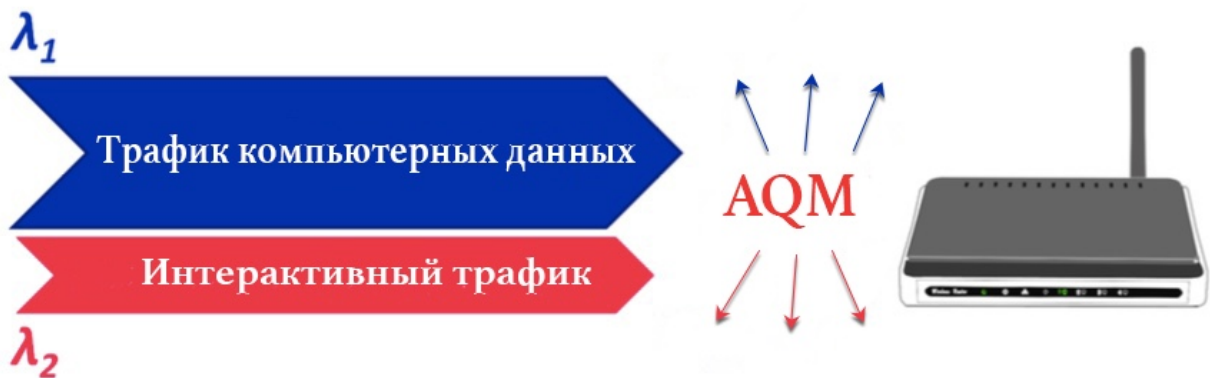


Рисунок 6 – Постановка задачи эксперимента

В результате в сетевое устройство поступает трафик смешанного типа, к которому применяется алгоритм AQM. Данный алгоритм может оказывать отрицательное влияние на скорость передачи фонового трафика. Поэтому необходимо

выбрать такой алгоритм AQM, который оказывает наименьшее влияние на трафик с заданными параметрами.

Входные параметры эксперимента. Так как на результаты эксперимента оказывает влияние набор различных факторов, каждый из которых может принимать различные значения. При проведении экспериментов были использованы методы планирования эксперимента. Установление параметров происходило следующим образом:

1. Задание исследуемого алгоритма AQM.
2. Установка параметров, которые не требуют варьирования. Данные параметры и их значения приведены в разделе 2.4, таблица 3.
3. Установка соотношения времен onTime и offTime. Осуществляется посредством выбора из следующих вариантов: 1:10, 10:1, 1:1, в таблице 4 приведены значения времени для данных соотношений. Выбранные значения соответствуют наиболее часто встречаемым периодам отправки данных в реальных приложениях [33].

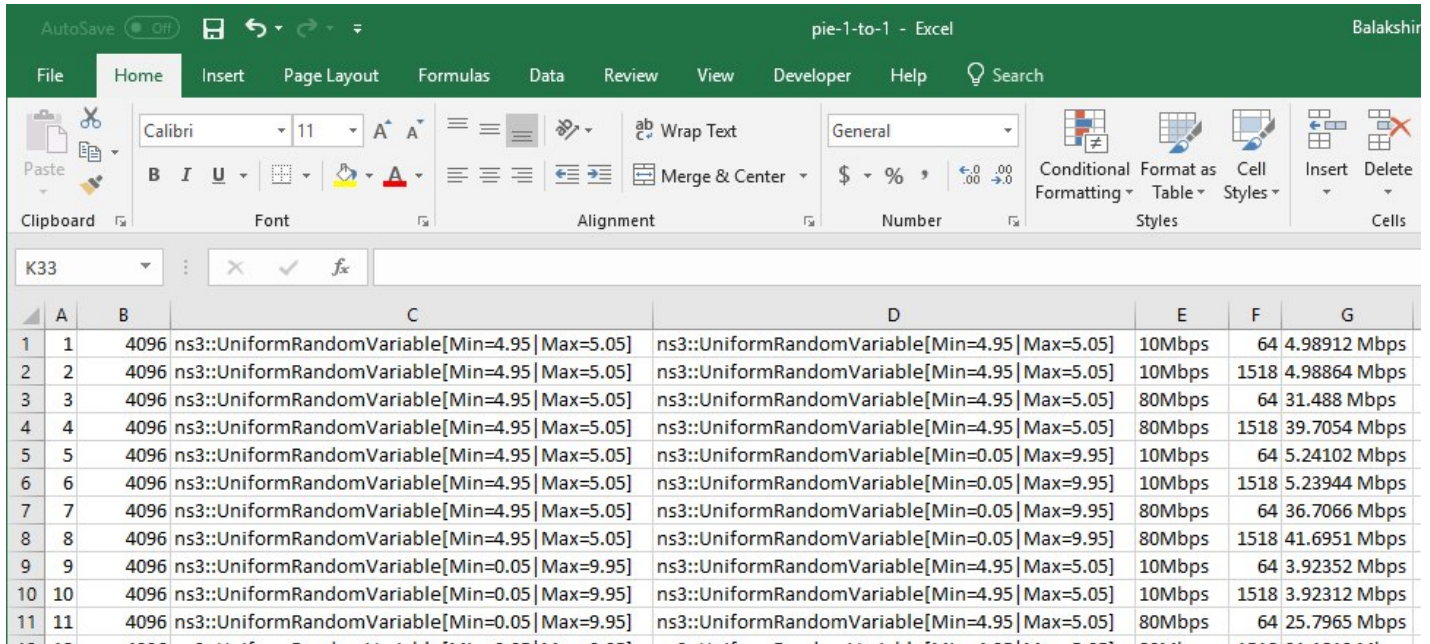
Таблица 4 – Выбор времен для различных соотношений onTime и offTime

Соотношение onTraffic/offTraffic	Устанавливаемое значение onTime	Устанавливаемое значение offTime
1:10	1 с	10 с
10:1	10 с	1 с
1:1	5 с	5 с

4. Установление значений варьируемых параметров. Данные параметры и их значения приведены в разделе 2.4, таблица 2.

Оптимизация запуска модели. Как можно видеть в предыдущем абзаце, набор задаваемых параметров довольно велик. Это приводит к необходимости проведения большого количества экспериментов, а значит многократного изменения значений входных параметров и запуска модели на исполнение. Выполнение этих действий вручную может потребовать значительных временных затрат, поэтому для упрощения задачи была произведена оптимизация запуска имитационной модели.

Для этого был написан скрипт для командной оболочки Shell (приложение 2). Данный скрипт выполняет необходимое количество запусков эксперимента, при разных входных параметрах. Результатом выполнения данного скрипта является таблица значений в формате .CSV (рисунок 7).



	A	B	C	D	E	F	G
1	1	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	10Mbps	64	4.98912 Mbps
2	2	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	10Mbps	1518	4.98864 Mbps
3	3	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	80Mbps	64	31.488 Mbps
4	4	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	80Mbps	1518	39.7054 Mbps
5	5	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	10Mbps	64	5.24102 Mbps
6	6	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	10Mbps	1518	5.23944 Mbps
7	7	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	80Mbps	64	36.7066 Mbps
8	8	4096	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	80Mbps	1518	41.6951 Mbps
9	9	4096	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	10Mbps	64	3.92352 Mbps
10	10	4096	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	10Mbps	1518	3.92312 Mbps
11	11	4096	ns3::UniformRandomVariable[Min=0.05 Max=9.95]	ns3::UniformRandomVariable[Min=4.95 Max=5.05]	80Mbps	64	25.7965 Mbps

Рисунок 7 – Результат выполнения скрипта, оптимизирующего запуск модели

2.6 Выводы по главе 2

1. Для создания имитационной модели была использована среда моделирования ns-3, поскольку она обладает следующим рядом преимуществ: открытый исходный код всех компонентов, свободное распространение данной среды моделирования, работоспособность в свободно распространяемой операционной системе Linux, а также существующая база разработанных моделей в области исследования. [23].
2. Реализована вся необходимая функциональность для проведения имитационных экспериментов согласно поставленной в первой главе задаче.
3. При подготовке к проведению опытов в реализованной модели, было осуществлено планирование эксперимента. Это позволило рассмотреть все

возможные ситуации и при этом не проводить количество опытов, превышающих необходимое значение.

4. Проведение ряда опытов в имитационной модели требует многократного изменения значений входных параметров и запуска модели на исполнение. Выполнение этих действий вручную может потребовать значительных временных затрат, поэтому для упрощения задачи была произведена оптимизация запуска имитационной модели. Для этого был написан скрипт в командной оболочке Shell.

3 АНАЛИЗ РЕЗУЛЬТАТОВ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

3.1 Верификация модели

Так как подключаемые алгоритмы AQM являются частью библиотеки ns-3, верификация их работы не требуется. Для верификации построенной топологии имитационной модели был произведен сбор статистики во время выполнения имитационного моделирования и осуществлено сравнение полученных данных с ожидаемыми. Для этого были выполнены следующие действия.

1. Для упрощения анализа полученных результатов был отключен алгоритм AQM, а генерация трафика осуществлялась только на одном узле из двух (рисунок 8). Второй узел подключен аналогичным образом, что позволяет ожидать аналогичные результаты при использовании его в верификации.

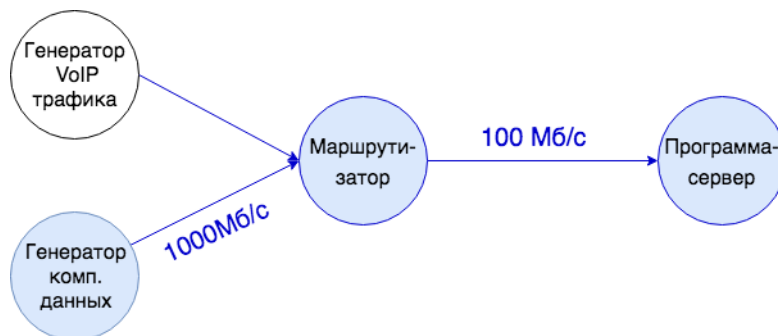


Рисунок 8 – Топология для верификации

2. Для сбора характеристик в модель был добавлен модуль FlowMonitor. Данный модуль позволяет отслеживать передачу данных и собирать различные характеристики для каждого потока данных непосредственно в ходе моделирования. В качестве собираемых характеристик были использованы количество отправленных генератором трафика пакетов, количество пакетов, доставленных в узел-получатель, скорость передачи данных, а также скорость приема полезных данных.
3. При сборе данных значение пропускной способности было фиксировано и составляло 100 Мб/с. Значение скорости генерации трафика варьировалось,

позволяя исследовать работу смоделированной сети при разных значениях загрузки.

4. Описанные эксперименты проводились несколько раз при использовании разных номеров генераторов, что позволило получить различные данные для расчета доверительного интервала. Используемые номера генераторов: 0 (по умолчанию), 365, 2018, 525600.
5. По результатам проведенных экспериментов был построен график зависимости доли доставленных в узел-получатель пакетов от загрузки канала от (рисунок 9), а также график показывающий изменение скорости генерации, скорости передачи данных, а также скорости приема полезных данных в зависимости от загрузки канала (рисунок 10). Так как доверительный интервал имеет малые значения, на графиках его отобразить не удалось.

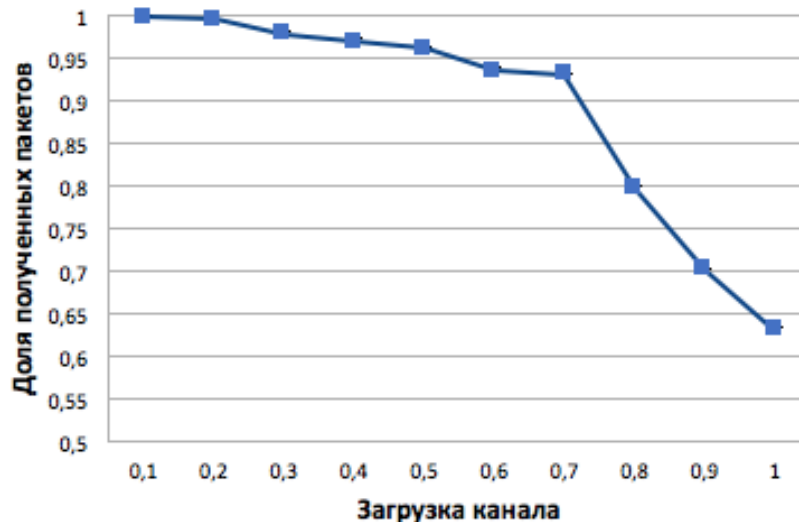


Рисунок 9 – Полученные при верификации результаты

На представленном рисунке можно видеть, что с увеличением загрузки канала количество полученных пакетов уменьшается. Данный результат является ожидаемым, поскольку при увеличении загрузки пакеты встают в очереди, пропускная способность потока уменьшается и не все отправленные пакеты успевают достичь узла-получателя за время симуляции. Кроме того, при построении графика

загрузка была рассчитана как количество отправленных данных, деленое на пропускную способность канала связи. При этом, реальная загрузка канала связи имеет большие значения, так как на нее оказывают дополнительное влияние служебные данные.

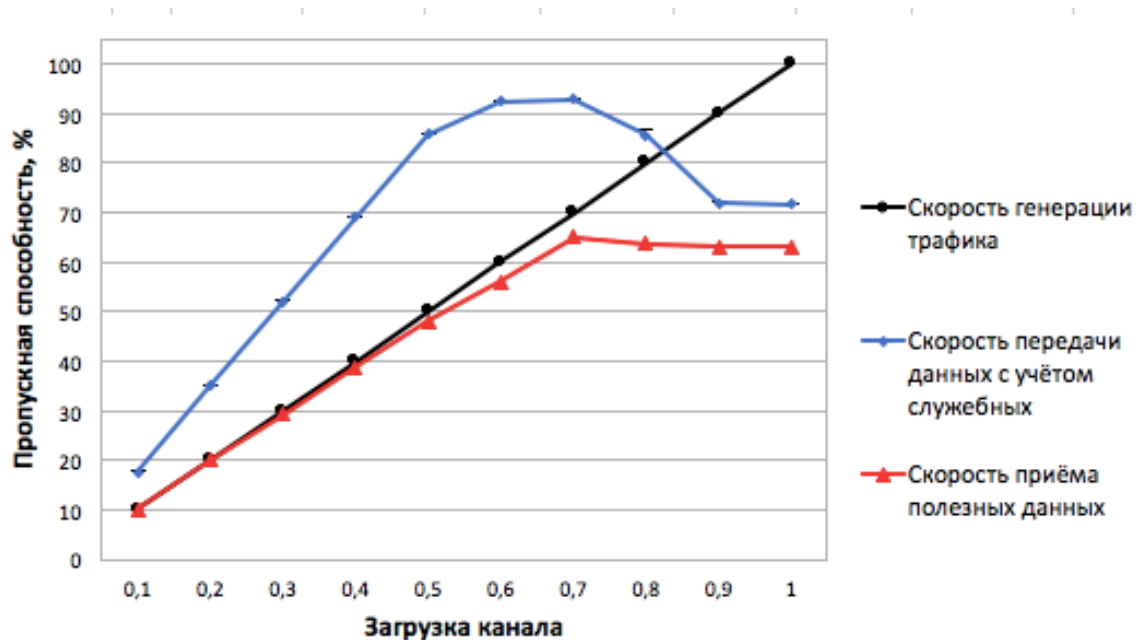


Рисунок 10 – Зависимость пропускной способности от загрузки канала

На представленном изображении видно снижение скорости приёма полезных данных при достижении загрузки в 70%, это связано с задержкой пакетов в очередях и с особенностями расчета загрузки канала (количество отправленных данных, деленое на пропускную способность канала связи). Также можно видеть, что скорость передачи данных превышает скорость генерации. Это связано с тем, что при расчете скорости передачи данных учитывается передача служебных данных.

3.2 Анализ полученных результатов

В таблице 5 представлены результаты всех экспериментов, как для алгоритма AQM CoDel, так и для алгоритма AQM PIE. В таблице 6 приведено сравнение исследованных алгоритмов, где зеленым цветом отмечены эксперименты, в которых наилучшее значение показал алгоритм CoDel, оранжевым цветом отмечены эксперименты, в которых наилучшим образом показал себя PIE, желтым цветом

отмечены значения, в которых разница составила менее 0,5%. Чтобы получить значения для сравнения алгоритмов, была проведена нормализация дельты между алгоритмами по формуле 3.

$$S = \frac{V_{CoDel} - V_{PIE}}{\max(V_{CoDel}; V_{PIE})} \cdot 100\%, \quad (3)$$

где V_{CoDel} – скорость передачи при использовании алгоритма AQM CoDel, V_{PIE} – скорость передачи при использовании алгоритма AQM PIE.

Так, S, равное 4,6 означает, что V_{CoDel} больше V_{PIE} на 4,6%. А S, равное -11,9 означает, что V_{PIE} больше V_{CoDel} на 11,9%.

Таблица 5 – Результаты проведенных экспериментов

№	Размер буфера, байт	onTime, %	offTime, %	Скорость генерации фонового трафика, Мб/с	Размер пакетов фонового трафика, байт	1-1_CoDel	1-1_PIE	10-1_CoDel	10-1_PIE	1-10_CoDel	1-10_PIE
1	4096	1	1	10	64	9,98426	9,97824	9,898097	9,901056	8,217792	8,239616
2	4096	1	1	10	1518	9,98336	9,97728	9,896568	9,899912	8,20952	8,22624
3	4096	1	1	80	64	62,7898	62,976	77,08503	73,50387	47,32079	47,3869
4	4096	1	1	80	1518	79,4564	79,4108	78,92676	78,9569	63,26848	63,43568
5	4096	1	99	10	64	9,92486	10,48204	9,890287	10,18336	10,217792	10,976064
6	4096	1	99	10	1518	9,92256	10,47888	9,88988	10,18248	10,20952	10,96832
7	4096	1	99	80	64	68,9272	73,4132	69,71393	75,73379	57,56081	69,26799
8	4096	1	99	80	1518	78,8424	83,3902	78,86989	81,19067	63,51928	85,32216
9	4096	99	1	10	64	8,44364	7,84704	9,620798	9,626496	7,101184	7,270208
10	4096	99	1	10	1518	8,44208	7,84624	9,619016	9,625704	7,0996	7,25648
11	4096	99	1	80	64	63,523	51,593	68,99189	66,86581	48,2976	51,33854
12	4096	99	1	80	1518	67,1324	62,3626	76,70806	76,75987	48,32224	55,0924
13	4096	99	99	10	64	8,44364	9,00326	9,386641	9,903586	7,101184	8,631744
14	4096	99	99	10	1518	8,44208	9,00144	9,384936	9,903256	7,0996	8,62752
15	4096	99	99	80	64	60,0366	58,2536	70,12082	69,85924	48,53762	52,61553
16	4096	99	99	80	1518	67,0472	71,6102	74,85874	78,98022	48,57304	65,99384
17	16777216	1	1	10	64	9,98426	9,97824	9,898097	9,901056	8,217792	8,239616
18	16777216	1	1	10	1518	9,98336	9,97728	9,896568	9,899912	8,20952	8,22624
19	16777216	1	1	80	64	76,7806	74,7618	76,76768	75,60762	58,74338	64,45615
20	16777216	1	1	80	1518	78,9124	79,0156	78,6258	78,7127	62,68328	62,8672
21	16777216	1	99	10	64	9,92486	10,48204	9,890287	10,18336	10,217792	10,976064
22	16777216	1	99	10	1518	9,92256	10,47888	9,88988	10,18248	10,20952	10,96832
23	16777216	1	99	80	64	75,4604	76,3548	76,79793	75,97942	71,91471	85,976
24	16777216	1	99	80	1518	78,5414	82,5664	78,61744	81,18231	71,24536	87,49576
25	16777216	99	1	10	64	8,44364	7,84704	9,620798	9,626496	7,101184	7,270208
26	16777216	99	1	10	1518	8,44208	7,84624	9,619016	9,625704	7,0996	7,25648
27	16777216	99	1	80	64	64,8462	58,1836	74,42468	73,38353	43,60335	49,49681
28	16777216	99	1	80	1518	66,5882	61,9674	76,4071	76,51578	47,73704	57,8512
29	16777216	99	99	10	64	8,44364	9,00326	9,386641	9,903586	7,101184	8,631744
30	16777216	99	99	10	1518	8,44208	9,00144	9,384936	9,903256	7,0996	8,62752
31	16777216	99	99	80	64	64,0902	67,5938	72,5571	75,91199	44,56574	44,07953
32	16777216	99	99	80	1518	66,7462	70,7864	74,60629	78,97186	46,29912	68,75264

Таблица 6 – Сравнение исследованных алгоритмов AQM

№	Размер буфера, байт	onTime, %	offTime, %	Скорость генерации фонового трафика, Мб/с	Размер пакетов фонового трафика, байт	Нормализованная дельта между алгоритмами CoDel и PIE, %		
						OnTime / OffTime = 1	OnTime / OffTime = 10	OnTime / OffTime = 0.1
1	4096	1	1	10	64	0,06	-0,03	-0,26
2	4096	1	1	10	1518	0,06	-0,03	-0,20
3	4096	1	1	80	64	-0,30	4,65	-0,14
4	4096	1	1	80	1518	0,06	-0,04	-0,26
5	4096	1	99	10	64	-5,32	-2,88	-6,91
6	4096	1	99	10	1518	-5,31	-2,87	-6,92
7	4096	1	99	80	64	-6,11	-7,95	-16,90
8	4096	1	99	80	1518	-5,45	-2,86	-25,55
9	4096	99	1	10	64	7,07	-0,06	-2,32
10	4096	99	1	10	1518	7,06	-0,07	-2,16
11	4096	99	1	80	64	18,78	3,08	-5,92
12	4096	99	1	80	1518	7,11	-0,07	-12,29
13	4096	99	99	10	64	-6,22	-5,22	-17,73
14	4096	99	99	10	1518	-6,21	-5,23	-17,71
15	4096	99	99	80	64	2,97	0,37	-7,75
16	4096	99	99	80	1518	-6,37	-5,22	-26,40
17	16777216	1	1	10	64	0,06	-0,03	-0,26
18	16777216	1	1	10	1518	0,06	-0,03	-0,20
19	16777216	1	1	80	64	2,63	1,51	-8,86
20	16777216	1	1	80	1518	-0,13	-0,11	-0,29
21	16777216	1	99	10	64	-5,32	-2,88	-6,91
22	16777216	1	99	10	1518	-5,31	-2,87	-6,92
23	16777216	1	99	80	64	-1,17	1,07	-16,35
24	16777216	1	99	80	1518	-4,87	-3,16	-18,57
25	16777216	99	1	10	64	7,07	-0,06	-2,32
26	16777216	99	1	10	1518	7,06	-0,07	-2,16
27	16777216	99	1	80	64	10,27	1,40	-11,91
28	16777216	99	1	80	1518	6,94	-0,14	-17,48
29	16777216	99	99	10	64	-6,22	-5,22	-17,73
30	16777216	99	99	10	1518	-6,21	-5,23	-17,71
31	16777216	99	99	80	64	-5,18	-4,42	1,09
32	16777216	99	99	80	1518	-5,71	-5,53	-32,66

0,06%	-0,49% ... +0,49%
4,65%	+0,5% ... +4,99%
7,07%	+5% ... +9,99%
17,78%	+10% ...
-2,88%	-4,99 ... -0,5%
-5,32%	-9,99% ... -5%
-16,90%	... -10%

В полученной таблице можно явно увидеть области, в которых наилучшим образом справляется тот или иной алгоритм, либо полученные характеристики при их использовании имеют близкие значения. Суммируя результаты по полученным данным, можно сделать следующие выводы:

1. Наибольшее влияние на результаты выполнения экспериментов оказывают периоды onTime и offTime: их соотношение и сила пульсации значений.
2. При малых всплесках трафика (пульсация onTime и offTime – 1%) алгоритмы AQM CoDel и PIE оказывают схожее влияние на скорость передачи данных.

3. При высоких значениях всплесков трафика (пульсация onTime и offTime – 99%) явное преимущество имеет алгоритм PIE, особенно это заметно при соотношении 1:10 onTime и offTime.
4. При соотношении времен onTime и offTime 1:1, высокой пульсации onTime (99%) и низкой пульсации offTime (1%) заметное преимущество имеет алгоритм CoDel.
5. При соотношении времен onTime и offTime 10:1, высокой пульсации onTime (99%) и низкой пульсации offTime (1%) алгоритмы CoDel и PIE оказывают схожее влияние на скорость передачи данных.
6. При соотношении времен onTime и offTime 10:1, низкой пульсации offTime (1%), высокой скорости генерации потокового трафика и малом значении размера пакета такого трафика однозначное преимущество имеет алгоритм CoDel.
7. Наибольшая разница в преимуществе алгоритма CoDel получена при соотношении времен onTime и offTime 1:1 при высокой пульсации onTime (99%), низкой пульсации offTime (1%), высокой скорости генерации потокового трафика и малом значении размера пакета такого трафика.
8. Наибольшая разница в преимуществе алгоритма PIE получена при соотношении времен onTime и offTime 1:10 при высокой пульсации offTime (99%), высокой скорости генерации потокового трафика и большом значении размера пакета такого трафика.
9. Согласно проведенным экспериментам алгоритм AQM PIE обеспечивает лучшую скорость передачи трафика в 55% случаев, алгоритм AQM CoDel – в 17% случаев, а в 28% экспериментов рассматриваемые алгоритмы оказывают схожее влияние на скорость передачи данных.

3.3 Рекомендации по выбору алгоритма активного управления очередью

В случае отсутствия времени или возможности исследования особенностей генерации трафика компьютерных данных, в том числе в условиях необходимости оперативной передачи этих данных, предпочтительнее использовать алгоритм AQM PIE. А в случае наличия времени и возможности рекомендуется воспользоваться следующими положениями:

1. При отсутствии заметных всплесков трафика современные алгоритмы AQM не оказывают значительного влияния на скорость передачи данных, и можно выбрать любой из вышеописанных.
2. При значениях периодов времени, в которых генерация трафика не производится, значительно превышающих значения периодов времени, в которых генерация трафика компьютерных данных производится, однозначно целесообразно использовать алгоритм AQM PIE.
3. При значениях периодов времени, в которых генерация трафика производится, примерно равных значениям периодов времени, в которых генерация трафика компьютерных данных не производится, но при этом имеющих заметные изменения периода времени, в которых происходит генерация, целесообразно использовать алгоритм AQM CoDel.
4. При значениях периодов времени, в которых генерация трафика компьютерных данных производится, значительно превышающих значения периодов времени, в которых генерация не производится, можно выбрать любой из вышеперечисленных алгоритмов, так как ни один из них не дает однозначного преимущества.

3.4 Выводы по главе 3

1. Выполнена успешная верификация реализованной имитационной модели. Для этого был произведен сбор статистики во время выполнения имитационного моделирования и осуществлено сравнение полученных данных с ожидаемыми.

2. Составлены таблицы для наглядного представления результатов имитационного моделирования. Произведен анализ полученных данных. Согласно проведенным экспериментам алгоритм AQM PIE обеспечивает лучшую скорость передачи трафика в 55% случаев, алгоритм AQM CoDel – в 17% случаев, а в 28% экспериментов рассматриваемые алгоритмы оказывают схожее влияние на скорость передачи данных. Также были выявлены параметры трафика, оказывающие наибольшее влияния на работу алгоритмов AQM. Указаны конкретные случаи, в которых преимущество имеет тот или иной алгоритм.
3. На основании результатов экспериментов сформированы рекомендации по выбору алгоритма AQM.

ЗАКЛЮЧЕНИЕ

В рамках выполнения данной работы получены следующие результаты:

1. Выполнен обзор исследований в области активного управления очередью, в ходе которого выявлен ряд недостатков среди аналогичных исследований.
2. Изучен терминологический базис в исследуемой области. В ходе выполнения работы были определены несоответствия терминов русско- и англоязычной литературы.
3. Реализована имитационная модель в системе моделирования ns-3 с помощью языка программирования C++. Полученная модель отвечает следующими требованиям: поддержка работы с современными алгоритмами AQM, поддержка передачи данных по TCP и UDP протоколам, возможность задания функциональных, нагрузочных и структурных параметров сетевого устройства, осуществление расчета средней скорости передачи данных исследуемого потока.
4. Выполнена успешная верификация реализованной имитационной модели. Для этого был произведен сбор статистики во время выполнения имитационного моделирования и осуществлено сравнение полученных данных с ожидаемыми.
5. Осуществлено планирование эксперимента, которое позволило рассмотреть все возможные ситуации и при этом не проводить количество опытов превышающих необходимое значение.
6. В соответствии с построенным планом был выполнен ряд экспериментов, позволяющих определить влияние современных алгоритмов активного управления очередью на среднюю скорость передачи фоновых трафиков. Согласно проведенным экспериментам алгоритм AQM PIE обеспечивает лучшую скорость передачи трафика в 55% случаев, алгоритм AQM CoDel – в 17% случаев, а в 28% экспериментов рассматриваемые алгоритмы оказывают схожее влияние на скорость передачи данных.

7. На основании результатов экспериментов сформированы рекомендации по выбору алгоритма AQM.

На основе полученных результатов можно сформулировать следующие выводы:

1. Для борьбы с излишней сетевой буферизацией используются различные алгоритмы AQM. Однако данные алгоритмы могут оказывать негативное влияние на передачу трафика компьютерных данных. Поэтому необходимо знать при использовании какого алгоритма обеспечивается наибольшая скорость передачи трафика данных, наиболее часто передаваемых в настраиваемой сети.
2. Выявлено, что при выборе алгоритма AQM, необходимо определить параметры наиболее часто передаваемого в сети трафика. При этом ключевыми показателями будут являться периоды времени, в которые производится генерация трафика (onTime) и в которые генерация трафика не производится (offTime): их соотношение и сила пульсации значений.
3. В результате проведения 192 экспериментов определено, что алгоритм AQM PIE обеспечивает лучшую скорость передачи трафика в 55% случаев, алгоритм AQM CoDel – в 17% случаев, а в 28% экспериментов рассматриваемые алгоритмы оказывают схожее влияние на скорость передачи данных.
4. Разработанная имитационная модель может быть использована системными администраторами, а также студентами в образовательном процессе как инструмент для исследования алгоритмов AQM.

Дальнейшее развитие имитационной модели и проведенного исследования возможно по следующим направлениям:

1. Реализация автоматического расчета доверительного интервала, при проведении моделирования.
2. Проведение экспериментов для более широкого набора типовых случаев. Например, для различных сочетаний трафика и различных значений пропускной способности.

3. Увеличение количества поддерживаемых в модели входных потоков трафика с 2 до нескольких или произвольного количества N , задаваемого пользователем.
4. Реализация и добавление в модель других современных алгоритмов AQM, а также модификаций исследованных алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Floyd, S. Random Early Detection gateways for Congestion Avoidance / S. Floyd, V. Jacobson // IEEE/ACM Transactions on Networking – 1993. – Vol 1. – Iss 4. – P. 397 – 413.
2. Gettys, J. The criminal mastermind: bufferbloat [Электронный ресурс] / J. Gettys // – Режим доступа: <http://gettys.wordpress.com/2010/12/03/introducing-the-criminal-mastermind-bufferbloat>
3. Nichols, K. Controlling Queue Delay / K. Nichols, V. Jacobson // Acmqueue – 2012. – May 6. – Vol 10. – Iss 5.
4. Pan, R. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem / R. Pan // IEEE Conference on High Performance Switching and Routing – 2013.
5. Bufferbloat [Электронный ресурс] – Режим доступа: www.bufferbloat.com
6. RFC 8289 Internet Engineering Task Force (IETF). Controlled Delay Active Queue Management / K. Nichols, V. Jacobson, A. McGregor, J. Iyengar – 2018.
7. RFC 8033 Internet Engineering Task Force (IETF). Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem / R. Pan, P. Natarajan, F. Baker, G. White – 2017.
8. Алиев Т.И. Основы моделирования дискретных систем. – СПб: СПбГУ ИТМО, 2009. – 363 с.
9. Traffic Control HOWTO: Components of Linux Traffic Control [Электронный ресурс] – Режим доступа: <http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/components.html> – 2013.
10. Traffic Control HOWTO: Traditional Elements of Traffic Control [Электронный ресурс] – Режим доступа: <http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/elements.html> – 2013.
11. Queuing Disciplines: Order of Packet Transmission and Dropping [Электронный ресурс] – Режим доступа: <http://www.eng.tau.ac.il/~netlab/resources/booklet/lab9> – 2006.

12. Kuhn, N. Revisiting Old Friends: Is CoDel Really Achieving What RED Cannot? / N. Kuhn, E. Lochin, O. Mehani // ACM SIGCOMM Capacity Sharing Workshop – 2014. August 18.
13. Kuhn, N. Operating ranges, tunability and performance of CoDel and PIE / N. Kuhn, D. Ros, A. Bagayoko, C. Kulatunga, G. Fairhurst, N. Khademie // Computer Communications – 2017. – May 17. – Vol 103. P. 74 – 82.
14. Özekes, S. Evaluation of active queue management algorithms / S. Özekes // İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi – 2005. – Vol 4. – Iss 7. P. 123 – 140.
15. Балакшина, Ю.А. Автоматизация подбора оптимальной дисциплины обслуживания при передаче VoIP-трафика / Ю.А. Балакшина, В.В. Соснин // Научно-технический вестник информационных технологий, механики и оптики – 2016. – Т. 16. – № 6(106). – С. 1084 – 1090.
16. Балакшина, Ю.А. Дисциплины обслуживания для управления трафиком в операционной системе Linux / Ю.А. Балакшина // Сборник трудов VIII научно-практической конференции молодых ученых «Вычислительные системы и сети (Майоровские чтения)» – 2017. – С. 60 – 63.
17. Балакшина, Ю.А. Исследование дисциплины обслуживания fq_codel / Ю.А. Балакшина // Сборник тезисов докладов конгресса молодых ученых. Электронное издание – 2017.
18. Балакшина, Ю.А. Выбор и обоснование архитектуры учебно-экспериментального комплекса имитационного моделирования современных дисциплин обслуживания в компьютерных сетях / Ю.А. Балакшина // Сборник тезисов докладов конгресса молодых ученых. – 2018.
19. Робачевский, А. В поисках утраченного качества [Электронный ресурс] / А. Робачевский // – Режим доступа: <http://www.ripn.net/articles/QoS/>.
20. Gettys, J. Diagnosing bufferbloat [Электронный ресурс] / J. Gettys // – Режим доступа: <http://gettys.wordpress.com/2012/02/20/diagnosing-bufferbloat/>.

21. Kochher, S. A Review on Active and Passive Queuing Techniques / S. Kochher, M. Sanghal, R. Kochher, G. Singh // International Journal for Science and Emerging Technologiest with Latest Trends – 2014. – June 17. – Vol 15. – P. 16 – 22.
22. Попова, Д.А. Исследование свойств дисциплины обслуживания wfq в компьютерных сетях / Попова Дарья Андреевна. – СПб., 2016. – 58 с.
23. Соснин, В. В. Моделирование маршрутизатора с поддержкой методов QoS в среде ns-3 / В.В Соснин, Д. Н. Шинкарук // Сборник трудов молодых ученых и сотрудников кафедры – 2011. – С. 50 – 55.
24. Traffic Control Layer [Электронный ресурс] – Режим доступа: <https://www.nsnam.org/docs/models/html/traffic-control.html>.
25. RFC 7928 AQM Characterization Guidelines / N. Kuhn, P. Natarajan, D. Ros, N. Khademi – 2016.
26. Hayes, D. Common TCP Evaluation Suite [Электронный ресурс] / D. Hayes, D. D. Ros, L. Andrew, S. Floyd // – Режим доступа: <https://tools.ietf.org/id/draft-irtf-icrg-tcpeval-01.html>. – 2015.
27. OnOffHelper Class Reference [Электронный ресурс] – Режим доступа: https://www.nsnam.org/doxygen/classns3_1_1_on_off_helper.html.
28. Красовский Г.И., Филаретов Г.Ф. Планирование эксперимента. – Минск: Изд-во БГУ, 1982. – 302 с.
29. Кунегин, С.В. Описание технологии Ethernet [Электронный ресурс] / С.В. Кунегин // – Режим доступа: <http://kunegin.com/ref1/eth1/formats.htm>.
30. Тариф «Домашний Интернет в комплекте 2в1» [Электронный ресурс] – Режим доступа: <https://spb.rt.ru/>.
31. Полоса пропускания для VOIP разговоров [Электронный ресурс] – Режим доступа: (<https://planetcalc.ru/3144/>).
32. How much bandwidth does Skype need? [Электронный ресурс] – Режим доступа: <https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need>.

33. Varet, A. How to generate realistic network traffic / A. Varet, N. Larrieu // IEEE COMPSAC 38th Annual International Computers, Software & Applications Conference – 2014.

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ,
СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ**

ДО – дисциплина обслуживания

AQM – active queue management

CoDel – controlling delay

FIFO – first in, first out

PIE – proportional integral controller enhanced

QoS – quality of service

RED – random early detection

RTT – round-trip time

TCP – Transmission Control Protocol

QDisc – queueing discipline

UDP – User Datagram Protocol

VoIP – voice over IP

ПРИЛОЖЕНИЕ А. Код разработанной имитационной модели

```

#include <cstdint>
#include <string>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/traffic-control-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-flow-classifier.h"
// #include "ns3/packet-sink.h"

// Network topology
//
//  VoIP(n0)    --+
//  192.168.1.1  |
//              |
//              192.168.1.2
//              gateway(n2) / 192.168.3.1  --- 192.16.3.2 / sink(n3)
//              192.168.2.2
//              |
//  FTP(n1)     |
//  192.168.2.1 --+
//
using namespace ns3;

NS_LOG_COMPONENT_DEFINE("Diplom");

static std::string opt_queue_disc = "codel";

// Channels' speed
static std::string opt_voip_channel_rate = "1000Mbps";
static std::string opt_voip_channel_delay = "1ms";
static std::string opt_ftp_channel_rate = "1000Mbps";
static std::string opt_ftp_channel_delay = "1ms";
static std::string opt_cable_bandwidth = "100Mbps";
static std::string opt_cable_prop_delay = "100ms";

static std::string opt_voip_rate = "8Mbps";
static std::uint32_t opt_voip_packet_size = 218;

static std::string opt_ftp_rate = "80Mbps";
static std::uint32_t opt_ftp_packet_size = 1518;
static std::string opt_ftp_ontime = "ns3::UniformRandomVariable";

```

```

static std::string opt_ftp_offtime = "ns3::UniformRandomVariable";

static std::uint32_t opt_buffer_size = 4096;

static std::uint16_t voip_port = 12345;
static std::uint16_t ftp_port = 20;
static double simulation_time = 41.0;

int
main(int argc, char* argv[])
{
    CommandLine cmd;

    cmd.AddValue("queue-disc", "Queue disc type (codel, pie)",
opt_queue_disc);
    cmd.AddValue("voip-channel-rate", "VoIP channel rate",
opt_voip_channel_rate);
    cmd.AddValue("voip-channel-delay", "VoIP channel delay",
opt_voip_channel_delay);
    cmd.AddValue("voip-rate", "VoIP data rate", opt_voip_rate);
    cmd.AddValue("voip-packet-size", "VoIP packet size",
opt_voip_packet_size);
    cmd.AddValue("ftp-channel-rate", "TCP channel rate",
opt_ftp_channel_rate);
    cmd.AddValue("ftp-channel-delay", "TCP channel delay",
opt_ftp_channel_delay);
    cmd.AddValue("ftp-rate", "TCP data rate", opt_ftp_rate);
    cmd.AddValue("ftp-packet-size", "TCP packet size",
opt_ftp_packet_size);
    cmd.AddValue("cable-bandwidth", "Cable bandwidth",
opt_cable_bandwidth);
    cmd.AddValue("cable-prop-delay", "Cable propagation delay",
opt_cable_prop_delay);
    cmd.AddValue("ftp-ontime", "TCP OnOffHelper.OnTime attribute",
opt_ftp_ontime);
    cmd.AddValue("ftp-offtime", "TCP OnOffHelper.OffTime
attribute", opt_ftp_offtime);
    cmd.AddValue("buffer-size", "Buffer size (bytes)",
opt_buffer_size);
    cmd.Parse(argc, argv);

    NS_LOG_INFO("Create nodes");
    NodeContainer c;
    c.Create(4);
    NodeContainer n0_n2 = NodeContainer(c.Get(0), c.Get(2));
    NodeContainer n1_n2 = NodeContainer(c.Get(1), c.Get(2));
    NodeContainer n2_n3 = NodeContainer(c.Get(2), c.Get(3));

    InternetStackHelper internet;

```

```

internet.Install(c);

NS_LOG_INFO("Create channels");
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate",
StringValue(opt_voip_channel_rate));
p2p.SetChannelAttribute("Delay",
StringValue(opt_voip_channel_delay));
NetDeviceContainer nd0_nd2 = p2p.Install(n0_n2);

p2p.SetDeviceAttribute("DataRate",
StringValue(opt_ftp_channel_rate));
p2p.SetChannelAttribute("Delay",
StringValue(opt_ftp_channel_delay));
NetDeviceContainer nd1_nd2 = p2p.Install(n1_n2);

p2p.SetDeviceAttribute("DataRate",
StringValue(opt_cable_bandwidth));
p2p.SetChannelAttribute("Delay",
StringValue(opt_cable_prop_delay));
NetDeviceContainer nd2_nd3 = p2p.Install(n2_n3);

TrafficControlHelper tc;
if (opt_queue_disc == "codel") {
    tc.SetRootQueueDisc("ns3::CoDelQueueDisc");
    Config::SetDefault("ns3::CoDelQueueDisc::Mode",
        EnumValue(CoDelQueueDisc::QUEUE_DISC_MODE_BYTES));
    Config::SetDefault("ns3::CoDelQueueDisc::MaxBytes",
        UIntegerValue(opt_buffer_size));
} else if (opt_queue_disc == "pie") {
    tc.SetRootQueueDisc("ns3::PieQueueDisc");
    Config::SetDefault("ns3::PieQueueDisc::Mode",
        EnumValue(PieQueueDisc::QUEUE_DISC_MODE_BYTES));
    Config::SetDefault("ns3::PieQueueDisc::QueueLimit",
        UIntegerValue(opt_buffer_size));
} else {
    NS_ABORT_MSG("Invalid 'queue_disc' value");
}
tc.Install(nd2_nd3);

NS_LOG_INFO("Assign IP addresses");
Ipv4AddressHelper ipv4;
ipv4.SetBase("192.168.1.0", "255.255.255.0");
Ipv4InterfaceContainer i0_i2 = ipv4.Assign(nd0_nd2);

ipv4.SetBase("192.168.2.0", "255.255.255.0");
Ipv4InterfaceContainer i1_i2 = ipv4.Assign(nd1_nd2);

```

```

    ipv4.SetBase("192.168.3.0", "255.255.255.0");
    Ipv4InterfaceContainer i2_i3 = ipv4.Assign(nd2_nd3);

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    ApplicationContainer apps;

    NS_LOG_INFO("Create VoIP applications");
    OnOffHelper voip_app("ns3::UdpSocketFactory",

    Address(InetSocketAddress(i2_i3.GetAddress(1), voip_port)));
        voip_app.SetAttribute("DataRate",
    StringValue(opt_voip_rate));
        voip_app.SetAttribute("PacketSize",
    UIntegerValue(opt_voip_packet_size));
        voip_app.SetAttribute("StartTime", TimeValue(Seconds(1.0)));
        apps.Add(voip_app.Install(c.Get(0)));

    PacketSinkHelper voip_sink("ns3::UdpSocketFactory",

    Address(InetSocketAddress(Ipv4Address::GetAny(), voip_port)));
        voip_sink.SetAttribute("StartTime", TimeValue(Seconds(0.0)));
        apps.Add(voip_sink.Install(c.Get(3)));

    NS_LOG_INFO("Create FTP applications");
    OnOffHelper ftp_app("ns3::TcpSocketFactory",

    Address(InetSocketAddress(i2_i3.GetAddress(1), ftp_port)));
        ftp_app.SetAttribute("DataRate", StringValue(opt_ftp_rate));
        ftp_app.SetAttribute("PacketSize",
    UIntegerValue(opt_ftp_packet_size));
        ftp_app.SetAttribute("OnTime", StringValue(opt_ftp_ontime));
        ftp_app.SetAttribute("OffTime",
    StringValue(opt_ftp_offtime));
        ftp_app.SetAttribute("StartTime", TimeValue(Seconds(1.001)));
        apps.Add(ftp_app.Install(c.Get(1)));

    PacketSinkHelper ftp_sink("ns3::TcpSocketFactory",

    Address(InetSocketAddress(Ipv4Address::GetAny(), ftp_port)));
        ftp_sink.SetAttribute("StartTime", TimeValue(Seconds(0.0)));
        apps.Add(ftp_sink.Install(c.Get(3)));

    NS_LOG_INFO("Install FlowMonitor");
    FlowMonitorHelper flowmon;
    Ptr<FlowMonitor> monitor = flowmon.InstallAll();

```

```

NS_LOG_INFO("Run simulation");
Simulator::Stop(Seconds(simulation_time));
Simulator::Run();
NS_LOG_INFO("Stop simulation");

    monitor->CheckForLostPackets();
    Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(
    flowmon.GetClassifier());
    FlowMonitor::FlowStatsContainer stats = monitor-
>GetFlowStats();
    /* for (const auto& iter : stats) {
        Ipv4FlowClassifier::FiveTuple t = classifier-
>FindFlow(iter.first);

        std::cout << "Flow " << iter.first << " (" <<
t.sourceAddress <<
            " -> " << t.destinationAddress << ")\n";

        std::cout << " Tx Packets: " << iter.second.txPackets <<
"\n";
        std::cout << " Tx Bytes: " << iter.second.txBytes <<
"\n";
        std::cout << " TxOffered: "
            << (iter.second.txBytes * 8.0 / 1000 / 1000) /
(simulation_time - 1.0)
            << " Mbps\n";
        std::cout << " Rx Packets: " << iter.second.rxPackets <<
"\n";
        std::cout << " Rx Bytes: " << iter.second.rxBytes <<
"\n";
        std::cout << " RxOffered: "
            << (iter.second.rxBytes * 8.0 / 1000 / 1000) /
(simulation_time - 1.0)
            << " Mbps\n";
        std::cout << " Throughput: "
            << (100 * iter.second.rxBytes / iter.second.txBytes)
<< "%\n";
    }
*/

    Ptr<PacketSink> sink1 = DynamicCast<PacketSink>(apps.Get(3));
    std::cout << "Real Total Rx from TCP, measured by sink: "
        << sink1->GetTotalRx() << "\n";
    std::cout << "Real TCP Throughput, measured by sink: "
        << (sink1->GetTotalRx() * 8.0 / 1000 / 1000) /
(simulation_time - 1.0)

```



```
        << " Mbps\n";

    Simulator::Destroy();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б. Скрипт для автоматизации запуска модели

```
#!/bin/bash -e

EXE_FILENAME=diplom-07

# Static parameters

CABLE_BANDWIDTH="100Mbps"
CABLE_DELAY="100ms"
FTP_CHANNEL_RATE="1000Mbps"
FTP_CHANNEL_DELAY="1ms"
VOIP_CHANNEL_RATE="1000Mbps"
VOIP_CHANNEL_DELAY="1ms"
VOIP_PACKET_SIZE=218
VOIP_RATE="8Mbps"

# Common variable parameters

BUFFER_SIZE="4096 16777216"
DATA_RATE="10Mbps 80Mbps"
PACKET_SIZE="64 1518"

# Variable parameters

ON_TIME_1_1="ns3::UniformRandomVariable[Min=4.95|Max=5.05]
ns3::UniformRandomVariable[Min=0.05|Max=9.95]"
OFF_TIME_1_1="ns3::UniformRandomVariable[Min=4.95|Max=5.05]
ns3::UniformRandomVariable[Min=0.05|Max=9.95]"

ON_TIME_10_1="ns3::UniformRandomVariable[Min=0.99|Max=1.01]
ns3::UniformRandomVariable[Min=0.01|Max=1.99]"
OFF_TIME_10_1="ns3::UniformRandomVariable[Min=9.9|Max=10.1]
ns3::UniformRandomVariable[Min=0.1|Max=19.9]"

ON_TIME_1_10="ns3::UniformRandomVariable[Min=9.9|Max=10.1]
ns3::UniformRandomVariable[Min=0.1|Max=19.9]"
OFF_TIME_1_10="ns3::UniformRandomVariable[Min=0.99|Max=1.01]
ns3::UniformRandomVariable[Min=0.01|Max=1.99]"

do_test() {
    local QUEUE_DISC=$1
    local ON_TIME=$2
    local OFF_TIME=$3

    local n=1

    for buffer_size in $BUFFER_SIZE; do
        for on_time in $ON_TIME; do
```

```

for off_time in $OFF_TIME; do
  for data_rate in $DATA_RATE; do
    for packet_size in $PACKET_SIZE; do

      local opts="--queue-disc=$QUEUE_DISC"
      opts+=" --voip-channel-rate=$VOIP_CHANNEL_RATE"
      opts+=" --voip-channel-delay=$VOIP_CHANNEL_DELAY"
      opts+=" --voip-packet-size=$VOIP_PACKET_SIZE"
      opts+=" --voip-rate=$VOIP_RATE"
      opts+=" --ftp-channel-rate=$FTP_CHANNEL_RATE"
      opts+=" --ftp-channel-delay=$FTP_CHANNEL_DELAY"
      opts+=" --cable-bandwidth=$CABLE_BANDWIDTH"
      opts+=" --cable-prop-delay=$CABLE_DELAY"

      opts+=" --buffer-size=$buffer_size"
      opts+=" --ftp-rate=$data_rate"
      opts+=" --ftp-packet-size=$packet_size"

      opts+=" --ftp-ontime=$on_time"
      opts+=" --ftp-offtime=$off_time"

      echo "[ $n]    ./waf    --run    \"scratch/$EXE_FILENAME
$opts\" 1>&2
      #./waf --run "scratch/$EXE_FILENAME"

      local                                     throughput=`ns-3-
dev/build/scratch/$EXE_FILENAME $opts | egrep -o "[0-9.]+ Mbps"`
      echo
      "$n,$buffer_size,$on_time,$off_time,$data_rate,$packet_size,$thro
ughput"

      n=$((n+1))
      echo 1>&2

    done
  done
done
done
done
}

# main

if [ $# -ne 1 ]; then
  echo "Usage: batch-test.sh <code|pie>" 1>&2
  exit 1
fi

```

```
queue_disc=$1
```

```
echo "1:1 test set"
```

```
do_test $queue_disc "$ON_TIME_1_1" "$OFF_TIME_1_1" | tee
```

```
$queue_disc-1-to-1.csv
```

```
echo
```

```
echo "10:1 test set"
```

```
do_test $queue_disc "$ON_TIME_10_1" "$OFF_TIME_10_1" | tee
```

```
$queue_disc-10-to-1.csv
```

```
echo
```

```
echo "1:10 test set"
```

```
do_test $queue_disc "$ON_TIME_1_10" "$OFF_TIME_1_10" | tee
```

```
$queue_disc-1-to-10.csv
```

```
echo
```