

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей

УТВЕРЖДАЮ

Заведующий кафедрой
прикладной информатики и
теории вероятностей
д.т.н., профессор
_____ К. Е. Самуйлов
«__» _____ 20__ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА
на тему
«Моделирование модуля активного управления трафиком сети передачи
данных»

Выполнил
Студент группы НФИбд-01-18
_____ С. М. Наливайко
«__» _____ 20__ г.

Руководитель
доцент кафедры прикладной информа-
тики и теории вероятностей
к.ф.-м.н., доцент
_____ А. В. Королькова

Москва 2022

Содержание

Список используемых сокращений	6
Англоязычные сокращения	6
Введение	7
1. Моделирование сетей в среде Mininet и анализ их производи- тельности	10
1.1. Обзор исследований в области моделирования в среде Mininet .	10
1.2. Обзор исследований в области анализа производительности се- тей и сетевых компонентов в среде Mininet	12
1.3. Алгоритмы управления перегрузками в сети	14
2. Исследование возможностей моделирования сетей и измерения сетевых характеристик в Mininet	17
2.1. Mininet	17
2.1.1. Общие сведения	17
2.1.2. Создание простой сети с помощью Python и модуля Mininet	18
2.1.3. MiniEdit	19
2.2. Генерация и измерение сетевого трафика с помощью утилиты iPerf3	21
2.2.1. Общие сведения	21
2.2.2. Тестирование пропускной способности с помощью iPerf3	22
2.3. Использование утилиты iproute2 для настройки интерфейсов сетевых элементов	24
2.3.1. Общие сведения	24
2.3.2. Утилита tc	25
3. Моделирование модуля активного управления трафиком сети передачи данных	28
3.1. Создание модуля для среды Mininet	28
3.2. Тестирование программного модуля	31
3.3. Пример простой сети с несколькими хостами	39
Заключение	46
Список литературы	49
A. Класс NetStatsPlotter	52
B. Класс Monitor	54
C. Класс CustomTopology	57

Список иллюстраций

2.1. Запуск простой сети и проверка достижимости узлов	20
2.2. Редактирование конфигурации хостов сети	20
2.3. Соединение элементов сети	21
2.4. Проверка достижимости h2 для h1	21
2.5. Сеть с простой топологией в Mininet	23
2.6. Запуск iPerf-клиента	23
2.7. Вывод статистики iPerf3	23
2.8. Информация о дисциплине очереди на сетевом устройстве s1-eth0	26
3.1. Диаграмма активностей приложения	29
3.2. Диаграмма классов приложения	30
3.3. Топология исследуемой сети	32
3.4. Запуск программы	35
3.5. График изменения количества переданных данных с течением времени	35
3.6. График изменения значения окна перегрузки с течением време- ни при использовании алгоритма TCP Reno	36
3.7. График изменения значения RTT с течением времени	36
3.8. График изменения значения вариации RTT с течением времени	37
3.9. График изменения значения пропускной способности с течени- ем времени	37
3.10. График изменения длины очереди на интерфейсе s2-eth2	38
3.11. График изменения значения повторно переданных пакетов во времени	38
3.12. Топология исследуемой сети	39
3.13. График изменения значения пропускной способности с течени- ем времени	44
3.14. График изменения значения окна перегрузки с течением време- ни при использовании алгоритма TCP Reno	44
3.15. График изменения значения RTT с течением времени	45

Аннотация

Объектом исследования данной выпускной квалификационной работы является способ моделирования модуля активного управления трафиком сети передачи данных. На сегодняшний день существует множество различных методов моделирования, применяемых для исследований сетей передачи данных. Среда виртуального моделирования Mininet, используемая в работе, позволяет использовать реальные сетевые приложения, сетевые протоколы и ядро Unix/Linux для тестирования и анализа характеристик моделируемых в ней компьютерных сетей и сетевых протоколов. Использование Mininet позволяет производить моделирование сети с минимальными временными затратами и минимальными финансовыми издержками.

В процессе написания работы были рассмотрены способы построения сети передачи данных, исследованы сетевые характеристики, такие как пропускная способность сети, длина очереди пакетов на сетевом интерфейсе устройства, размер окна TCP на компьютере отправителя, круговая задержка. Создан программный комплекс на языке программирования Python, который позволяет создавать сеть и рассматривать ее сетевые характеристики, не прибегая к изменению программного кода. В качестве примера работы с программой были рассмотрены способы создания и исследования сетей имеющие различные сетевые параметры и топологии.

Список используемых сокращений

Англоязычные сокращения

API — Application Programming Interface — программный интерфейс приложения

CBF — Credit Base Fair Queue

CLI — Command line interface — интерфейс командной строки

CWND — congestion window — окно перегрузки

HDN — Hardware Defined Network — аппаратно определяемая сеть

HTB — Hierarchy Token Bucket

IP — Internet Protocol — межсетевой протокол

LLDP — Link Layer Discovery Protocol

ONF — Open Networking Foundation

QoS — quality of service — качество обслуживания

RED — Random Early Detection

RTT — Round Trip Time

SCTP — Stream Control Transmission Protocol — протокол передачи с управлением потоком

SDN — Software-defined Networking

SFQ — Stochastic Fairness Queueing

TBF — Token bucket filter

TCP — Transmission Control Protocol — протокол управления передачей

UDP — User Datagram Protocol — протокол пользовательских датаграмм

Введение

Объектом исследования данной выпускной квалификационной работы является способ моделирования модуля активного управления трафиком сети передачи данных, а также иллюстрация применения данной модели для исследований работы реальных сетей и сетевых приложений.

Актуальность

Актуальность темы обусловлена потребностью организаций в грамотном проектировании и развертывании локальной сети предприятия. Создание сети не может обходиться без предварительного анализа и прототипирования. Для данных задач могут использоваться современные программы, которые позволяют создавать и испытывать сети без реальных сетевых компонентов. Такое решение дешево в построении, а сбор данных сетевых характеристик заметно ускоряется и упрощается.

Так же, исследователям в области сетевых технологий требуется проверять качество работы сетевых протоколов или приложений в определенных условиях. В данной ситуации применение программного комплекса, который поможет автоматически получить приближенные данные эффективности работы сетевого компонента, поможет сократить временные затраты на развертывание и исследование поведения сети.

Цель работы

Целью работы является создание программного компонента, который позволяет моделировать и измерять сетевые характеристики передачи данных без использования реальных сетевых компонентов.

Задачи

1. Построить модуль активного управления трафиком в Mininet.
2. Измерить и визуализировать характеристики моделируемой сети передачи данных для качественной оценки производительности сети.
3. Оценить влияние различных комбинаций протоколов и сетевых топологий на общую производительность моделируемой сети.
4. Исследовать результат моделирования.

Методы исследования

Методами исследования предметной области являются эксперимент, наблюдение, сравнение и измерение. Каждый из этих методов полезен на практике при построении сетей.

Апробация

Результаты, полученные в ходе выполнения работы, были представлены на конференции “Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологических систем” (ИТТММ 2022) (Москва, РУДН, 18–22 Апреля 2022 г.).

Публикации

По теме выпускной квалификационной работы были опубликованы в [34].

Структура работы

Работа состоит из введения, трех глав, заключения и списка используемой литературы из 34 наименований и 4 приложений.

Во введении сформулированы цели и задачи, описана актуальность работы и методы, используемые в работе.

В первой главе приводятся основные сведения о методах моделирования модуля активного управления трафиков сети передачи данных и литературный обзор по заданной тематике.

Во второй главе описаны программные средства, с помощью которых создается программный комплекс для моделирования модуля активного управления трафиком и исследования сетевых характеристик элементов сети.

Третья глава посвящена способу взаимодействия с программным модулем и исследованию сетевых характеристик сетей, имеющих различные топологии и сетевые параметры. В данной главе реализован программный модуль для среды Mininet, который позволяет создавать виртуальные сети без правки исходного кода программы.

В Заключение представлены результаты и выводы по проделанной работе.

В Приложении представлены полные листинги программ, написанных в ходе подготовки Выпускной квалификационной работы.

1. Моделирование сетей в среде Mininet и анализ их производительности

1.1. Обзор исследований в области моделирования в среде Mininet

Качество работы сетевого приложения определяется, в значительной степени, качествами линии передачи данных, сетевых устройств и работающих в них алгоритмах обработки потока данных. Современные сети передачи данных не могут обеспечить нужный уровень качества обслуживания по ряду причин. Одна из них – слабая взаимосвязь физических сетевых элементов разных уровней. Контроль передачи данных лежит на каждом устройстве отдельно, что, в некоторых случаях, ведет к многочисленным потерям данных и задержкам. Консорциум Open Networking Foundation (ONF) [18] предложил решение данной проблемы: отделить уровень контроля передачи данных от сетевого устройства и перенести все эти заботы на некоторый контроллер. Управление данным контроллером ведется централизованно с помощью протокола OpenFlow [19], что помогает установить нужный уровень предоставления услуг для конкретного приложения. Такое устройство сети консорциум ONF назвал Software-defined Networking (SDN) [26]. Для изучения производительности сетей SDN методом моделирования организация ONF разработала эмулятор сети Mininet.

В работе [22] речь идет об имитационном моделировании сети SDN с помощью средств Mininet и измерении производительности сети. Были затронуты следующие показатели производительности: RTT [23] для каждого направления связи, пропускная способность ветвей и направлений связи, величина задержки на сетевых элементах, загрузка портов OpenFlow Switch, элементы сети с наибольшей задержкой, число обслуженных и потерянных пакетов.

Mininet хорошо подходит для задач имитационного моделирования и исследований общей эффективности работы сети, однако, для создания сети требуется умение писать программы и знание Mininet API.

Работа [13] представляет собой подробное руководство по взаимодействию с Mininet, как с помощью CLI [3], так и с помощью написания программ на языке программирования Python. Авторы показывают простоту создания сетевых топологий различной сложности, такие как Minimal, Single, Linear, Tree, Reversed. Также, в работе представлена программа, которая создает собственную сеть внутри Mininet и проводит простой анализ ее производительности. Созданная сеть детерминирована и состоит всего из 3 узлов, соединенных коммутатором. Анализ работы сети не имеет смысла, так как сеть создана, скорее, для демонстрации принципов работы с Mininet, а не для исследований. Из данной работы видно, что Mininet отлично подходит для исследований производительности сети и сетевых компонентов, благодаря своей простоте и настройке.

Идея автоматизации процесса создания топологий сети возникает сама собой, после написания двух-трех программ. Авторы работы [7] описывают способ такой автоматизации. Способ основан на создании конфигурационного файла, в котором описаны основные правила адресации, создания сетевых устройств и соединений между ними. Чтение конфигурационного файла происходит в программе, написанной на языке программирования Python. Данный способ заметно ускоряет развертывание сети, избавляя исследователя от постоянных модификаций программного кода.

1.2. Обзор исследований в области анализа производительности сетей и сетевых компонентов в среде Mininet

Простое моделирование сети не дает информации об ее производительности. Нужно провести качественный анализ работы сетевых компонентов, рассчитать пропускную способность соединения, количество потерянных и доставленных пакетов и т. д. Работа [24] является хорошим началом для изучения методов анализа производительности сети. Авторы исследуют задержки в сети при передаче данных и сравнивают показатели в HDN и SDN. Hardware Defined Network (HDN) — привычные нам сети, где управление передачи данных лежит на устройствах сетевого и канального уровней. Работа показала, что SDN справляется с работой лучше и средняя задержка передачи данных в ней ниже (3.891 мс против 8.277 мс). Однако, стоит заметить, что здесь речь идет о передаче простого ICMP пакета, а не потока TCP/UDP трафика.

Работа [20] является более интересным примером анализа производительности сети в Mininet. В ней рассматриваются вопросы оценки производительности механизмов для эффективной работы с перегрузки в SDN. Авторы оценивают общую производительность сети, сравнивая ее с производительностью сети, которая использует Link Layer Discovery Protocol (LLDP) [14]. Данный протокол канального уровня позволяет сетевому оборудованию оповещать оборудование, работающее в локальной сети, о своём существовании и передавать ему свои характеристики. Протокол отлично подходит для новой концепции построения сетей SDN, которую мы рассматривали ранее, так как позволяет SDN-контроллеру знать характеристики элементов сети и в зависимости от этого управлять потоками трафика. Оценка производительности основывается на трех пунктах:

- Уровень потери пакетов;
- Уровень доставки пакетов;

— Общая пропускная способность.

Данные оценки определяются для каждой итерации исследования. Всего таких итераций 4, и они имеют следующие сетевые топологии:

1. 1 SDN-контроллер, 4 коммутатора, 1 хост, 4 соединения;
2. 1 SDN-контроллер, 15 коммутаторов, 16 хостов, 30 соединений;
3. 1 SDN-контроллер, 40 коммутаторов, 81 хост, 120 соединений;
4. 1 SDN-контроллер, 85 коммутаторов, 256 хостов, 340 соединений.

В ходе работы были построен график, которые отображают зависимость сетевой характеристики от сетевой топологии для обычной сети и сети, которая использует LLDP. На графиках видно, что сеть второго типа показывает лучший уровень производительности для каждой сетевой характеристики передачи данных: ниже уровень потерь пакетов, выше уровень доставки пакетов, выше общая пропускная способность.

Mininet отлично подходит для проведения исследований поведенческих особенностей сетевых компонентов. Однако, можно также провести исследование работоспособности и производительности сетевых протоколов и приложений. Авторы статьи [1] исследуют производительность алгоритма для эффективной работы с перегрузками BBRv2. Анализ работы алгоритма, презентацию и исходный код можно найти в [32]. BBRv1 – нестандартный алгоритм управления перегрузками разработанный в Google, который не использует потерю пакетов как маркер для снижения скорости отправки. Алгоритм BBRv1 [2], в сравнении с предшествующими алгоритмами, выдает большую пропускную способность для потоков данных при равных условиях. Одним из минусов алгоритма является его слабая совместимость с более старыми алгоритмами, используемыми в сети, и алгоритм BBRv2 создан как раз для исправления данной брешы. В исследовании средствами Mininet создается сеть, имеющая 100 узлов отправителей и 100 узлов получателей. Отправители и получатели связаны между собой сетью из 3-х коммутаторов, каждый из которых имеет свою задачу: - коммутатор 1: является точкой входа для узлов-отправителей,

эмулирует потери и задержки данных с помощью средств NetEm [29], соединен с коммутатором 2; - коммутатор 2: эмулирует «узкое горлышко» между отправителями и получателями, ограничивая скорость передачи данных до 1 Гбит/с. Ограничивание передачи осуществляется с помощью дисциплины очередей TBF [30]. - коммутатор 3: соединяет коммутатор 2 и хосты получатели.

В ходе работы были исследованы и сравнены алгоритмы CUBIC [4], BBRv1, BBRv2 на такие сетевые характеристики: пропускная способность, индекс справедливости [5], сосуществование. Результаты показывают, что BBRv2 обеспечивает лучшее сосуществование с потоками, использующие алгоритм CUBIC, по сравнению со своим предшественником. Кроме того, BBRv2 способен обеспечить более справедливую долю пропускной способности по сравнению с BBRv1, когда сетевые условия, такие как пропускная способность и задержка, динамически изменяются.

1.3. Алгоритмы управления перегрузками в сети

Часто в теории сетей встречается термин перегрузка. Данное понятие характеризуется уменьшением пропускной способности сети вследствие возникновения заторов на интерфейсах коммутаторов. Пакеты, поступая на интерфейс, образуют очередь, если пакет не может быть передан дальше по линии передачи данных. Это происходит из-за несоответствий входной и выходной пропускной способности линии передачи данных. Данная очередь хранится в буфере коммутатора и непосредственно влияет на передачу данных. Например, если памяти много и очередь длинная, то пакет вынужден ждать значительное время, прежде чем он будет передан. И наоборот, если очередь мала, то пакет быстрее пойдет дальше по сети. Однако, в таком случае, пакеты, не помещающиеся в очередь, будут попросту отброшены и не дойдут до получателя.

Исследователи в области сетевых технологий постоянно ищут баланс между

временем доставки пакета и пропускной способностью сети, в которой сосуществует много пользователей, единственным желанием которых является отправка наибольшего количества данных за наименьшее количество времени. В данный момент существует множество алгоритмов, удовлетворяющих, хотя бы отчасти, эти запросы.

Работа [28] описывает механизмы, используемые в алгоритмах работы с перегрузками. Данные механизмы именуются как медленный старт, быстрая повторная передача, быстрое восстановление. Данные механизмы завязаны на увеличении/уменьшении окна перегрузки (некоторый набор сегментов данных tcp) в зависимости от того, был ли переданный пакет из сегмента потерян. Определяется потеря пакета как сама потеря или же как получение трех дубликатов получения пакета.

Медленный старт – фаза наращивания передачи данных, при которой окно перегрузки увеличивается экспоненциально каждый раз, когда получено подтверждение. Это происходит до тех пор, пока для какого-то сегмента не будет получено подтверждение или будет достигнуто какое-то заданное пороговое значение таймера. Как только произойдет событие потери пакета, задается пороговое значение медленного старта, равное половине окна перегрузки.

Быстрое восстановление - вариация алгоритма медленного старта, которая использует быструю повторную передачу последующей фазой предотвращения перегрузки. В алгоритме быстрого восстановления, во избежание перегрузки, когда пакет не получен, размер окна перегрузки сводится к порогу медленного старта, а не к меньшему начальному значению.

Механизмы медленного старта и быстрого восстановления, включающий в себя механизм быстрой повторной передачи, используются, например, в алгоритме TCP Reno. Помимо прочего, в TCP Reno имеется механизм аддитивного увеличения и мультипликативного уменьшения, который включается после потери пакета в фазе медленного старта.

Однако, не все алгоритмы работы с перегрузками используют как маркер

перегрузки в сети потерю пакета. При взаимодействии потоков данных часто полезно использовать маркер перегрузки значение пропускной способности и RTT, не доходя до появления потерь. Данный метод подробно описан в работе [2] и реализован в алгоритме TCP BBR и расширен в TCP BBRv2 [32]. Данный алгоритм позволяет держать пропускную способность в оптимальном значении, постоянно исследуя изменения RTT (изменение возможно только в случае физического изменения длины передачи данных) и текущей пропускной способности. Начинается работа алгоритма с фазы медленного старта, находится оптимальное значение RTT и BW (пропускная способность) и поочередно на каждом шаге происходит проверка на улучшение сетевых показателей. Например, стало ли нам доступно большая полоса пропускания данных, можем ли мы получить меньший RTT и т. д. Данный алгоритм требует больших вычислительных ресурсов в ряду своей сложности, однако, дает приближенные к оптимальным значения характеристик BW и RTT при передаче данных. Следует так же заметить, что TCP BBR вытесняет потоки данных, использующие другие версии алгоритмов управления перегрузками, которые используют потерю пакетов как метрику перегрузки. Данная проблема была решена в алгоритме TCP BBRv2.

2. Исследование возможностей моделирования сетей и измерения сетевых характеристик в Mininet

2.1. Mininet

2.1.1. Общие сведения

Mininet [15] — это виртуальная среда, которая позволяет разрабатывать и тестировать сетевые инструменты и протоколы. В сетях Mininet работают реальные сетевые приложения Unix/Linux, а также реальное ядро Linux и сетевой стек. С помощью одной команды Mininet может создать виртуальную сеть на любом типе машины, будь то виртуальная машина, размещенная в облаке или же собственный персональный компьютер. Это дает значительные плюсы при тестировании работоспособности протоколов или сетевых программ:

- позволяет быстро создавать прототипы программно-определяемых сетей;
- тестирование не требует экспериментов в реальной сетевой среде, вследствие чего разработка ведется быстрее;
- тестирование в сложных сетевых топологиях обходится без необходимости покупать дорогое оборудование;
- виртуальный эксперимент приближен к реальному, так как Mininet запускает код на реальном ядре Linux;
- позволяет работать нескольким разработчикам в одной топологией независимо.

Машины (хосты) в сети создаются по образу машины, которая запускает Mininet, со всеми вытекающими обстоятельствами. Например, если количество памяти, допустимое для буфера передачи сокета TCP, на рабочей машине

равно, например, значению 4096MB, то и на виртуальной машине это значение будет таким же. Изменение конфигурации на машине в виртуальной сети не вносит изменений в конфигурацию рабочей машины.

2.1.2. Создание простой сети с помощью Python и модуля Mininet

Mininet предоставляет гибкий API [16] в виде модуля для программ на языке программирования Python. С его помощью можно строить сети различных топологий и производить с ними требуемые манипуляции. Для того, чтобы получить доступ к API, требуется с помощью pip [21] установить mininet и подключить данный модуль к исполняемому файлу. Приведем пример построения сети простой топологии. Пусть, у нас имеется 10 хостов, соединенных между собой с помощью 1 коммутатора. Нам требуется присвоить каждому ip-адрес и проверить достижимость к каждому узлу.

Код такой программы на языке программирования Python представлен ниже:

```
1 from mininet.link import TCLink
2 from mininet.net import Mininet
3 from mininet.node import CPULimitedHost
4 from mininet.topo import Topo
5
6 '''
7 Создание простой топологии:
8 10 хостов, 1 коммутатор, 10 соединений
9 '''
10 class CustomTopology(Topo):
11
12     def __init__(self, **opts):
13         super(CustomTopology, self).__init__(**opts)
14         s1 = self.addSwitch(name="s1")
15         for i in range(1, 11):
16             host = self.addHost(name="h%d" % i, ip="10.0.0.%d" % i)
17             self.addLink(host, s1)
18
19
```

```

20 '''
21 Запуск сети и проверка достижимости элементов
22 '''
23 if __name__ == "__main__":
24     topology = CustomTopology()
25     net = Mininet(topo=topology, host=CPULimitedHost, link=TCLink)
26     net.start()
27     print("Сеть заработала")
28     net.pingAll()
29     net.stop()
30     print("Сеть остановилась")

```

Для того, чтобы Mininet мог знать, какую сеть строить, создается класс топологии CustomTopology, являющийся наследником класс Торо, предоставляемого модулем Mininet. Далее все просто: к топологии добавляется коммутатор, через цикл добавляются хосты и соединяются с коммутатором. В основной части программы создается объект класса Mininet, в который мы через параметр конструктора добавляем нашу топологию, запускаем сеть, с помощью метода Mininet **pingAll** проверяем доступность узлов сети и, в конце концов, останавливаем сеть.

Запустим программу с помощью команды

```
sudo python3.8 main.py
```

На рис.2.1 видно, что все узлы являются достижимыми.

2.1.3. MiniEdit

Mininet предоставляет графический интерфейс управления виртуальной сетью — MiniEdit. MiniEdit — программа, написанная на языке программирования Python, которая является надстройкой над mn и позволяет управлять сетью в удобном для пользователя виде. Данная программа расположена в директории examples исходных файлов mininet. В моем случае это директория /usr/local/lib/python3.8/dist-packages/mininet/examples/miniedit.py.

Создадим простую топологию из двух хостов и коммутатора в MiniEdit.

```

sergey@sergey:~/PycharmProjects/mininet_test$ sudo python3.8 main.py
Сеть заработала
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Results: 0% dropped (90/90 received)
Сеть остановилась

```

Рис. 2.1. Запуск простой сети и проверка достижимости узлов

1. Запустим Miniedit из директории /usr/local/lib/python3.8/dist-packages/mininet/examples/miniedit.py

`sudo python3`

↪ `/usr/local/lib/python3.8/dist-packages/mininet/examples/miniedit.py`

2. Создадим 2 хоста, выбрав иконку терминала и кликнув по рабочей области 2 раза. Имена для хостов присваиваются автоматически.
3. Кликнув правой кнопкой мыши по хосту и выбрав раздел Properties, зададим в поле IP Address адреса 10.0.0.1 и 10.0.0.2 соответственно (рис. 2.2).

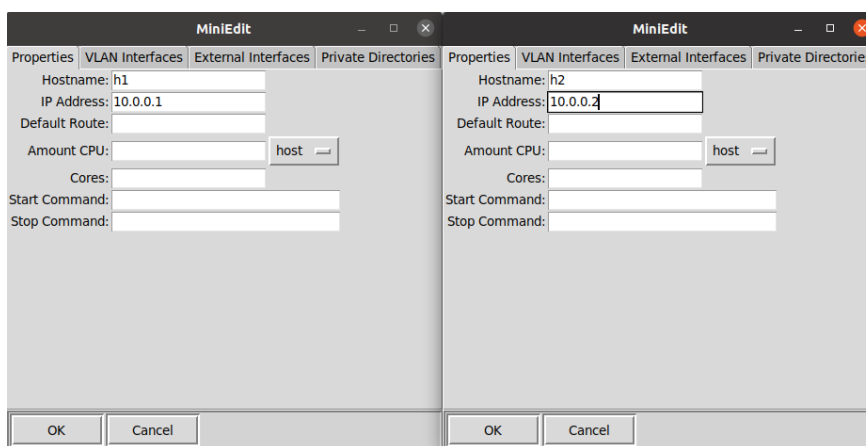


Рис. 2.2. Редактирование конфигурации хостов сети

4. Добавим в рабочую область коммутатор, выбрав элемент LegacySwitch.



Рис. 2.3. Соединение элементов сети

5. Выберем элемент NetLink и соединим элементы сети (рис. 2.3).
6. Запустим сеть нажав на кнопку Run.
7. Откроем терминал первого хоста, нажав правой кнопкой мыши по хосту и выбрав пункт Terminal. Отправим ping второму хосту (рис. 2.4).

```
Host: h1
root@sergey:/home/sergey# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.446 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.063 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.063/0.160/0.446/0.164 ms
root@sergey:/home/sergey#
```

Рис. 2.4. Проверка достижимости h2 для h1

2.2. Генерация и измерение сетевого трафика с помощью утилиты iPerf3

2.2.1. Общие сведения

iPerf3 [10] — кроссплатформенная консольная клиент-серверная программа-генератор TCP, UDP и SCTP трафика для тестирования пропускной способности

сети. По умолчанию тест выполняется в направлении от клиента к серверу. Для выполнения тестирования программа должна быть запущена на двух устройствах (это могут быть как компьютеры, так и смартфоны, планшеты). Одно из них будет выполнять роль сервера, а другое роль клиента. Между ними и будет происходить передача данных для измерения пропускной способности соединения.

2.2.2. Тестирование пропускной способности с помощью iPerf3

Для запуска сервера iPerf требуется выполнить следующую команду

```
iperf3 -s [параметры]
```

Для запуска iPerf-клиента требуется выполнить следующую команду

```
iperf3 -c server_ip [параметры]
```

Подробнее со списком параметров iPerf3 можно ознакомиться в [11]

Проведем тестирование сети в Mininet.

1. Запустим MiniEdit.
2. Создадим 2 хоста (h1 и h2) и коммутатор (s1).
3. Соединим элементы сети.
4. Запустим сеть (рис. 2.5).
5. Запустим iPerf-сервер на h2.

```
iper3 -s
```

6. Запустим iPerf-клиент на хосте h1 на 60 секунд, пропустив первые 10 секунд для статистики (рис. 2.6).

```
iperf3 -c 10.0.0.2 -t 60 -O 10
```

7. Просмотрим статистику, которую сгенерировал iPerf (рис. 2.7).

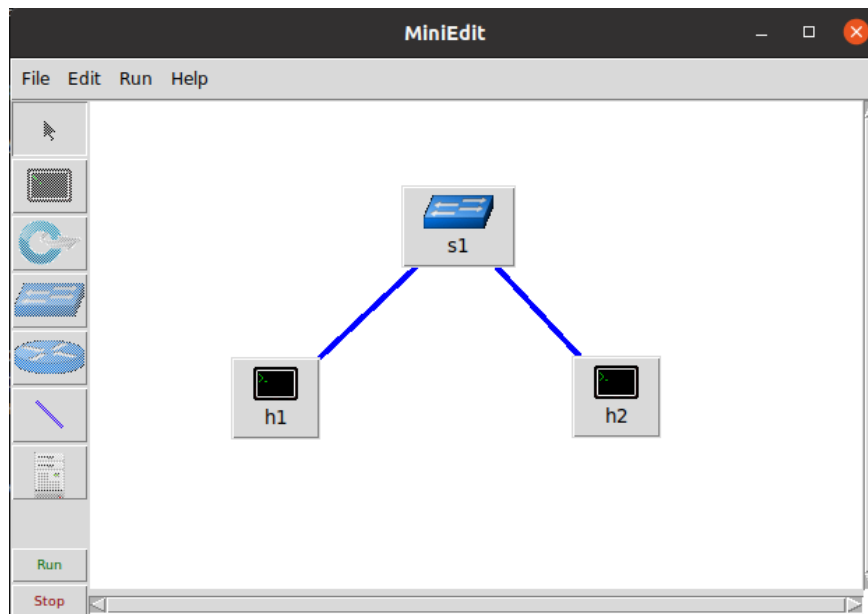


Рис. 2.5. Сеть с простой топологией в Mininet

"Host: h1"									
Connecting to host 10.0.0.2, port 5201									
[7] local 10.0.0.1 port 47538 connected to 10.0.0.2 port 5201									
[ID]	Interval		Transfer	Bitrate	Retr	Cwnd			
[7]	0.00-1.00	sec	5.60 GBytes	48.1 Gbits/sec	0	310 KBytes	(omitted)		
[7]	1.00-2.00	sec	5.74 GBytes	49.3 Gbits/sec	0	386 KBytes	(omitted)		
[7]	2.00-3.00	sec	5.75 GBytes	49.4 Gbits/sec	0	386 KBytes	(omitted)		
[7]	3.00-4.00	sec	5.73 GBytes	49.2 Gbits/sec	0	386 KBytes	(omitted)		
[7]	4.00-5.00	sec	5.65 GBytes	48.6 Gbits/sec	0	525 KBytes	(omitted)		
[7]	5.00-6.00	sec	5.44 GBytes	46.8 Gbits/sec	0	525 KBytes	(omitted)		
[7]	6.00-7.00	sec	5.36 GBytes	46.1 Gbits/sec	0	583 KBytes	(omitted)		
[7]	7.00-8.00	sec	5.74 GBytes	49.3 Gbits/sec	0	583 KBytes	(omitted)		
[7]	8.00-9.00	sec	5.65 GBytes	48.5 Gbits/sec	0	583 KBytes	(omitted)		
[7]	9.00-10.00	sec	5.47 GBytes	47.0 Gbits/sec	0	583 KBytes	(omitted)		
[7]	0.00-1.00	sec	5.47 GBytes	47.0 Gbits/sec	0	1.01 MBytes			
[7]	1.00-2.00	sec	5.70 GBytes	48.9 Gbits/sec	0	1.01 MBytes			
[7]	2.00-3.00	sec	5.70 GBytes	49.0 Gbits/sec	0	1.01 MBytes			

Рис. 2.6. Запуск iPerf-клиента

"Host: h1"									
[7]	51.00-52.00	sec	5.67 GBytes	48.7 Gbits/sec	0	8.45 MBytes			
[7]	52.00-53.00	sec	5.62 GBytes	48.2 Gbits/sec	0	8.45 MBytes			
[7]	53.00-54.00	sec	5.60 GBytes	48.1 Gbits/sec	0	8.45 MBytes			
[7]	54.00-55.00	sec	5.66 GBytes	48.7 Gbits/sec	0	8.45 MBytes			
[7]	55.00-56.00	sec	5.74 GBytes	49.3 Gbits/sec	0	8.45 MBytes			
[7]	56.00-57.00	sec	5.74 GBytes	49.3 Gbits/sec	0	8.45 MBytes			
[7]	57.00-58.00	sec	5.70 GBytes	49.0 Gbits/sec	0	8.45 MBytes			
[7]	58.00-59.00	sec	5.77 GBytes	49.5 Gbits/sec	0	8.45 MBytes			
[7]	59.00-60.00	sec	5.47 GBytes	47.0 Gbits/sec	0	8.45 MBytes			
[ID]	Interval		Transfer	Bitrate	Retr				
[7]	0.00-60.00	sec	335 GBytes	48.0 Gbits/sec	0	sender			
[7]	0.00-60.00	sec	335 GBytes	48.0 Gbits/sec	0	receiver			

iperf Done.
root@sergey:/home/sergey#

Рис. 2.7. Вывод статистики iPerf3

2.3. Использование утилиты `iproute2` для настройки интерфейсов сетевых элементов

2.3.1. Общие сведения

`iproute2` [12] — это набор утилит для управления параметрами сетевых устройств в ядре Linux. Эти утилиты были разработаны в качестве унифицированного интерфейса к ядру Linux, которое непосредственно управляет сетевым трафиком.

`iproute2` заменил полный набор классических сетевых утилит UNIX, которые ранее использовались для настройки сетевых интерфейсов, таблиц маршрутизации и управления arp-таблицами: `ifconfig`, `route`, `arp`, `netstat` и других, предназначенных для создания IP-туннелей. `iproute2` предлагает унифицированный синтаксис для управления самыми разными аспектами сетевых интерфейсов.

Набор утилит включает в себя три основные программы:

- `ip` [9] — утилита для просмотра параметров и конфигурирования сетевых интерфейсов, сетевых адресов, таблиц маршрутизации, правил маршрутизации, arp-таблиц, IP-туннелей, адресов multicast рассылки, маршрутизацией multicast пакетов.
- `tc` [31] — утилита для просмотра и конфигурирования параметров управления трафиком. Позволяет управлять классификацией трафика, дисциплинами управления очередями для различных классов трафика либо целиком для сетевого интерфейса, что, в свою очередь, позволяет реализовать QoS в нужном для системы объёме:
 - разделение разных типов трафика по классам;
 - назначение разных дисциплин обработки очередей трафика с разным приоритетом, механизмами прохождения очереди, ограничениями по скорости и т. п.
- `ss` [27] — утилита для просмотра текущих соединений и открытых портов.

Аналог утилиты netstat.

2.3.2. Утилита tc

Утилита tc [31] наиболее полезна для исследования, так как она позволяет гибко настроить поведение контроля исходящего трафика. Система контроля трафика состоит из:

- **Ограничения исходящего трафика.** Когда трафик сформирован, его полоса пропускания начинает контролироваться. Ограничение может дать больше, чем уменьшение полосы пропускания — оно также используется для сглаживания пиков для более прогнозируемого поведения сети.
- **Планирования передачи пакетов.** Механизм позволяет увеличить интерактивность исходящего трафика при гарантировании полосы пропускания для передачи данных большого объема. Такое упорядочение также называется приоритезацией и применяется для исходящего трафика.
- **Ограничения исходящего трафика.** Этот механизм позволяет ограничить количество пакетов или байт в потоке входящего трафика, соответствующих определенной классификации.
- **Отбрасывания.** Трафик, превышающий установленную полосу пропускания, может быть отброшен как для входящего, так и исходящего трафика. Обработка трафика контролируется тремя типами объектов: очередями, классами и фильтрами. Обработка трафика контролируется тремя типами объектов: очередями, классами и фильтрами.

Для информации о tc используйте команду

```
tc help
```

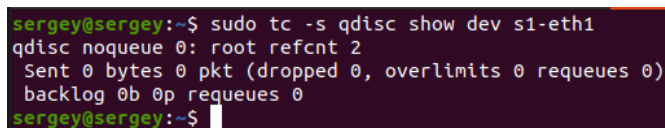
Дисциплина очереди — это алгоритм обработки очереди сетевых пакетов. Дисциплин на одном интерфейсе может быть задействовано несколько, а непосредственно к интерфейсу крепится корневая дисциплина (root qdisc).

При этом каждый интерфейс имеет свою собственную корневую дисциплину. Каждой дисциплине и каждому классу назначается уникальный дескриптор, который может использоваться последующими инструкциями для ссылки на эти дисциплины и классы. Помимо исходящей дисциплины, интерфейс так же может иметь и входящую дисциплину, которая производит управление входящим трафиком. Дисциплины на интерфейсе образуют иерархию, где в верху иерархии находится корневая дисциплина. Сам интерфейс ничего не знает о дисциплинах, находящихся под корневой, а поэтому работает только с ней. Дисциплины делятся на классовые (CBF, HTB, PRIО) и бесклассовые (pfifo, pfifo_fast, RED, SFQ, TBF). Подробнее о видах дисциплин очередей можно прочесть в [31].

Воспользуемся сетью Mininet из прошлой главы и с помощью `tc` выведем информацию о дисциплине очереди на сетевом устройстве `s1-eth0` (интерфейс коммутатора, к которому подключен хост `h1`). Для этого воспользуемся командой

```
tc -s qdisc show dev s1-eth1
```

На рис. 2.8 видно, что корневой дисциплиной очереди назначена дисциплина `noqueue`. Данная дисциплина означает «отправляй мгновенно, не ставь в очередь».



```
sergey@sergey:~$ sudo tc -s qdisc show dev s1-eth1
qdisc noqueue 0: root refcnt 2
  Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
  backlog 0b 0p requeues 0
sergey@sergey:~$
```

Рис. 2.8. Информация о дисциплине очереди на сетевом устройстве `s1-eth0`

В выводе `tc` можно увидеть полезные для исследования данные:

- *Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)* — означает, что было отправлено 0 байт (0 пакетов), из которых 0 пакетов отброшено и 0 пакетов

вышли за пределы лимита.

— *backlog Ob Op requeues 0* — размер очереди в байтах и пакетах.

Запомним параметр backlog: он будет полезен нам при сборе статистики.

3. Моделирование модуля активного управления трафиком сети передачи данных

3.1. Создание модуля для среды Mininet

В главе 2 уже был рассмотрен способ моделирования сети в среде Mininet. Данный способ включает в себя создание программы на языке программирования Python. В программе описывается топология сети, манипуляции с сетью, ее старт и завершение. Написав пару таких сетей, можно столкнуться с мыслью, что части кода идентичны друг другу, а каждое редактирование и исследование сети занимает время. Можно написать автоматизированное решение, которое требует только задание нужных сетевых элементов и их конфигураций, а программа сама считывает данные, создаст сеть, построит графики сетевых характеристик и т. д. Данная идея будет описана далее в этой главе.

Для начала требуется выбрать формат конфигурационного файла. Выбор формата конфигурационного файла производится из собственных предпочтений и удобства анализа файла. В данной работе будет использован формат `toml` [33]. Определим объекты сети, которые мы будем описывать в конфигурационном файле: хосты, коммутаторы, соединения, мониторинговые характеристики, команды `qdisc` для интерфейсов коммутатора.

Имея подобный конфигурационный `toml`-файл, его можно прочесть с помощью средств Python, обработать и положить требуемые значения в объекты. Такой подход позволяет строить сколь угодно большие топологии без правки логики приложения. На основе данного файла можно построить программу, использующую диаграмму активностей, показанную на рис.3.1

Программу можно разделить на такие модули как: мониторинг сети, построение графиков сетевых характеристик, модуль модели сети, который включает в себя топологию сети и два предыдущих элемента. Исходя из этого можно

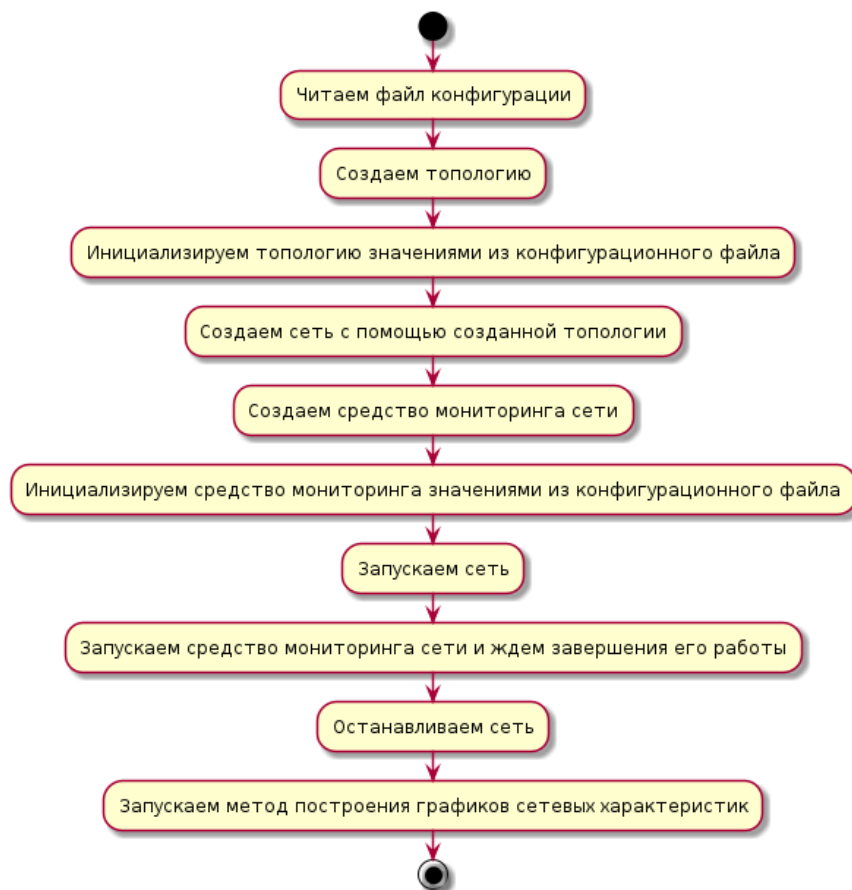


Рис. 3.1. Диаграмма активностей приложения

построить диаграмму классов приложения, показанную на рис.3.2.

Главным классом, включающим в себя все остальные, является CustomModel. В методе simulation создаются объекты классов Monitor, mininet.net.Mininet, CustomTopology и NetStatsPlotter. Рассмотрим подробнее классы, создаваемые в методе simulation:

- mininet.net.Mininet — предоставляемый Mininet API класс, отвечающий за создание сети с топологией, указанной в CustomTopology;
- CustomTopology — класс топологии сети;
- Monitor — класс, в котором происходят замеры сетевых характеристик исследуемой сети;
- NetStatsPlotter — класс, объект которого занимается построением графиков сетевых характеристик.

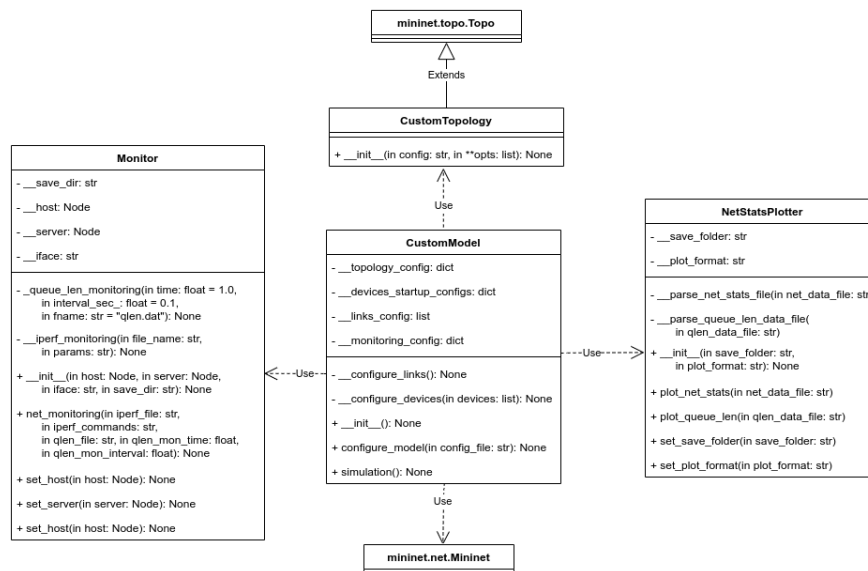


Рис. 3.2. Диаграмма классов приложения

Код классов NetStatsPlotter, Monitor, CustomTopology и CustomModel находятся в приложении А, приложении В, приложении С и приложении D соответственно.

Точкой входа в данную программу будет файл main.py. Содержимое данного файла представлено ниже:

```

1  '''
2  #!/usr/bin/python3.8
3
4  import argparse
5  from model.CustomModel import CustomModel
6
7  if __name__ == '__main__':
8      parser = argparse.ArgumentParser()
9      h = "Файл конфигурации"
10     parser.add_argument('-c', '--config', type=str, help=h)
11     args = parser.parse_args()
12     if args.config:
13         top = CustomModel()
14         top.configure_model(args.config)
15         top.simulation()
16     else:
17         print("Введите название конфиг-файла")
  
```

Это обычный скрипт на языке программирования Python, который создает модель нашей сети из конфигурационного файла, переданного параметром командной строки. Запустить данный скрипт можно с помощью команды

```
sudo ./main.py -c config/pfifo_config.toml
```

Параметр `-c` отвечает за местоположение конфигурационного файла. Естественно, перед запуском скрипта требуется задать право на исполнение файла. Делается это с помощью команды

```
chmod +x main.py
```

После запуска программы в каталоге приложения появится директория с именем, которое было указано в `toml`-файле. В ней содержатся графики изменения сетевых характеристик и сырые данные, которые были обработаны объектом класса `NetStatsPlotter`.

3.2. Тестирование программного модуля

У нас имеется готовая программа в наличии и нам следует ее протестировать. Для начала требуется описать сеть, заполнить конфигурационный файл по характеристикам заданной сети, запустить программу и исследовать сетевые характеристики сети передачи данных.

Пусть у нас имеется сеть, заданная топологией, приведенной на рис.3.3.

Предполагается, что в данной сети будут действовать следующие правила:

- скорость передачи данных ограничена;
- максимальная пропускная способность соединения `h1-s1` равна 100 Мбит/с;
- максимальная пропускная способность соединения `s2-h2` равна 50 Мбит/с;
- на коммутаторе `s2` стоит дисциплина обработки очередей FIFO с максимальным количеством пакетов, равным 60;



Рис. 3.3. Топология исследуемой сети

- потери в сети составляют 0.001%;
 - задержка имеет нормальное распределение с математическим ожиданием в 30 мс и с дисперсией в 7 мс.
 - в сети работает алгоритм для работы с перегрузками TCP Reno.
- Опишем данные характеристики в конфигурационном файле.

```
1  '''
2  # device settings
3  [devices]
4      [devices.h1]
5          name = "h1"
6          ip = "10.0.0.1"
7          cmd = [
8              "sysctl -w net.ipv4.tcp_congestion_control=reno"
9          ]
10     [devices.h2]
11         name = "h2"
12         ip = "10.0.0.2"
13         cmd = [
14             "sysctl -w net.ipv4.tcp_congestion_control=reno"
15         ]
16
17  # switch settings
18  [switches]
19      [switches.s1]
20          name = "s1"
21      [switches.s2]
```



```

22         name = "s2"
23
24 # link settings
25 [links]
26 pairs = [
27     ["h1", "s1"],
28     ["s1", "s2"],
29     ["s2", "h2"]
30 ]
31 cmd = [
32     "tc qdisc replace dev s1-eth2 root handle 10: tbf rate 100mbit burst 50000 limit
↵ 150000",
33     "tc qdisc add dev s1-eth2 parent 10: handle 20: netem loss 0.001% delay 30ms 7ms
↵ distribution normal",
34     "tc qdisc replace dev s1-eth1 root handle 10: tbf rate 100mbit burst 50000 limit
↵ 150000",
35     "tc qdisc add dev s1-eth1 parent 10: handle 20: netem loss 0.001% delay 30ms 7ms
↵ distribution normal",
36     "tc qdisc replace dev s2-eth2 root handle 10: tbf rate 50mbit burst 25000 limit
↵ 75000",
37     "tc qdisc add dev s2-eth2 parent 10: handle 15: pfifo limit 30",
38     "tc qdisc replace dev s2-eth1 root handle 10: tbf rate 50mbit burst 25000 limit 75000"
39 ]
40
41 [monitoring]
42 monitoring_time = 60
43 monitoring_interval = 0.005
44 host_client = "h1"
45 host_server = "h2"
46 interface = "s2-eth2"
47 iperf_file_name = "iperf.json"
48 iperf_flags = ""
49 queue_data_file_name = "qlen.data"
50 plots_dir = "plots_dir_first"

```

В представленном toml-файле раздел **devices** отвечает за настройку конечных узлов сети, раздел **switches** отвечает за настройку коммутаторов, а раздел **links** отвечает за настройку соединений узлов сети и конфигурацию интерфейсов коммутаторов.

На хостах указывается ip-адрес, имя хоста и алгоритм работы с перегруз-

ками. На коммутаторах прописывается только имя, однако, список настроек можно расширить, изменив программную логику в классе с топологией. В разделе **links** явно указывается, какие пары сетевых устройств соединяются, и команды, которые настраивают дисциплину очередей на интерфейсах. Важно уточнить, что соединения на коммутаторе происходят последовательно, т. е. если первым идет подключение h1-s1, то интерфейсом на коммутаторе, отвечающим за данное соединение, является интерфейс s1-eth1. Если, например, подключить еще одно устройство к коммутатору, то интерфейсом коммутатора в соединении будет s1-eth2, и так далее. Данный момент следует учитывать при установке правил дисциплин очередей на интерфейсах.

Также, в toml-файле имеется блок **monitoring**, в котором заданы следующие параметры:

- `monitoring_time` — время мониторинга сети в секундах;
- `monitoring_interval` — интервалы между замерами длины очереди в секундах;
- `host_client` — узел, который будет отправлять данные;
- `host_server` — узел, который будет принимать данные;
- `interface` — интерфейс, на котором будет мониториться размер очереди;
- `iperf_file_name` — имя файла с отчетом мониторинга iperf;
- `iperf_flags` — iperf-флаги клиента;
- `queue_data_file_name` — имя файла с отчетом мониторинга длины очереди;
- `plots_dir` — директория со всеми графиками сетевых характеристик.

Все параметры являются обязательными.

Запуск программы показан рис. 3.4.

Рассмотрим получившиеся графики сетевых характеристик:

- На рис. 3.5 приведен график, показывающий динамику объема переданных данных за время моделирования.
- На рис. 3.6 приведен график изменения значений окна перегрузки TCP

```

sergey@sergey: /scl_work/automatic_monitoring$ sudo ./main.py -c config/pfifo_config.toml
Начало работы iperf. Хост: h1, сервер: h2. Файл с данными: plots_dir_first/iperf.json
Начало мониторинга сети на интерфейсе s2-eth2. Продолжительность мониторинга: 60 сек. с интервалом 0.05
Мониторинг окончен. Строим графики.
Графики построены и находятся в директории plots_dir_first.
sergey@sergey: /scl_work/automatic_monitoring$

```

Рис. 3.4. Запуск программы

Reno (CWND).

- Динамика значений RTT (измеряется в миллисекундах) демонстрируется на рис 3.7.
- График отклонений значений RTT приведен на рис. 3.8.
- Динамика изменения пропускной способности показана на рис. 3.9.
- Изменение длины очереди на интерфейсе s2-eth2 показано на рис. 3.10
- Изменение значения количества повторно переданных пакетов показано на рис. 3.11

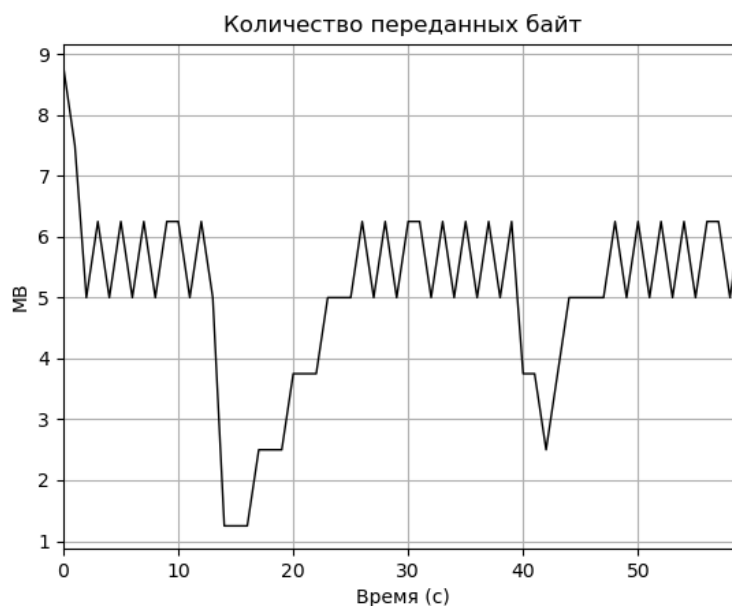


Рис. 3.5. График изменения количества переданных данных с течением времени

Из графиков четко видно, что пропускная способность заметно снижается, когда размер окна уменьшается. График CWND показывает, что таких умень-

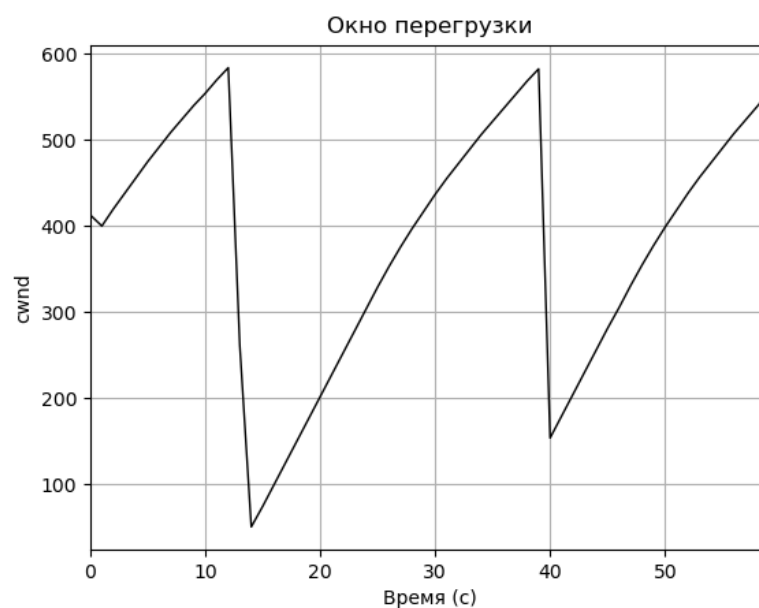


Рис. 3.6. График изменения значения окна перегрузки с течением времени при использовании алгоритма TCP Reno

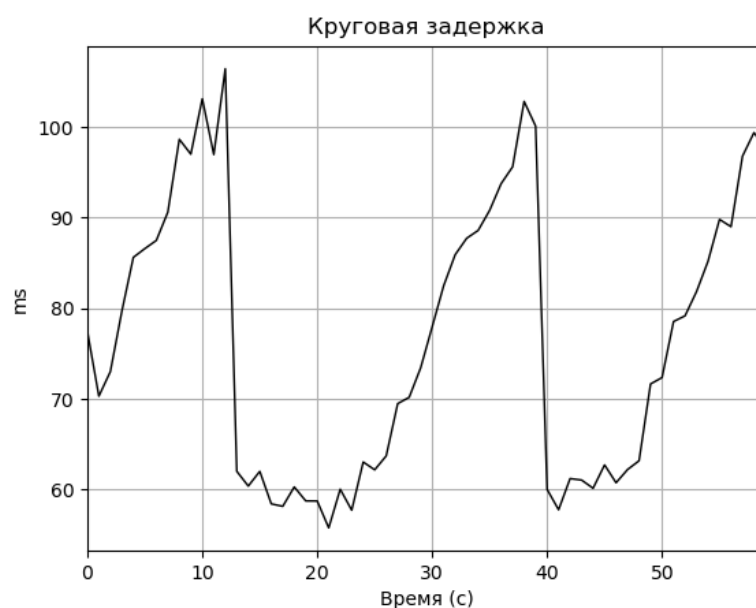


Рис. 3.7. График изменения значения RTT с течением времени

шений размера окна 3 и связаны эти уменьшения, явно, с потерей пакетов в сети. Потери в сети обусловлены общими потерями, связанными с ненадежностью линии передачи данных, и размером длины очереди на интерфейсе коммутатора. Так как при моделировании выбирался самый простой тип дис-

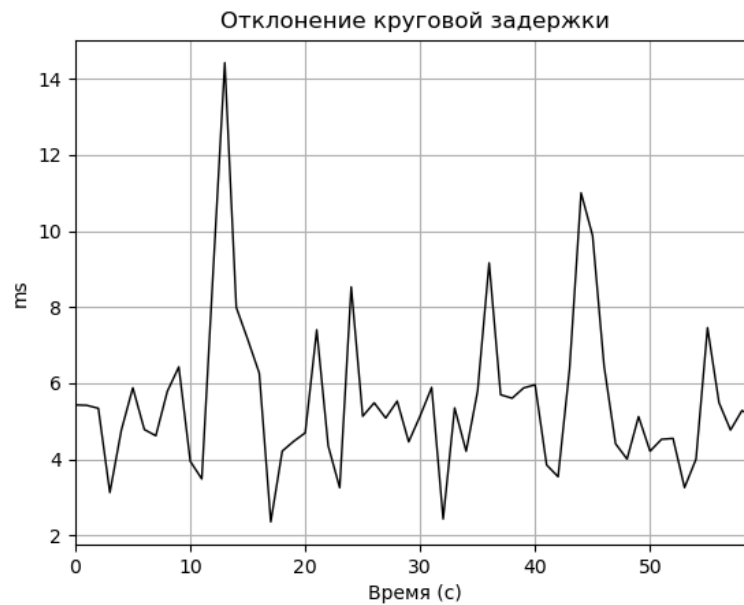


Рис. 3.8. График изменения значения вариации RTT с течением времени

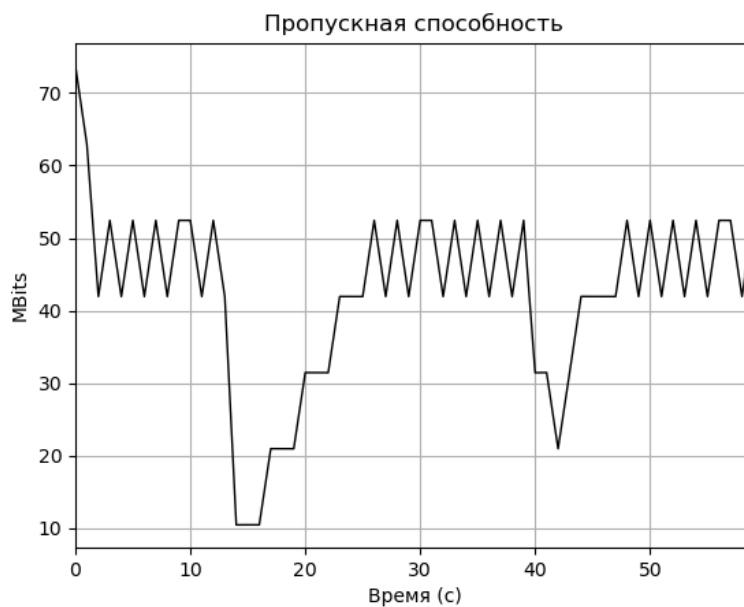


Рис. 3.9. График изменения значения пропускной способности с течением времени

циплины очередей droptail (pfifo), то на графике изменения длины очереди можно увидеть, что пакеты начинают сбрасываться при достижении критического значения длины очереди (в нашем случае 60 пакетов). Естественно, что при увеличении длины очереди пакеты достигают получателя медленнее и

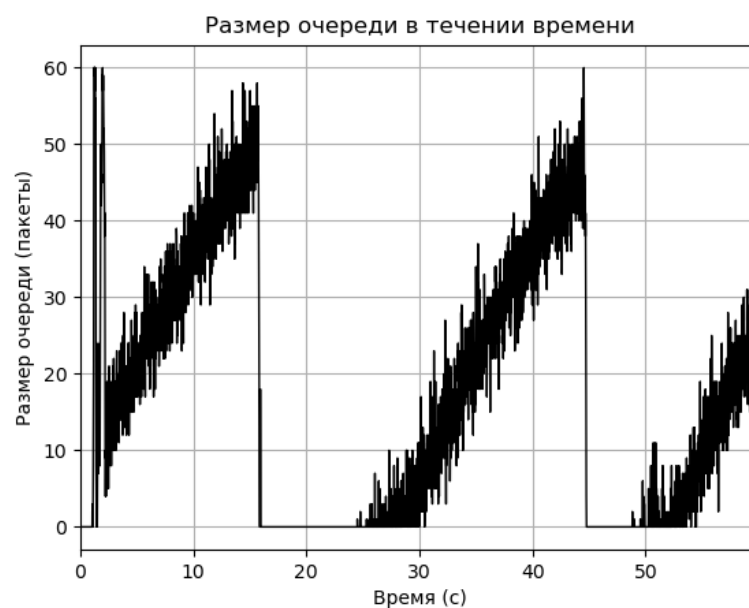


Рис. 3.10. График изменения длины очереди на интерфейсе s2-eth2

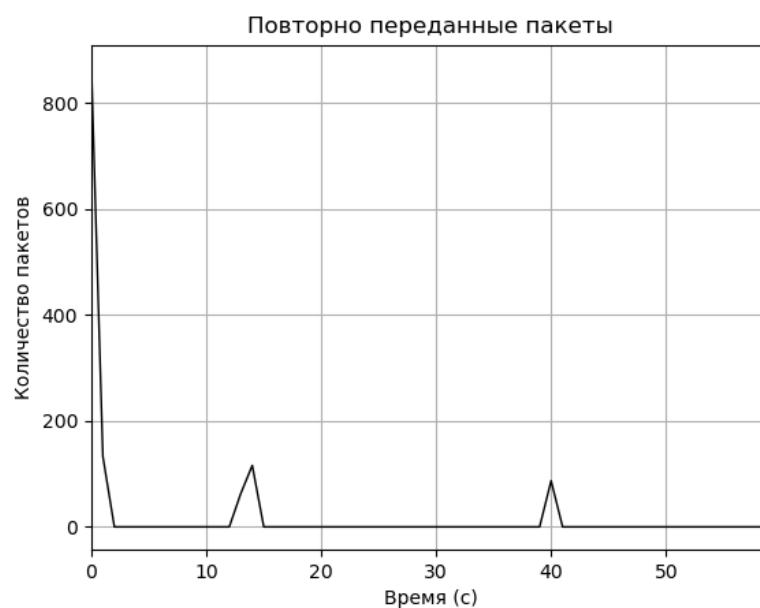


Рис. 3.11. График изменения значения повторно переданных пакетов во времени

значение RTT растет пропорционально увеличению этой самой очереди.

3.3. Пример простой сети с несколькими хостами

Сеть, в которой существуют только хост-сервер и хост-клиент встречаются довольно редко. Требуется усложнить модель сети, добавив в нее несколько хостов, которые в некоторый промежуток времени начинают передавать данные на сервер. Реализовать временные задержки довольно легко, если вспомнить про существование скриптовых файлов, которые можно запустить на хостах. Внутри данного файла будет настройка хоста, подключение к серверу и симуляция задержек во времени с помощью утилиты `sleep` [25].

Пусть у нас имеется сеть, заданная топологией, приведенной на рис.3.12.

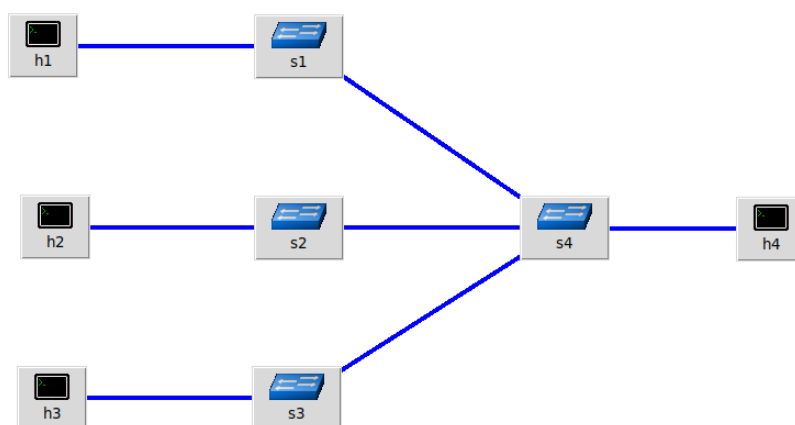


Рис. 3.12. Топология исследуемой сети

Предполагается, что в данной сети будут действовать следующие правила:

- скорость передачи данных ограничена;
- максимальная пропускная способность соединения $s4-s1$ равна 15 Мбит/с;
- максимальная пропускная способность соединения $s4-s2$ равна 10 Мбит/с;

- максимальная пропускная способность соединения s4-s3 равна 20 Мбит/с;
- максимальная пропускная способность соединения s4-h4 равна 20 Мбит/с;
- на коммутаторе s4 стоит дисциплина обработки очередей FIFO с максимальным количеством пакетов, равным 45;
- потери в сети составляют 0.001%;
- задержка имеет нормальное распределение с математическим ожиданием в 10 мс и с дисперсией в 3 мс.
- в сети работает алгоритм для работы с перегрузками TCP BBR.

Предполагается, что h2 начнет передавать данные на сервер через 10 секунд после старта сети и время передачи равняется 20 секундам, а h3 начнет передавать данные через 20 секунд с таким же временем передачи. В данной сети коммутаторы s1, s2, s3 существуют лишь для технической реализации сетевых задержек, потерь и ограничения скорости передачи данных.

Опишем данные характеристики в конфигурационном файле.

```
1  '''
2  # device settings
3  [devices]
4      [devices.h1]
5          name = "h1"
6          ip = "10.0.0.1"
7          cmd = [
8              "sysctl -w net.ipv4.tcp_congestion_control=bbr"
9          ]
10     [devices.h2]
11         name = "h2"
12         ip = "10.0.0.2"
13         cmd = [
14             "./config/host_configs/h2.sh"
15         ]
16     [devices.h3]
17         name = "h3"
18         ip = "10.0.0.3"
```



```

19         cmd = [
20             "./config/host_configs/h3.sh"
21         ]
22     [devices.h4]
23         name = "h4"
24         ip = "10.0.0.4"
25         cmd = [
26             "iperf3 -s -p 7778 -1",
27             "iperf3 -s -p 7779 -1",
28         ]
29
30
31     # switch settings
32     [switches]
33         [switches.s1]
34             name = "s1"
35         [switches.s2]
36             name = "s2"
37         [switches.s3]
38             name = "s3"
39         [switches.s4]
40             name = "s4"
41
42
43     # link settings
44     [links]
45     pairs = [
46         ["h1", "s1"],
47         ["h2", "s2"],
48         ["h3", "s3"],
49         ["s1", "s4"],
50         ["s2", "s4"],
51         ["s3", "s4"],
52         ["s4", "h4"]
53     ]
54     cmd = [
55         "tc qdisc replace dev s4-eth4 root handle 10: tbf rate 20mbit burst 10000 limit
56         ↪ 30000",
57         "tc qdisc replace dev s4-eth1 root handle 10: tbf rate 20mbit burst 10000 limit
58         ↪ 30000",
59         "tc qdisc replace dev s4-eth2 root handle 10: tbf rate 20mbit burst 10000 limit
60         ↪ 30000",

```

```

58     "tc qdisc replace dev s4-eth3 root handle 10: tbf rate 20mbit burst 10000 limit
    ↪ 30000",
59     "tc qdisc add dev s4-eth4 parent 10: handle 15: pfifo limit 45",
60
61     "tc qdisc replace dev s1-eth2 root handle 10: tbf rate 15mbit burst 7500 limit 22500",
62     "tc qdisc replace dev s2-eth2 root handle 10: tbf rate 10mbit burst 5000 limit 15000",
63     "tc qdisc replace dev s3-eth2 root handle 10: tbf rate 20mbit burst 10000 limit
    ↪ 30000",
64
65     "tc qdisc replace dev s1-eth1 root handle 10: tbf rate 15mbit burst 7500 limit 22500",
66     "tc qdisc replace dev s2-eth1 root handle 10: tbf rate 10mbit burst 5000 limit 15000",
67     "tc qdisc replace dev s3-eth1 root handle 10: tbf rate 20mbit burst 10000 limit
    ↪ 30000",
68
69
70     "tc qdisc add dev s1-eth2 parent 10: handle 20: netem loss 0.001% delay 10ms 3ms
    ↪ distribution normal",
71     "tc qdisc add dev s2-eth2 parent 10: handle 20: netem loss 0.001% delay 10ms 3ms
    ↪ distribution normal",
72     "tc qdisc add dev s3-eth2 parent 10: handle 20: netem loss 0.001% delay 10ms 3ms
    ↪ distribution normal",
73
74     "tc qdisc add dev s1-eth1 parent 10: handle 20: netem loss 0.001% delay 10ms 3ms
    ↪ distribution normal",
75     "tc qdisc add dev s2-eth1 parent 10: handle 20: netem loss 0.001% delay 10ms 3ms
    ↪ distribution normal",
76     "tc qdisc add dev s3-eth1 parent 10: handle 20: netem loss 0.001% delay 10ms 3ms
    ↪ distribution normal"
77
78 ]
79
80 [monitoring]
81 monitoring_time = 60
82 monitoring_interval = 0.05
83 host_client = "h1"
84 host_server = "h4"
85 interface = "s4-eth4"
86 iperf_file_name = "iperf.json"
87 iperf_flags = "-b 15mbit"
88 queue_data_file_name = "qlen.data"
89 plots_dir = "plots_dir_second"

```

Видно, что здесь, в разделе настроек хостов, у хоста h4 запускаются 2 iperf-сервера. К ним, через скрипт-файлы, подключатся хосты h2 и h3.

Скрипт-файл для хоста h2:

```
1 #!/bin/bash
2 sysctl -w net.ipv4.tcp_congestion_control=bbr
3 sleep 10
4 iperf3 -c 10.0.0.4 -p 7778 -t 20
```

Скрипт-файл для хоста h3:

```
1 #!/bin/bash
2 sysctl -w net.ipv4.tcp_congestion_control=bbr
3 sleep 20
4 iperf3 -c 10.0.0.4 -p 7779 -t 20
```

Запустим данную сеть и рассмотрим наиболее интересующие нас графики сетевых характеристик.

Динамика изменения пропускной способности показана на рис. 3.13. Видно, что наименьшее значение пропускной способности достигалось в промежутке, когда одновременно все хосты передавали данные на сервер. Помимо этого, на графике четко видно сколько хостов ведут передачу данных.

На рис. 3.14 приведен график изменения значений окна перегрузки TCP BBR (CWND). На графике видно, что наибольшее значение CWND достигается только тогда, когда 1 хост передает данные на сервер, а все остальные молчат.

Динамика значений RTT демонстрируется на рис 3.15. Видно, RTT увеличивается пропорционально росту окна перегрузки.

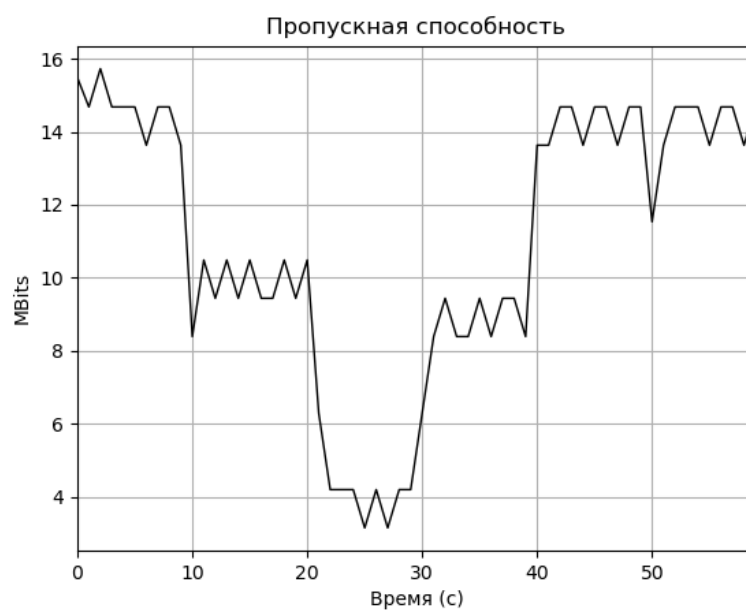


Рис. 3.13. График изменения значения пропускной способности с течением времени

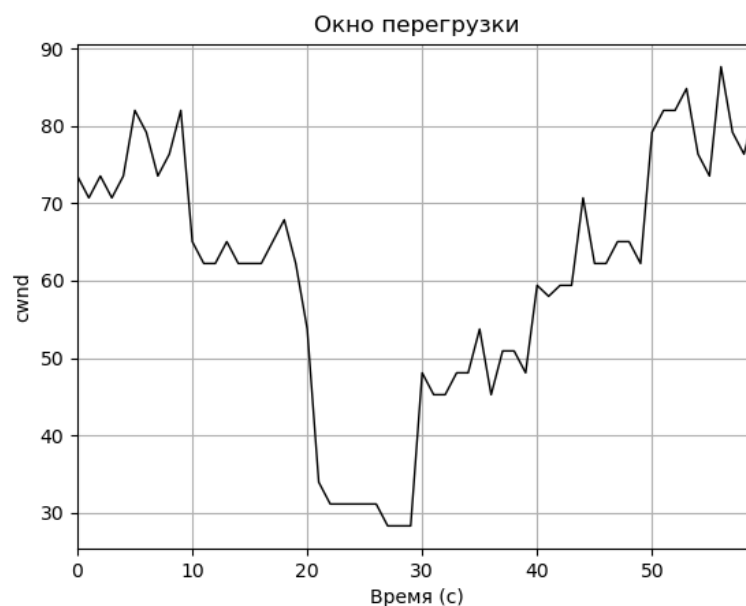


Рис. 3.14. График изменения значения окна перегрузки с течением времени при использовании алгоритма TCP Reno

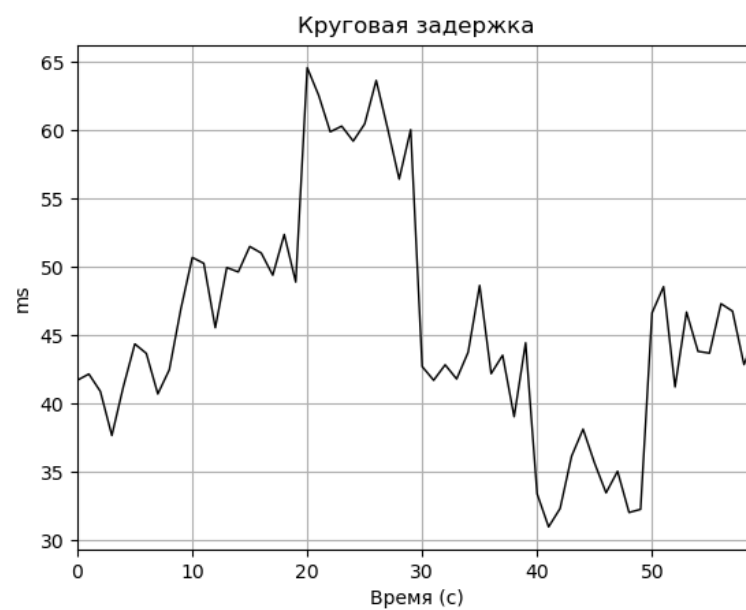


Рис. 3.15. График изменения значения RTT с течением времени

Заключение

В ходе данной работы был дан литературный обзор, посвященный исследованиям в области изучения возможностей современных сетей и возможный переход на сети нового поколения. Данные сети, именуемые как SDN, отлично подходят для современных реалий, так как они ориентированы на программы и предоставляемый уровень качества обслуживания данных программ. Программно-определяемые сети могут применяться в облачных вычислениях, в интернете вещей [8], в условиях крупных центров обработки данных, позволяя сократить издержки на сопровождение сети за счёт централизации управления потоками трафика на программном контроллере и повысить процент использования ресурсов сети благодаря динамическому управлению. Помимо этого, в литературном обзоре затрагиваются возможности построения виртуальных классических сетей и сетей нового поколения, а также измерения их сетевых характеристик с помощью программного комплекса Mininet.

Mininet — это виртуальная среда, которая позволяет разрабатывать и тестировать сетевые инструменты и протоколы. В сетях Mininet работают реальные сетевые приложения Unix/Linux, а также реальное ядро Linux и сетевой стек. С помощью одной команды Mininet может создать виртуальную сеть на любом типе машины, будь то виртуальная машина, размещенная в облаке или же собственный персональный компьютер. Это дает значительные плюсы при тестировании работоспособности протоколов или сетевых программ:

- позволяет быстро создавать прототипы программно-определяемых сетей;
- тестирование не требует экспериментов в реальной сетевой среде, вследствие чего разработка ведется быстрее;
- тестирование в сложных сетевых топологиях обходится без необходимости покупать дорогое оборудование;

- виртуальный эксперимент приближен к реальному, так как Mininet запускает код на реальном ядре Linux;
- позволяет работать нескольким разработчикам в одной топологией независимо.

С помощью программ, написанных на языке программирования Python, можно создавать виртуальные сети и проводить внутри них некоторые эксперименты. Для этого достаточно установить на компьютер с помощью `pip` пакет `mininet` и следовать API, которой предоставляет Mininet. В главе 2 был рассмотрен такой метод построения сети.

В данной работе также были проведены 2 эксперимента, в ходе которых были получены измерения сетевых характеристик и построены графики данных сетевых характеристик. В третьей главе был создан программный комплекс, который позволяет проводить автоматизированные эксперименты, управляя поведением сети через конфигурационный `toml`-файл, не прибегая к редактированию кода. В ходе таких экспериментов было рассмотрено поведение потоков данных при различных условиях и зависимость одной сетевой характеристики от остальных.

Однако, данный способ манипуляций с сетью может быть не удобен. В ходе работы приходилось обращаться к некоторым вспомогательным утилитам системы на базе ядра Linux, чтобы симулировать поведенческие особенности сетевых компонентов. Например, был описан метод симуляции потерь и задержек поступления пакетов с помощью дисциплины очередей NetEm. Такие особенности системы ведут к общему усложнению построения сети и необходимости узнавать множество дополнительной информации, которая не решает доменную проблему. Данные особенности не встречаются в других сетевых симуляторах. Например, в системах `ns-3` [17] или `gns3` [6].

Таким образом, в рамках данной работы

1. Рассмотрено применение системы Mininet для исследований производительности сетевых компонентов;

2. Представлены технологии, которые позволяют производить измерения сетевых характеристик, и технологии, которые позволяют визуализировать полученные данные;
3. Построен программный комплекс, который использует Mininet, iperf и iproute для создания виртуальной сети и анализа производительности ее сетевых компонентов.

Список литературы

1. A Performance Evaluation of TCP BBRv2 Alpha / J. Gomez [и др.] // 2020 43rd International Conference on Telecommunications and Signal Processing (TSP). — IEEE, 2020.
2. BBR: Congestion-Based Congestion Control / N. Cardwell [и др.] // ACM Queue. — 2016. — Т. 14. — С. 20—53.
3. Command-line interface. — URL: https://en.wikipedia.org/wiki/Command-line_interface.
4. CUBIC for Fast Long-Distance Networks : RFC / I. Rhee [и др.] ; RFC Editor. — 02.2018. — № 8312. — URL: <https://www.rfc-editor.org/rfc/rfc8312.txt>.
5. *Floyd S.* Metrics for the Evaluation of Congestion Control Mechanisms : RFC / RFC Editor. — 03.2008. — № 5166. — URL: <https://www.rfc-editor.org/rfc/rfc5166.txt>.
6. gns3. — URL: <https://www.gns3.com>.
7. Implementation of simplified custom topology framework in Mininet / C. Pal [и др.] // 2014 Asia-Pacific Conference on Computer Aided System Engineering (APCASE). — 2014.
8. Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios / K. Shafique [и др.] // Information and Telecommunication Sciences. — 2020.
9. ip(8) — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/tc.8.html>.
10. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. — URL: <https://iperf.fr/iperf-doc.php>.

11. iPerf 3 user documentation. — URL: <https://iperf.fr/iperf-doc.php>.
12. iproute2. — URL: <https://en.wikipedia.org/wiki/Iproute2>.
13. *Kaur K., Singh J., Ghumman N.* Mininet as Software Defined Networking Testing Platform // International Conference on COMMUNICATION, COMPUTING & SYSTEMS (ICCCS-2014). — EXCEL INDIA PUBLISHERS, 2014. — С. 139—142.
14. Link Layer Discovery Protocol. — URL: https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol.
15. Mininet. — URL: <http://mininet.org/>.
16. Mininet Python API Reference Manual. — URL: <http://mininet.org/api/annotated.html>.
17. ns-3 - Network Simulator. — URL: <https://www.nsnam.org>.
18. Open Networking Foundation. — URL: <https://opennetworking.org>.
19. OpenFlow. — URL: <https://opennetworking.org/sdn-resources/customer-case-studies/openflow/>.
20. Performance Analysis of Congestion Control Mechanism in Software Defined Network (SDN) / M. Z. A. Rahman [и др.] // — 2017.
21. pip. — URL: <https://pypi.org/project/pip/>.
22. RESEARCH OF SDN NETWORK PERFORMANCE PARAMETERS USING MININET NETWORK EMULATOR / O. I. Romanov [и др.] // Information and Telecommunication Sciences. — 2021.
23. Round Trip Time. — URL: [https://developer.mozilla.org/en-US/docs/Glossary/Round_Trip_Time_\(RTT\)](https://developer.mozilla.org/en-US/docs/Glossary/Round_Trip_Time_(RTT)).
24. *Shivayogimath C. N., Reddy N. V. U.* Performance Analysis of a Software Defined Network Using Mininet // Artificial Intelligence and Evolutionary Computations in Engineering Systems: Proceedings of ICAIECES 2015 / под ред. S. S. Dash [и др.]. — Springer, 2016. — С. 391—398.

25. `sleep(3)` — Linux manual page. — URL: <https://man7.org/linux/man-pages/man3/sleep.3.html>.
26. Software-defined networking. — URL: https://en.wikipedia.org/wiki/Software-defined_networking.
27. `ss(8)` — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/ss.8.html>.
28. *Stevens W. R.* TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms : RFC / RFC Editor. — 01.1997. — № 2001. — URL: <https://datatracker.ietf.org/doc/html/rfc2001>.
29. `tc-netem(8)` — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/tc-netem.8.html>.
30. `tc-tbf(8)` — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/tc-tbf.8.html>.
31. `tc(8)` — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/tc.8.html>.
32. TCP BBR v2 Alpha/Preview Release. — URL: <https://github.com/google/bbr/blob/v2alpha/README.md>.
33. TOML. — URL: <https://toml.io/en/>.
34. *Наливайко С. М.* Автоматизация процессов моделирования и измерения сетевых характеристик в Mininet // Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем: материалы Всероссийской конференции с международным участием. — Москва, РУДН, 2022. — С. 397—403.

A. Класс NetStatsPlotter

Данный класс представляет собой набор средств для анализа файла с сырыми данными сетевых характеристик и построения графиков данных сетевых характеристик.

```
1 import json
2 import matplotlib.pyplot as plt
3
4 class NetStatsPlotter:
5     def __init__(self, save_folder, plot_format):
6         self.save_folder = save_folder
7         self.plot_format = plot_format
8
9     # Построение графиков сетевых характеристик из iperf
10    def plot_net_stats(self, net_data_file):
11        x_stats, y_stats = self.__parse_net_stats_file(net_data_file)
12        for i in y_stats:
13            plt.plot(x_stats, y_stats[i][0], 'k', linewidth=1)
14            plt.grid()
15            plt.xlim(xmin=0, xmax=x_stats[-1])
16            plt.xlabel(y_stats[i][1]["x"])
17            plt.ylabel(y_stats[i][1]["y"])
18            plt.title(y_stats[i][1]["title"])
19            plt.savefig("{} / {}.{}".format(self.save_folder, i, self.plot_format))
20            plt.clf()
21
22    # Построение графика изменения длины очереди
23    def plot_queue_len(self, qlen_data_file):
24        x_stats, y_stats = self.__parse_queue_len_data_file(qlen_data_file)
25        plt.plot(x_stats, y_stats, 'k', linewidth=1)
26        plt.grid()
27        plt.xlim(xmin=0, xmax=x_stats[-1])
28        plt.xlabel("Время (с)")
29        plt.ylabel("Размер очереди (пакеты)")
30        plt.title("Размер очереди в течении времени")
31        plt.savefig("{} / queue_len.{}".format(self.save_folder, self.plot_format))
32
33    # Анализ сырого файла, полученного от утилиты iperf
34    def __parse_net_stats_file(self, net_data_file):
```

```

35     net_data = open(net_data_file, "r")
36     raw_data = json.load(net_data)
37     x = []
38     y = {
39         "bytes": [[], {"title": "Количество переданных байт", "x": "Время (с)", "y":
40             ↪ "MB"}],
41         "cwnd": [[], {"title": "Окно перегрузки", "x": "Время (с)", "y": "cwnd"}],
42         "MTU": [[], {"title": "Максимальный размер пакета", "x": "Время (с)", "y":
43             ↪ "B"}],
44         "retransmits": [[], {"title": "Повторно переданные пакеты", "x": "Время (с)",
45             ↪ "y": "Количество пакетов"}],
46         "rtt": [[], {"title": "Круговая задержка", "x": "Время (с)", "y": "ms"}],
47         "rttvar": [[], {"title": "Отклонение круговой задержки", "x": "Время (с)",
48             ↪ "y": "ms"}],
49         "throughput": [[], {"title": "Пропускная способность", "x": "Время (с)", "y":
50             ↪ "MBits"}]
51     }
52     for i in raw_data["intervals"]:
53         tmp_data = i["streams"][0]
54         x.append(tmp_data["start"])
55         y["bytes"][0].append(tmp_data["bytes"] / 1024 / 1024)
56         y["cwnd"][0].append(tmp_data["snd_cwnd"] / 1024)
57         y["MTU"][0].append(tmp_data["pmtu"])
58         y["retransmits"][0].append(tmp_data["retransmits"])
59         y["rtt"][0].append(tmp_data["rtt"] / 1000)
60         y["rttvar"][0].append(tmp_data["rttvar"] / 1000)
61         y["throughput"][0].append(tmp_data["bits_per_second"] / 1000000)
62     return [x, y]
63
64 # Анализ сырого файла длины очереди
65 def __parse_queue_len_data_file(self, qlen_data_file):
66     qlen_data = open(qlen_data_file, "r")
67     x_stats = []
68     y_stats = []
69     for line in qlen_data:
70         line = line.split(" ")
71         x_stats.append(float(line[0]))
72         y_stats.append(float(line[1]))
73     return [x_stats, y_stats]

```

В. Класс Monitor

Данный класс представляет собой средство мониторинга сетевых характеристик. Метод **net_monitoring** запускает методы **__queue_len_monitoring** и **__iperf_monitoring**. **__queue_len_monitoring** следит за размером очереди qdisc заданного интерфейса и записывает данные в файл. **__iperf_monitoring** запускает программу iperf на сервере и клиенте и ведет сбор статистики в заданный файл json.

```
1 from time import sleep
2 from subprocess import *
3 from threading import Thread
4 import re
5 import os
6
7 from plotting.NetStatsPlotter import NetStatsPlotter
8
9 class Monitor:
10
11     def __init__(self, host, server, iface, save_dir="monitoring_plots"):
12
13         self.save_dir = save_dir
14         if not os.path.exists(self.save_dir):
15             os.makedirs(self.save_dir)
16             os.system("chmod 777 {}".format(save_dir))
17         self.host = host
18         self.server = server
19         self.iface = iface
20
21     # Запуск системы мониторинга
22     def net_monitoring(self, iperf_file, iperf_commands,
23                       qlen_file, qlen_mon_time, qlen_mon_interval):
24         th1 = Thread(target=self.__iperf_monitoring,
25                     args=(iperf_file, iperf_commands,))
26         th2 = Thread(target=self.__queue_len_monitoring,
27                     args=(qlen_mon_time, qlen_mon_interval, qlen_file))
28         th1.start()
29         th2.start()
```

```

30     th1.join()
31     th2.join()
32     print("Мониторинг окончен. Строим графики.")
33     plotter = NetStatsPlotter(self.save_dir, "png")
34     plotter.plot_net_stats(os.path.join(self.save_dir, iperf_file))
35     plotter.plot_queue_len(os.path.join(self.save_dir, qlen_file))
36     print("Графики построены и находятся в директории {}".format(self.save_dir))
37
38     # Мониторинг очереди сетевого интерфейса
39     def __queue_len_monitoring(self, time=1, interval_sec_=0.1, fname="qlen.dat"):
40         print("Начало мониторинга сети на интерфейсе {}. Продолжительность мониторинга: "
41               "{} сек. с интервалом {}".format(self.iface, time, interval_sec_))
42         current_time = 0
43         # Регулярное выражение для поиска данных с tc
44         pat_queued = re.compile(r'backlog\s[^\s]+\s([\d]+)p')
45         cmd = "tc -s qdisc show dev {}".format(self.iface)
46         # Открытие файла мониторинга на запись
47         file = open("{}{}".format(self.save_dir, fname), 'w')
48         # Цикл, в котором происходит мониторинг до прерывания
49         while current_time < time:
50             # Вызов команды в tc в терминале и поиск значения длины очереди, количества
51             ↪ отброшенных пакетов
52             p = Popen(cmd, shell=True, stdout=PIPE)
53             output = p.stdout.read().decode('utf-8')
54             matches_queue = pat_queued.findall(output)
55             if matches_queue:
56                 t = "%f" % current_time
57                 current_time += interval_sec_
58                 file.write(t + ' ' + matches_queue[-1] + " " + '\n')
59                 sleep(interval_sec_)
60                 current_time += interval_sec_
61             os.system("chmod 777 {}".format(self.save_dir, fname))
62             file.close()
63
64     # Запуск утилиты iperf
65     def __iperf_monitoring(self, file_name, params):
66         print("Начало работы iperf. Хост: {}, сервер: {}. "
67               "Файл с данными: {}".format(self.host.name, self.server.name, self.save_dir, file_name))
68
69         self.server.popen("iperf3 -s -p 7777 -1")
70         self.host.cmd("iperf3 -c {} -p 7777 {} -J > {}".format(self.server.IP(), params, self.save_dir, file_name))
71

```

```
os.system("chmod 777 {}/{}/{}".format(self.save_dir, file_name))
```

С. Класс CustomTopology

Данный класс является описанием топологии сети. Внутри класса анализируется файл конфигурации и на его основе создаются сетевые элементы.

```
1 from mininet.topo import Topo
2
3
4 class CustomTopology(Topo):
5
6     # Считывание конфигурационного файла и
7     # наполнение топологии сетевыми элементами
8     def __init__(self, config, **opts):
9         super(CustomTopology, self).__init__(**opts)
10
11         for i in config["devices"]:
12             current = config["devices"][i]
13             self.addHost(name=current["name"], ip=current["ip"])
14         for i in config["switches"]:
15             current = config["switches"][i]
16             self.addSwitch(name=current["name"])
17
18         for i in config["links"]["pairs"]:
19             self.addLink(i[0], i[1])
```

D. Класс CustomModel

Класс **CustomModel** является основным во всей иерархии классов программы. Он включает в себя классы CustomTopology, Monitor и NetStatsPlotter для создания виртуальной сети, мониторинга ее сетевых характеристик и построения графиков данных сетевых характеристик.

```
1  import os
2
3  import toml
4  from mininet.link import TCLink
5  from mininet.net import Mininet
6  from mininet.node import CPULimitedHost
7
8  from monitoring.Monitor import Monitor
9  from topology.CustomTopology import CustomTopology
10
11
12  class CustomModel:
13
14      def __init__(self):
15          self.topology_config = {}
16          self.devices_startup_configs = {}
17          self.links_config = []
18          self.monitoring_config = None
19
20      # Конфигурация сетевых элементов
21      def configure_model(self, config_file):
22          try:
23              file = toml.load(config_file)
24              self.monitoring_config = file["monitoring"]
25              self.topology_config = {"devices": file["devices"],
26                                     "switches": file["switches"],
27                                     "links": file["links"]}
28
29              self.links_config = file["links"]["cmd"]
30              for i in file["devices"]:
31                  self.devices_startup_configs[i] = file["devices"][i]["cmd"]
32
```

```

33         except FileNotFoundError:
34             print("Введите корректное имя файла")
35
36     # Симуляция модели
37     def simulation(self):
38         topology = CustomTopology(self.topology_config)
39         net = Mininet(topo=topology, host=CPULimitedHost, link=TCLink)
40         net.start()
41         try:
42             self.__configure_links()
43             self.__configure_devices(net.hosts)
44
45             h1, h2 = net.get(
46                 self.monitoring_config["host_client"],
47                 self.monitoring_config["host_server"]
48             )
49             monitor = Monitor(h1, h2, self.monitoring_config["interface"],
50                               self.monitoring_config["plots_dir"])
51             qlen_mon_time = self.monitoring_config["monitoring_time"]
52             iperf_file = self.monitoring_config["iperf_file_name"]
53             iperf_commands = self.monitoring_config["iperf_flags"] + " -t %d" % \
54                               qlen_mon_time
55             qlen_file = self.monitoring_config["queue_data_file_name"]
56             qlen_mon_interval = self.monitoring_config["monitoring_interval"]
57             monitor.net_monitoring(iperf_file, iperf_commands, qlen_file, qlen_mon_time,
58                                   ↪ qlen_mon_interval)
59         finally:
60             net.stop()
61
62     # Настройка соединений между сетевыми элементами
63     def __configure_links(self):
64         for i in self.links_config:
65             os.system(i)
66
67     # Настройка хостов
68     def __configure_devices(self, devices):
69         for i in devices:
70             command = self.devices_startup_configs["{}".format(i)]
71             for j in command:
72                 i.popen(j)

```
