

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.

КУРСОВАЯ РАБОТА
на тему
«Имитационное моделирование сети передачи данных в Mininet»
по дисциплине «Компьютерный практикум по математическому
моделированию»

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 2021

Содержание

Список используемых сокращений	4
Русскоязычные сокращения	4
Англоязычные сокращения	4
Введение	5
1. Построение виртуальной сети с помощью Mininet	7
1.1. Общие сведения	7
1.2. Установка Mininet	8
1.2.1. Указания к установке	8
1.2.2. Процесс установки	8
1.3. Работа с программой Mininet	9
1.4. MiniEdit	10
2. Генерация и измерение сетевого трафика с помощью утилиты iPerf3	12
2.1. Общие сведения	12
2.2. Тестирование пропускной способности с помощью iPerf3	12
2.3. Построение графиков сетевых характеристик	15
3. Использование утилиты iproute2 для настройки интерфейсов сетевых элементов	19
3.1. Общие сведения	19
3.2. Утилита tc	20
3.3. Виды дисциплин очередей	22
3.3.1. Бесклассовые очереди	22
3.3.2. Очереди с классами	22
3.4. Пример построения иерархии дисциплин	23
3.5. Измерение размера очереди на дисциплине	28
Заключение	32
Список литературы	33
А. Построение графиков сетевых характеристик из полученных данных iPerf3	34
В. Построение графика изменения длины очереди на сетевом интерфейсе	37

Список иллюстраций

1.1. Запуск Mininet	9
1.2. Проверка достижимости h2 для h1	10
1.3. Редактирование конфигурации хостов сети	11
1.4. Соединение элементов сети	11
1.5. Проверка достижимости h2 для h1	11
2.1. Сеть с простой топологией в Mininet	14
2.2. Запуск iPerf-клиента	15
2.3. Вывод статистики iPerf3	15
2.4. График изменения количества переданных данных с течением времени	16
2.5. График изменения значения окна перегрузки с течением времени при использовании алгоритма TCP Cubic	17
2.6. График изменения значения RTT с течением времени	17
2.7. График изменения значения вариации RTT с течением времени	18
2.8. График изменения значения пропускной способности с течением времени	18
3.1. Информация о дисциплине очереди на сетевом устройстве s1-eth0	21
3.2. Топология сети	24
3.3. Мониторинг сети	26
3.4. График изменения значения окна перегрузки TCP Reno	27
3.5. График изменения количества повторно переданных данных	27
3.6. График изменения значения RTT с течением времени	28
3.7. График изменения значения пропускной способности	29
3.8. Диаграмма активностей для функции замеров длины очереди	30
3.9. График изменения длины очереди на интерфейсе s2-eth2	31

Список используемых сокращений

Русскоязычные сокращения

ОС — Операционная система

Англоязычные сокращения

AMD — Advanced Micro Devices

CBQ — Class Based Queueing

CLI — Command-Line Interface

HTB — Hierarchy Token Bucket

IP — Internet Protocol

JSON — JavaScript Object Notation

PID — Process Identity Document

QoS — Quality of Service

RED — Random Early Detection

RTT — Round-Trip Time

SCTP — Stream Control Transmission Protocol

SFQ — Stochastic Fairness Queueing

SSD — Solid-State Drive

TBF — Token Bucket Filter

TCP — Transmission Control Protocol

UDP — User Datagram Protocol

Введение

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.

Целью работы является изучение возможностей измерения сетевых характеристик передачи данных без использования реальных сетевых компонентов и варьирование сетевых протоколов для достижения наилучшей производительности детерминированной сети.

Основные задачи моей работы:

1. Построить имитационную модель простой сети передачи данных в Mininet [4].
2. Измерить и визуализировать характеристики моделируемой сети передачи данных для качественной оценки производительности сети.
3. Оценить влияние различных комбинаций протоколов на общую производительность моделируемой сети при заданных сетевых характеристиках.

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

Курсовая работа состоит из введения, трех разделов, посвященных построению виртуальной сети, ее конфигурации и измерению сетевых характеристик передачи данных, заключения и списка используемой литературы.

Во введении кратко дается представление исследуемой области.

В первом разделе рассказывается о виртуальной среде Mininet, в которой

будет проводится исследование. Далее будет дан способ установки Mininet на свою рабочую станцию и будет осуществлен первый запуск виртуальной сети.

Во втором разделе дается представление о генерации трафика и его передачи между хостами с помощью утилиты iPerf3 [2]. Будут рассмотрены параметры запуска данной утилиты и способы визуализации сетевых характеристик передачи данных, полученных данной утилитой.

В третьем разделе рассматривается утилита iproute2 [3], которая помогает настраивать интерфейсы сетевых элементов. Далее рассматриваются виды дисциплин очередей, настройка дисциплин очередей на сетевых интерфейсах и получение статистических данных дисциплины с помощью iproute2.

В заключении подведены основные выводы проделанной работы.

1. Построение виртуальной сети с помощью Mininet

1.1. Общие сведения

Mininet [4] — 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.

1.2. Установка Mininet

1.2.1. Указания к установке

Описание выполнения установки приведено для техники со следующими характеристиками:

- ОС ubuntu-20.04.3
- AMD Ryzen 7 3700X 3600 MHz, 4Гб оперативной памяти, 20Гб свободно на SSD.

1.2.2. Процесс установки

1. Перейдем на сайт mininet.org.
2. Перейдем в раздел Download и выберем режим установки mininet. Mininet дает нам 3 опции по установке: установка образа посредством github, установка из исходного кода на персональный компьютер или установка пакетом для Ubuntu посредством apt. Мы выбрали второй вариант установки.
3. Установим исходный код программы mininet и перейдем в установленную директорию при помощи команды

```
git clone git://github.com/mininet/mininet && cd mininet
```

4. Произведем установку программы mininet при помощи команды

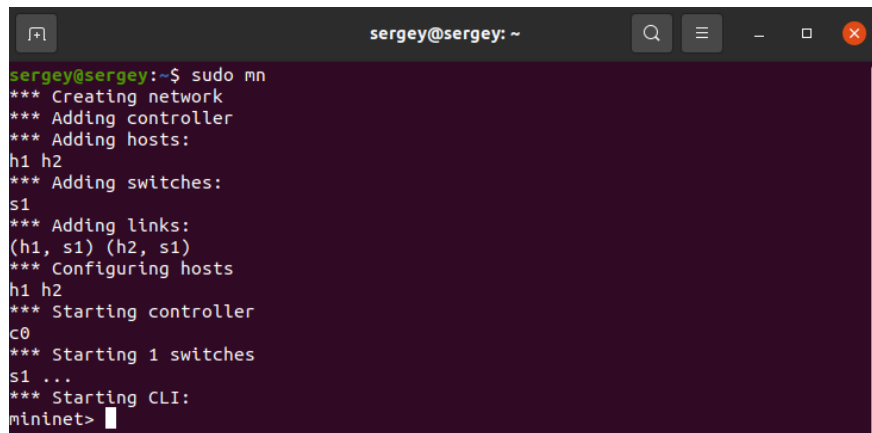
```
./util/install.sh -a
```

Опция -a в команде означает, что мы хотим установить программу полностью со всеми зависимостями (например POX, Open vSwitch, OpenFlow).

После установки в директории `/usr/local/lib/pythonV/dist-packages` появится каталог mininet.

5. Если в ходе установки не возникло ошибок, то можно запустить Mininet и проверить работоспособность при помощи команды `sudo mn`.

Вывод команды приведен на рис. 1.1. Mininet требует права суперпользователя для запуска.



```
sergey@sergey: ~  
sergey@sergey:~$ sudo mn  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet>
```

Рис. 1.1. Запуск Mininet

6. Удалим исходный код программы командой `rm -rf mininet`.

1.3. Работа с программой Mininet

После запуска Mininet посредством команды `sudo mn` у нас создается простая сеть с хостами `h1` и `h2` и коммутатором `s1`, который соединяет хосты между собой. Также хостам были присвоены `ip`-адреса (`10.0.0.1` для `h1` и `10.0.0.2` для `h2`). Мы можем проверить достижимость сетевых компонентов с помощью команды `h1 ping h2` посредством CLI или же запустить терминал для хоста, например, `h1` и выполнять всю работу из терминала, как на реальном устройстве. Запуск терминала осуществляется командой `xterm h1` из CLI.

Проверим достижимость `h2` для `h1` из терминала можно с помощью команды `ping` (рис. 1.2).

Узнать список доступных команд Mininet можно в CLI с помощью команды `help`.

```

"Node: 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=1.58 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.439 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.071 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3053ms
rtt min/avg/max/mdev = 0.063/0.539/1.583/0.621 ms

```

Рис. 1.2. Проверка достижимости h2 для h1

1.4. 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.

1. Запустим Miniedit:

```
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-адреса 10.0.0.1 и 10.0.0.2 соответственно (рис. 1.3).
4. Добавим в рабочую область коммутатор, выбрав элемент LegacySwitch.
5. Выберем элемент NetLink и соединим элементы сети (рис. 1.4).
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. 1.5).

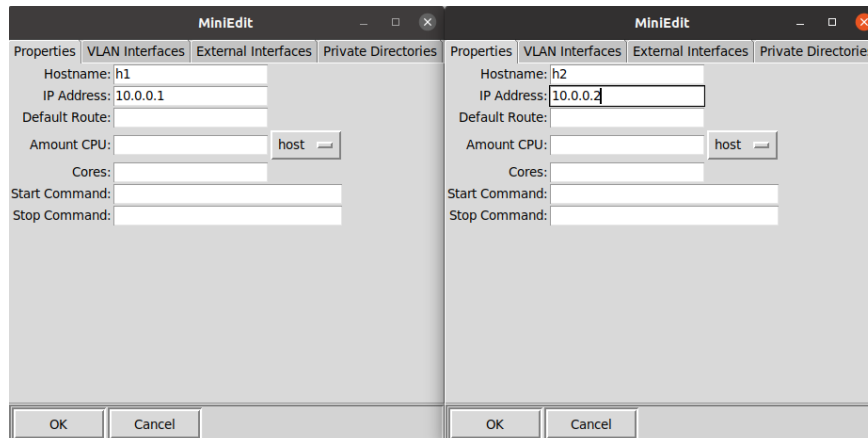


Рис. 1.3. Редактирование конфигурации хостов сети

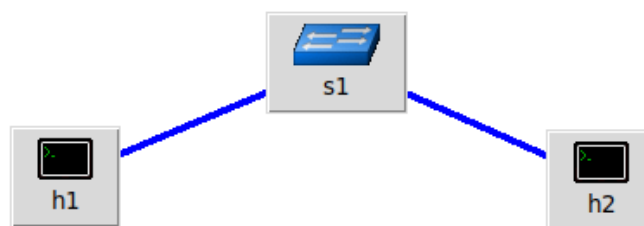


Рис. 1.4. Соединение элементов сети

```

"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. 1.5. Проверка достижимости h2 для h1

2. Генерация и измерение сетевого трафика с помощью утилиты iPerf3

2.1. Общие сведения

iPerf3 [2] — 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. Тестирование пропускной способности с помощью iPerf3

Для запуска сервера iPerf требуется выполнить следующую команду

```
iPerf3 -s [параметры]
```

Список параметров для сервера:

- -D, -daemon (запуск сервера как процесс демон);
- -I, -pidfile file (ведется запись в PID-файл);
- -1, -one-off (после обслуживания 1 клиента работа сервера прекращается).

Для запуска iPerf-клиента требуется выполнить следующую команду:

```
iPerf3 -c server_ip [параметры]
```

Список параметров для клиента:

- -u, -udp (используется);
- -b, -bandwidth (указывает пропускную способность);
- -t, -time (время передачи);
- -n, -bytes (количество байтов для передачи);
- -k, -blockcount (количество блоков данных для передачи);
- -l, -len (размер буфера для передачи/приема);
- -sport (указание порта передачи для клиента);
- -P, -parallel (количество потоков передачи);
- -R, -reverse (указание поменять сервер и клиент местами);
- -w, -window (изменение размера окна TCP);
- -M, -set-mss (изменение максимального размера сегмента);
- -N, -no-delay (удалить задержку в TCP/SCTP);
- -4, -version4 (использовать только ipv4);
- -6, -version6 (использовать только ipv6);
- -S, -tos N (установить тип услуги);
- -Z, -zerocopy (использовать метод отправки данных с нулевой копией);
- -O, -omit N (опустить первые n секунд передачи при статистике);
- -T, -title str (префикс каждой выходной строки с этой строкой);
- -get-server-output (получение результатов от сервера);
- -udp-counters-64bit (использовать 64-битные счетчики в тестовых пакетах UDP).

Общие параметры для сервера и клиента:

- -p, -port (порт для прослушивания);
- -f, -format (формат статистики: Мбит, Кбит ...);
- -i, -interval (интервалы между замерами);
- -F, -file name (xmit/recv файл);
- -B, -bind (связать определенный интерфейс);
- -V, -verbose (более детальная статистика);

- -J, -json (вывод статистики в json- файл);
- -logfile f (отправить данные в лог-файл);
- -d, -debug (выдать отладочную информацию);
- -v, -version (показать версию программы и выйти);
- -h, -help (показать help-информацию и выйти).

Проведем тестирование сети в Mininet:

1. Запустим MiniEdit, создадим два хоста (h1 и h2) и коммутатор (s1), соединим элементы сети, запустим сеть (рис. 2.1).

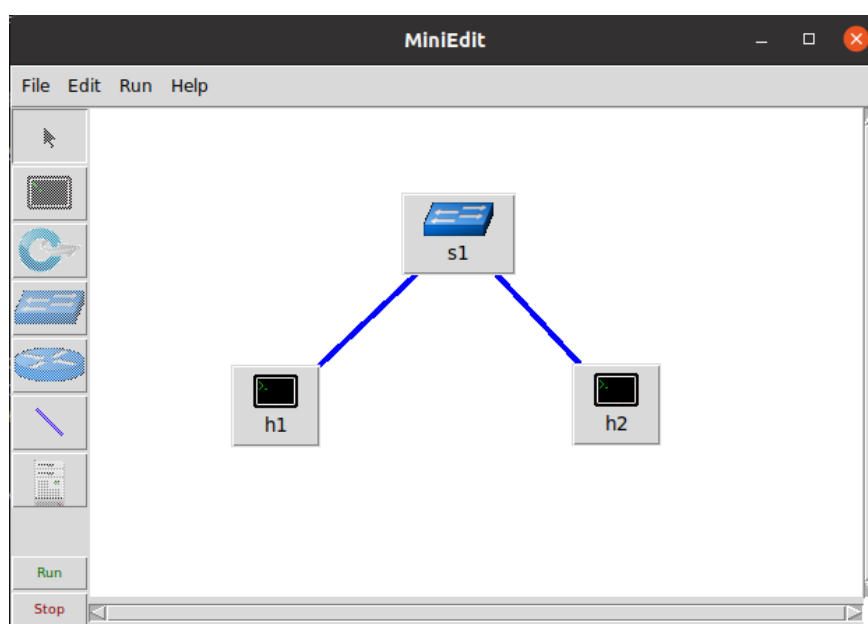


Рис. 2.1. Сеть с простой топологией в Mininet

2. Запустим iPerf-сервер на h2, введя в терминале этого устройства команду
`iperf3 -s`

3. Запустим iPerf-клиент на хосте h1 на 60 секунд, пропустив первые 10 секунд для статистики (рис. 2.2):

```
iPerf3 -c 10.0.0.2 -t 60 -O 10
```

4. Просмотрим статистику, которую сгенерировал iPerf (рис. 2.3).

"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. 2.2. 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. 2.3. Вывод статистики iPerf3

2.3. Построение графиков сетевых характеристик

iPerf3 предоставляет нам обширные количество данных, которые можно исследовать. Наиболее удобным представлением данных сетевых характеристик является графическое представление.

Для того, чтобы построить графики воспользуемся самописным скриптом `plotter.py`, который подробно рассмотрен в Приложении А. Данный скрипт обрабатывает файл `json`, который мы получаем в iPerf3, и на основе него строит графики.

Визуализируем данные сетевой статистики.

1. Воспользуемся виртуальной сетью из прошлого раздела.
2. Запустим iPerf-сервер на хосте h2.
3. Запустим iPerf3-клиент на хосте h1 с параметром `-J`, который формирует `json`-файл со статистикой:

```
iPerf3 -c 10.0.0.2 -t 60 -J > data.json
```

4. Дождемся окончания передачи данных и запустим скрипт из Приложения А:

```
./plotter.py -i data.json
```

В результате выполнения скрипта создается директория `monitoring`, в которой находятся все статистические данные iPerf.

5. Рассмотрим полученные графики:

На рис. 2.4 приведен график, показывающий динамику объема переданных данных за время моделирования.

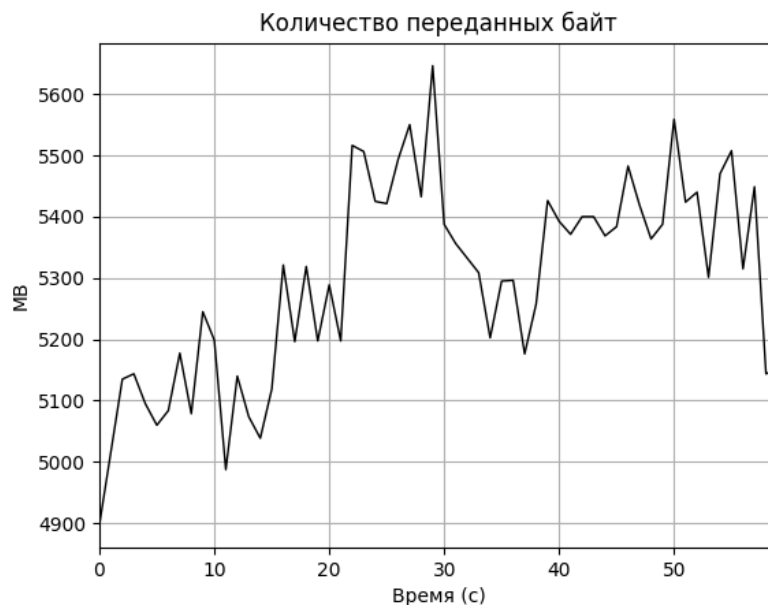


Рис. 2.4. График изменения количества переданных данных с течением времени

На рис. 2.5 приведен график изменения значений окна перегрузки протокола TCP Cubic (CWND).

Динамика значений RTT (измеряется в миллисекундах) демонстрируется на рис. 2.6.

График отклонений значений RTT приведен на рис. 2.7.

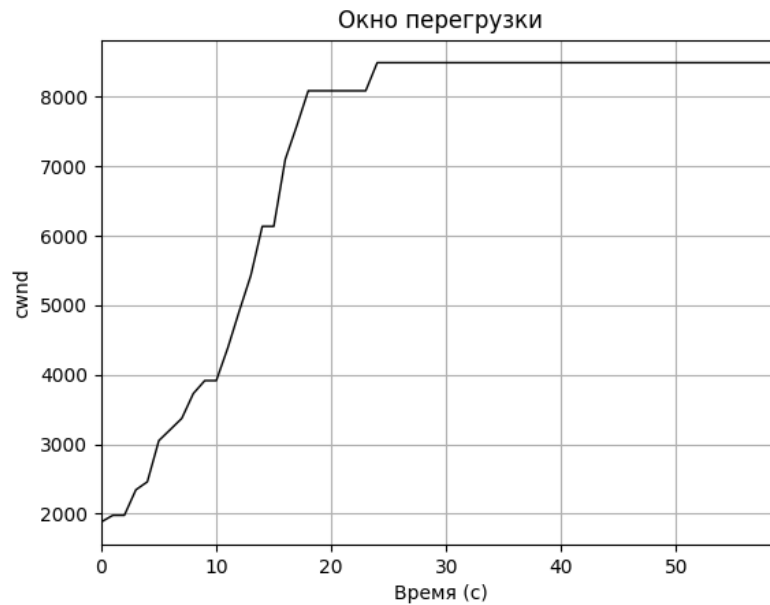


Рис. 2.5. График изменения значения окна перегрузки с течением времени при использовании алгоритма TCP Cubic

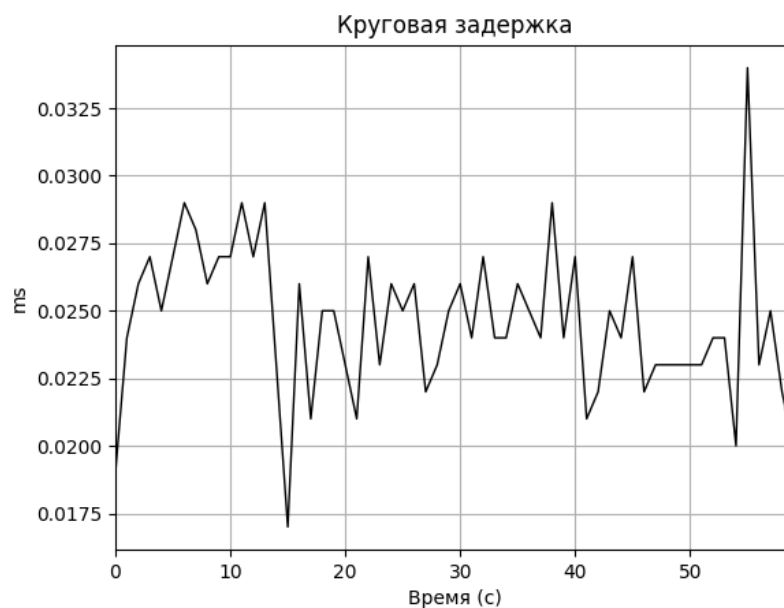


Рис. 2.6. График изменения значения RTT с течением времени

Динамика изменения пропускной способности показана на рис. 2.8. Видим, что данные передаются с максимально допустимой скоростью передачи на интерфейсе, чего нельзя добиться в реальных сетях.

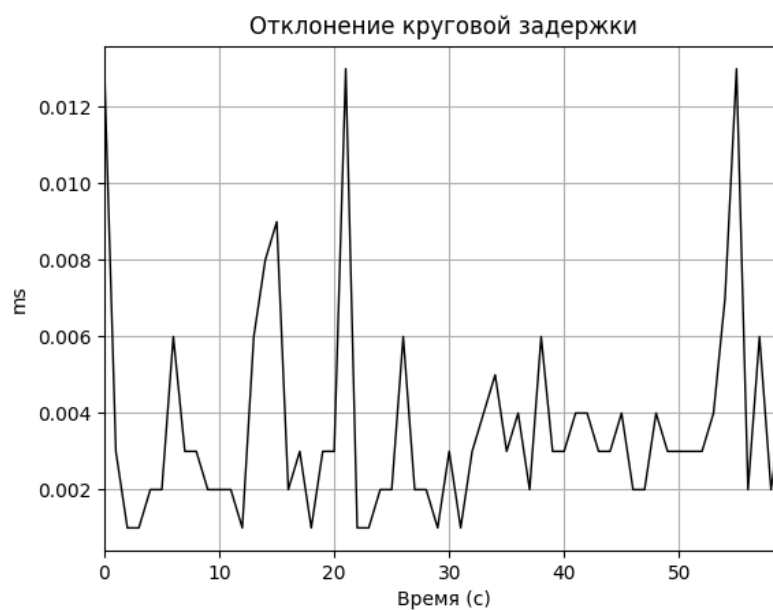


Рис. 2.7. График изменения значения вариации RTT с течением времени

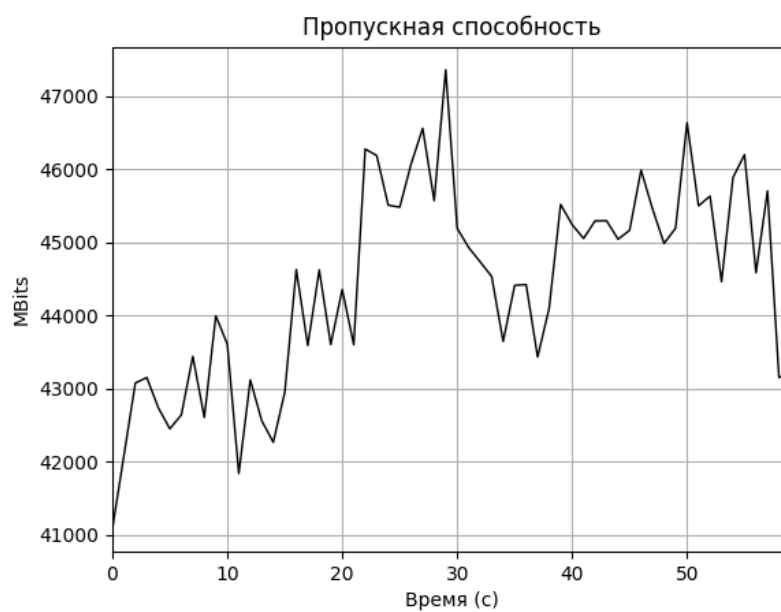


Рис. 2.8. График изменения значения пропускной способности с течением времени

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

3.1. Общие сведения

`iproute2` — это набор утилит для управления параметрами сетевых устройств в ядре Linux. Утилиты разработаны в качестве унифицированного интерфейса к ядру Linux, непосредственно управляющего 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 др., предназначенных для создания IP-туннелей. `iproute2` предлагает унифицированный синтаксис для управления аспектами сетевых интерфейсов.

Набор утилит включает в себя три основные программы:

- `ip` [1] — утилита для просмотра параметров и конфигурирования сетевых интерфейсов, сетевых адресов, таблиц и правил маршрутизации, ARP-таблиц, IP-туннелей, адресов multicast рассылки, маршрутизации multicast-пакетов;
- `tc` [6] — утилита для просмотра и конфигурирования параметров управления трафиком, позволяющая 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` [5] — утилита для просмотра текущих соединений и открытых портов.

3.2. Утилита tc

Утилита tc [6] наиболее полезна для исследования, так как она позволяет гибко настроить поведение контроля исходящего трафика. Система контроля трафика состоит из следующих компонентов.

- Ограничения исходящего трафика (SHAPING). 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 (SCHEDULING). Это 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 (POLICING). Этот механизм позволяет ограничить количество пакетов или байт в потоке входящего трафика, соответствующих определенной классификации.
- Отбрасывания (DROPPING). Трафик, превышающий установленную полосу пропускания, может быть отброшен как для входящего, так и исходящего трафика. Обработка трафика контролируется тремя типами объектов: очередями (qdiscs), классами и фильтрами. Обработка трафика контролируется тремя типами объектов: очередями (qdiscs), классами и фильтрами.

Для информации о tc используется команда `tc help`.

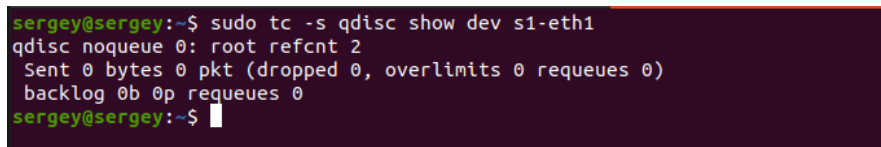
Дисциплина очереди — это алгоритм обработки очереди сетевых пакетов. Дисциплин на одном интерфейсе может быть задействовано несколько, а непосредственно к интерфейсу крепится корневая дисциплина (root qdisc). При этом каждый интерфейс имеет свою собственную корневую дисциплину.

Каждой дисциплине и каждому классу назначается уникальный дескриптор (некоторый номер), 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.

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. 3.1 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. 3.1. 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 0b 0p requeues 0* — размер очереди в байтах и пакетах.

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

3.3. Виды дисциплин очередей

3.3.1. Бесклассовые очереди

Ниже перечислены виды бесклассовых очередей:

- pfifo/bfifo — простейшие очереди без обработки, лишь реализующие принцип «First In, First Out» (первый вошел – первый вышел). Ограничены в пакетах или байтах.
- pfifo_fast — стандартная очередь ядра с ‘расширенным роутером’. Состоит из очереди с тремя потоками, учитывающей флаги типа услуги (Type of Service) и приоритет пакета.
- RED — Random Early Detection (случайное раннее обнаружение) симулирует физический затор, отбрасывая пакеты случайным образом при близком достижении заданной полосы пропускания. Хорошо подходит для приложений, требующих очень широкую полосу пропускания.
- SFQ — Stochastic Fairness Queueing (очередь равномерного случайного распределения пакетов) реорганизует трафик в очереди таким образом, что каждый ‘сеанс’ (‘session’, виртуальная подочередь) получает право отправить пакет, а маркер переходит на следующий сеанс.
- TBF — Token Bucket Filter (фильтр буфера токенов) удерживает скорость передачи пакетов на примерно постоянном уровне (меньшем, чем реальная скорость интерфейса). Хорошо масштабируется для широких каналов.

Подробнее о видах бесклассовых очередей можно прочесть в [6].

3.3.2. Очереди с классами

Ниже перечислены виды очередей с классами [6]:

- CBQ — очередь, базирующаяся на классах (Class Based Queueing), реализует мощную иерархию классов. Поддерживает ограничения и приоритеты. Разделение осуществляется по времени простоя канала, вычисляемого

на основании среднего размера пакета и полосы пропускания.

- НТВ — очередь (Hierarchy Token Bucket — иерархический буфер токенов) реализует мощную иерархию классов с упором на согласование с существующей практикой. НТВ обеспечивает гарантированную полосу пропускания для классов, также позволяет устанавливать верхние пределы межклассового разделения очереди. Содержит объекты ограничения, базирующиеся на TBF, и может устанавливать приоритеты для классов.
- PRIO — очередь может разделять трафик между тремя полосами, которые являются очередями любого типа. При извлечении пакета из очереди вначале исследуется подочередь с большим приоритетом, если в последней нет пакетов для обработки, то выбирается очередь с более низким приоритетом. Для установки приоритета используются биты типа услуги (Type of Service).

3.4. Пример построения иерархии дисциплин

Построим простую топологию сети. Пусть, у нас имеется 2 хоста соединенные с сетью. Сеть состоит из двух коммутаторов с разными скоростями передачи данных. Назовем сетевые элементы как h1 (хост-передатчик), h2 (хост-приемник), s1 (коммутатор, соединенный с h1) и s2(коммутатор, соединенный с h2). Тогда сеть будет иметь следующие характеристики:

- скорость передачи данных от s1 к h1 равна 100 Мбит/с;
- скорость передачи данных от s1 к s2 равна 100 Мбит/с;
- скорость передачи данных от s2 к h2 равна 50 Мбит/с;
- задержка распространения между h1 и s1 равна 30 мс +- 7 мс;
- задержка распространения между s1 и s2 равна 30 мс +- 7 мс;
- процент потерь пакетов на каждом сетевом соединении равен 0.01%;
- дисциплиной очереди, которая установлена на интерфейсе s2-eth2, будет pfifo с максимальным количеством пакетов 30.

Скорость передачи на интерфейсе можно ограничить с помощью дисциплины tbf, а эмулировать задержки распространения и потери мы будем с помощью дисциплины NetEm. NetEm обеспечивает функциональность сетевой эмуляции для тестирования протоколов, имитируя свойства глобальных сетей.

1. Откроем MiniEdit и создадим простую топологию сети (рис. 3.2).

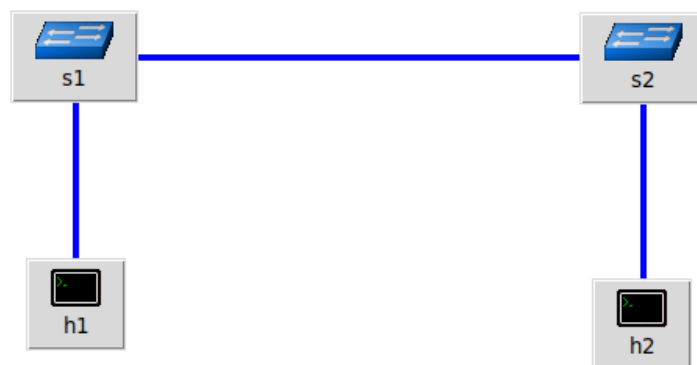


Рис. 3.2. Топология сети

2. Запустим сеть.
3. Введем следующие команды tc:

```
1  sudo tc qdisc replace dev s1-eth2 root handle 10: tbf rate 100mbit burst
   ↪ 50000 limit 150000
2  sudo tc qdisc add dev s1-eth2 parent 10: handle 20: netem loss 0.01%
   ↪ delay 30ms 7ms distribution normal
3  sudo tc qdisc replace dev s1-eth1 root handle 10: tbf rate 100mbit burst
   ↪ 50000 limit 150000
4  sudo tc qdisc add dev s1-eth1 parent 10: handle 20: netem loss 0.01%
   ↪ delay 30ms 7ms distribution normal
```



```

5  sudo tc qdisc replace dev s2-eth2 root handle 10: tbf rate 50mbit burst
    ↪ 25000 limit 75000
6  sudo tc qdisc add dev s2-eth2 parent 10: handle 15: pfifo limit 30
7  sudo tc qdisc replace dev s2-eth1 root handle 10: tbf rate 50mbit burst
    ↪ 25000 limit 75000

```

Любая команда добавления дисциплины на интерфейс, работающая с бесклассовыми дисциплинами, имеет вид:

```

tc qdisc [add/replace] dev [interface] [root/parent num] handle [number]:
    ↪ [Queue Discipline] [discipline parameters]

```

Параметры команды:

- [add/replace] — выбирается в зависимости от иерархии дисциплин. Если дисциплина еще не назначена — ставим add, если требуется заменить существующую дисциплину — ставим replace;
- [interface] — фактический сетевой интерфейс устройства;
- [root/parent num] — выбирается в зависимости от иерархии дисциплин. Если дисциплина должна быть назначена как корневая — ставим root, если дисциплина должна прикрепиться к родителю — ставим parent с дескриптором родителя;
- [number] — дескриптор, который мы назначаем на дисциплину;
- [Queue Discipline] — название дисциплины;
- [discipline parameters] — параметры дисциплины.

TBF — дисциплина, построенная на понятии токена и заполненности некоторого буфера, называемого ведром». Данные поступают на вход алгоритма, токены генерируются и поступают в «ведро». Если токен имеется в «ведре», то данные проходя на интерфейс и отправляются в сеть. Если «ведро» пусто, то пакеты ставятся в очередь. По достижению некоторого лимита очереди пакеты отбрасываются.

Параметры tbf:

- rate — фактическая скорость, с которой генерируются токены;
- burst — количество байтов, которое может поместиться в ведре;
- limit — размер очереди.

Параметры netem:

- loss — процент потерь на соединении;
 - delay — задержка распространения. В нашем примере запись *delay 30ms 7ms distribution normal* означает, что задержка равна 30 мс, при этом это значение варьируется от 23 до 37 мс с вероятностью, заданной нормальным распределением.
4. Запустим iPerf-клиент и iPerf-сервер (рис. 3.3). Запишем данные в файл json и построим графики сетевых характеристик.

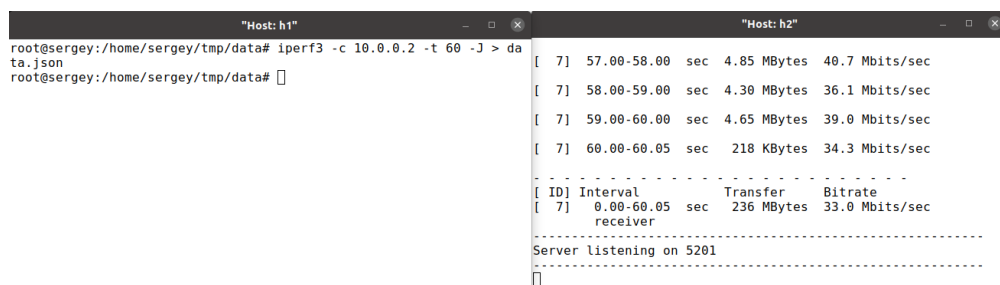


Рис. 3.3. Мониторинг сети

5. Рассмотрим, как изменилось поведение нашей сети в сравнении с из прошлым разделом, где рассматривалась идеальная сеть без заданных сетевых параметров (потерь, задержек, максимальной скорости передачи).

Окно перегрузки (рис. 3.4) достаточно долго не может найти оптимальное значение, так как происходят частые потери из-за увеличения размера очереди на коммутаторе. Так как по умолчанию на рабочей машине стоит алгоритм для работы с перегрузками TCP Reno, то видно что график принимает пилообразные очертания.

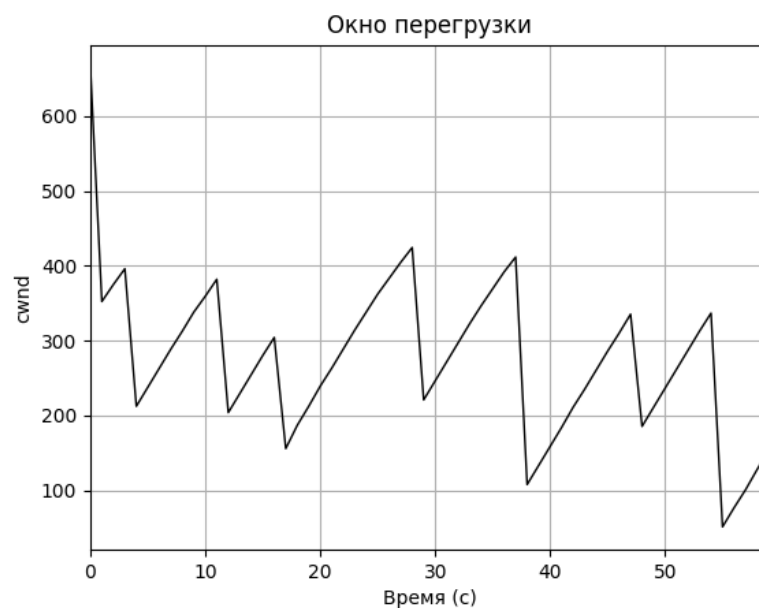


Рис. 3.4. График изменения значения окна перегрузки TCP Reno

На рис. 3.5 приведен график изменения значения количества повторно переданных данных с течением времени — появились потери пакетов на соединении, причем, достаточно частые.

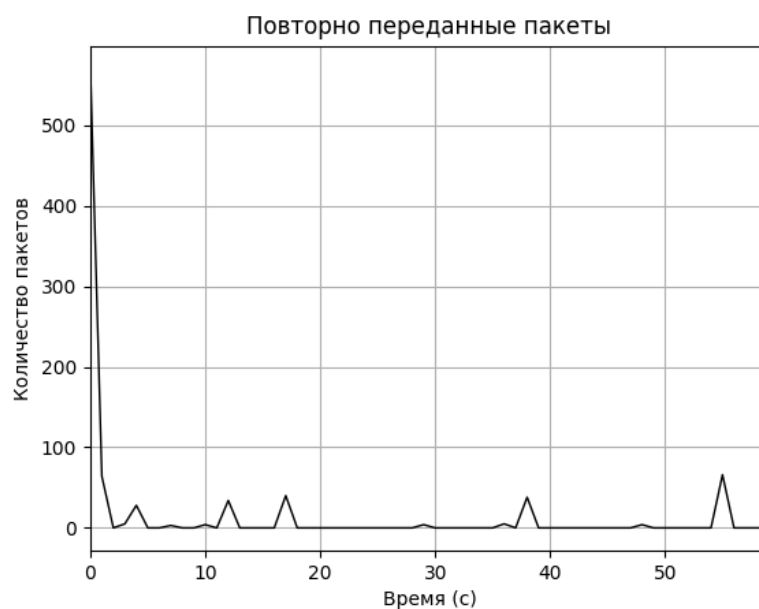


Рис. 3.5. График изменения количества повторно переданных данных

Среднее значение RTT (рис. 3.6) сильно увеличилось (0.04 мс против 60 мс), что связано с мгновенной отправкой данных без задержки в очереди.

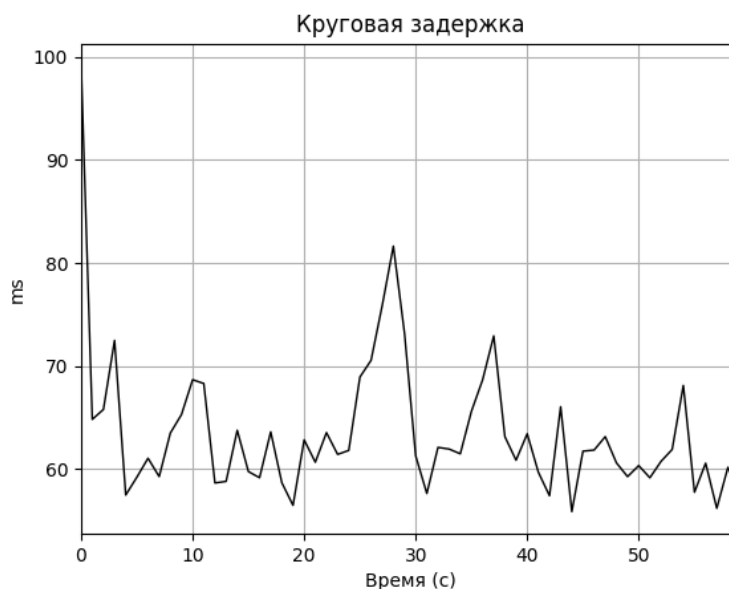


Рис. 3.6. График изменения значения RTT с течением времени

Пропускная способность (рис. 3.7) колеблется от 0 Мбит/с до 70 мбит/с из-за искусственного ограничения скорости передачи, потерь в сети и задержек распространения.

3.5. Измерение размера очереди на дисциплине

Некоторые сетевые характеристики мы уже успели рассмотреть в прошлой главе. Однако, иногда для анализа работы сети оказывается полезной еще одна характеристика, которую мы не можем измерить с помощью iPerf, так как данных по ней попросту нет ни на хосте, ни на сервере. Речь идет о длине очереди на сетевом устройстве.

Задача разработчиков сети — достигнуть оптимальной скорости передачи данных с минимально возможной задержкой. Одним из факторов, который

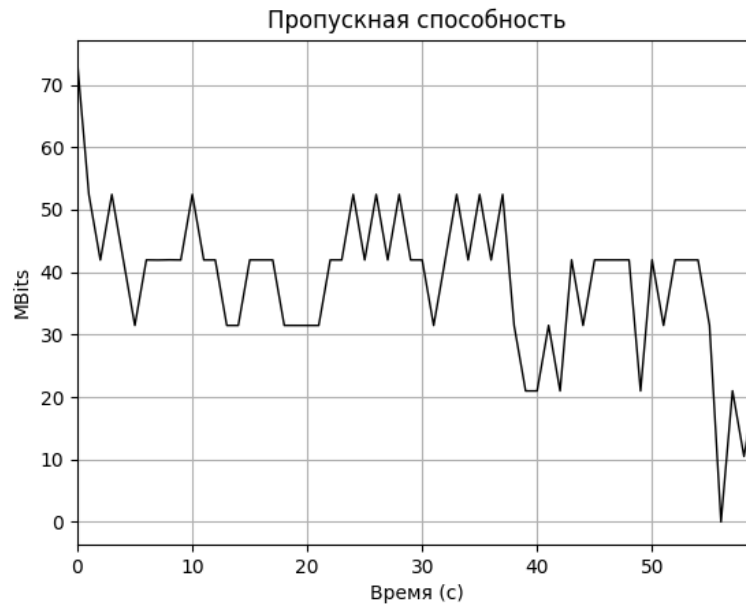


Рис. 3.7. График изменения значения пропускной способности

сказывается на это, является очередь на коммутаторах/маршрутизаторах. Буфер в коммутаторе (место, куда поступают пакеты и становятся в очередь на обслуживание) помогает не потерять пакеты, если имеет место высокая интенсивность поступления трафика. Если сеть работает на пределе, очередь коммутатора/маршрутизатора будет заполнена и пакеты начнут теряться. Чем длиннее очередь, тем больше задержка, что в свою очередь сильно сказывается на общей пропускной способности сети.

Реакцию очереди на изменение конфигураций передачи данных можно измерить и в зависимости от результатов подстроить другие параметры для достижения лучшей работы сети в целом. Например, изменив алгоритм работы с перегрузками на хостах, или дисциплину очереди, можно достичь лучшей производительности сети.

Для измерения длины очереди полезным окажется вывод статистики `tc`. Запись `backlog` указывает на размер очереди дисциплины в байтах и пакетах. Это статистику можно снять и измерять каждый промежуток времени с помощью функции на любом языке программирования. Функция записывает

время и размер очереди в файл, с помощью которого легко построить график, например, в gnuplot.

Напишем функцию, которая замеряет размер очереди с заданным интервалом времени. Диграмма активностостей для такой функции приведена на рис. 3.8. С помощью скрипта на языке программирования python (см. приложение В) построим график изменения длины очереди на интерфейсе 3.9.

Настройки сети аналогичны сети из прошлой главы (включая настройки qdisc).



Рис. 3.8. Диаграмма активностей для функции замеров длины очереди

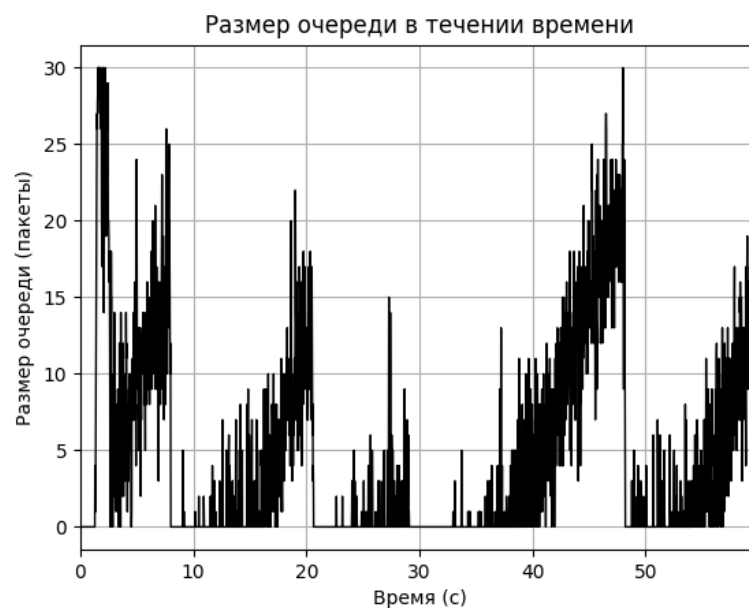


Рис. 3.9. График изменения длины очереди на интерфейсе s2-eth2

На графике видно, что в определенные моменты времени длина очереди или возрастает, или убывает. Связано это с алгоритмом TCP Reno. При высокой загрузке канала связи увеличивается длина очереди и, как следствие, RTT. В момент, когда RTT переходит за допустимую норму, источник уменьшает окно перегрузки, что влечет за собой уменьшение битрейта и длины очереди.

Заключение

В ходе курсовой работы мы научились моделировать простые виртуальные сети с помощью программы Mininet, рассмотрели технологии, которые помогают в сборе данных сетевых характеристик, оценили производительность сетей и рассмотрели способы визуализации полученных данных. Mininet предоставляет разработчикам быстрый и дешевый в построении испытательный полигон, в котором можно проектировать и отлаживать сетевые программы, что является большим плюсом для бизнес-сегмента.

Мониторинг сетевых характеристик передачи данных является неотъемлемой частью при проектировании сетей и сетевых программ. Умея анализировать данные сетевых характеристик, можно строить гибкие решения, которые позволяют получить максимальную производительность сетевого ресурса.

Список литературы

1. ip(8) — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/tc.8.html>.
2. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. — URL: <https://iperf.fr/iperf-doc.php>.
3. iproute2. — URL: <https://en.wikipedia.org/wiki/Iproute2>.
4. Mininet. — URL: <http://mininet.org/>.
5. ss(8) — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/ss.8.html>.
6. tc(8) — Linux manual page. — URL: <https://man7.org/linux/man-pages/man8/tc.8.html>.

А. Построение графиков сетевых характеристик из полученных данных iPerf3

Имея данные в формате json их обработка на языке программирования python сводится к тривиальной. Первым делом мы подключаем модуль json к программе, с помощью метода load загружаем данные переменную. На выходе у нас имеется готовый список данных, обработка которых представляет собой выбор по ключевому полю данных и их запись в нужную переменную. Код для функции обработки данных представлен ниже.

```
1  '''
2      Функция обработки данных iPerf.
3      Данные обрабатываются из предположения, что нас
4      интересуют только данные, перечисленные в
5      словаре y. Функция возвращает словарь [x,y].
6  '''
7  def parse_netstat_file(net_data_file):
8      raw_data = json.load(net_data_file)
9      x = []
10     y = {
11         "bytes": [],
12         {"title": "Количество переданных байт", "x": "Время (с)", "y": "МВ"}},
13         "cwnd": [],
14         {"title": "Окно перегрузки", "x": "Время (с)", "y": "cwnd"}},
15         "MTU": [],
16         {"title": "Максимальный размер пакета", "x": "Время (с)", "y": "В"}},
17         "retransmits": [],
18         {"title": "Повторно переданные пакеты", "x": "Время (с)", "y": "Количество
19             ↪ пакетов"}},
20         "rtt": [],
21         {"title": "Круговая задержка", "x": "Время (с)", "y": "ms"}},
22         "rttvar": [],
23         {"title": "Отклонение круговой задержки", "x": "Время (с)", "y": "ms"}},
24         "throughput": [],
25         {"title": "Пропускная способность", "x": "Время (с)", "y": "МBits"}}
26     '''
```

```

27         Цикл считывания данных из json-файла.
28         В список x сохраняются временные данные, а
29         в словарь y – все остальные. Характеристики для y
30         нормируются в зависимости от требуемой
31         шкалы измерения данных.
32     """
33     for i in raw_data["intervals"]:
34         tmp_data = i["streams"][0]
35         x.append(tmp_data["start"])
36         y["bytes"][0].append(tmp_data["bytes"] / 1024 / 1024)
37         y["cwnd"][0].append(tmp_data["snd_cwnd"] / 1024)
38         y["MTU"][0].append(tmp_data["pmtu"])
39         y["retransmits"][0].append(tmp_data["retransmits"])
40         y["rtt"][0].append(tmp_data["rtt"] / 1000)
41         y["rttvar"][0].append(tmp_data["rttvar"] / 1000)
42         y["throughput"][0].append(tmp_data["bits_per_second"] / 1000000)
43     return [x, y]

```

Получив обработанные данные, мы можем построить графики. Код для функции построения графиков представлен ниже.

```

1     """
2     Функция построения графиков сетевых характеристик.
3     На вход поступает список значений, где первый элемент –
4     время, а второй элемент – список значений сетевых
5     характеристик.
6     Графики сохраняются в заданном формате
7     (параметр plot_format) и в заданную
8     директорию (параметр folder)
9     """
10    def plot_net_stats(stats, plot_format, folder):
11        x_stats, y_stats = stats
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(folder, i, plot_format))
20            plt.clf()

```

Теперь мы можем соединить эти функции воедино в скриптовом файле, выбрать целевой файл, передать параметры для функций и построить графики сетевых характеристик.

В. Построение графика изменения длины очереди на сетевом интерфейсе

Файл данных для очереди представляет собой плоский файл, в котором имеется две колонки: время в секундах и размер очереди в пакетах, разделенные пробелом. Все, что нам нужно сделать - это отделить данные друг от друга, записать их в нужные структуры данных и позже данными для построения графика. Пример кода для функции обработки данных размера очереди представлен ниже.

```
1  '''
2      Функция обработки данных изменения
3      длины очереди. Разделяем данные из
4      предположения, что 1-й элемент -
5      значение времени (значение заносится в список x),
6      2-й элемент - длина очереди (значение
7      заносится в список y).
8      Функция возвращает список [x,y]
9  '''
10 def parse_queue_len(qlen_data_file):
11     x_stats = []
12     y_stats = []
13     for line in qlen_data_file:
14         line = line.split(" ")
15         x_stats.append(float(line[0]))
16         y_stats.append(float(line[1]))
17     return [x_stats, y_stats]
```

Имея список значений x и список значений y, построим график изменения длины очереди.

```
1  '''
2      Функция построения графика изменения длины очереди.
3      На вход поступает список значений, где первый элемент -
4      время, а второй элемент - список значений длины очереди.
```

```

5     Графики сохраняются в заданном формате
6     (параметр plot_format) и в заданную
7     директорию (параметр folder)
8     '''
9     def plot_queue_len(stats, plot_format, folder):
10         x_stats, y_stats = stats
11         plt.plot(x_stats, y_stats, 'k', linewidth=1)
12         plt.grid()
13         plt.xticks(np.arange(0.0, x_stats[-1], step=10.0))
14         plt.xlim(xmin=0, xmax=x_stats[-1])
15         plt.xlabel("Время (с)")
16         plt.ylabel("Размер очереди (пакеты)")
17         plt.title("Размер очереди в течении времени")
18         plt.savefig("{}queue_len.{}".format(folder, plot_format))

```

Теперь мы можем соединить эти функции воедино в скриптовом файле, выбрать целевой файл, передать параметры для функций и построить график изменения длины очереди на сетевом интерфейсе.