

Лабораторная работа № 3

**Измерение и тестирование пропускной способности сети.
Воспроизводимый эксперимент**

Беличева Дарья Михайловна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Копирование файла emptynet.py	7
4.2	Содержание файла lab_iperf3_topo.py	8
4.3	Создание топологии и ее основные параметры	9
4.4	Изменение скрипта lab_iperf3_topo.py	10
4.5	Проверка работы внесенных изменений	10
4.6	Настройка параметров производительности	12
4.7	Запуск скрипта с настройкой параметров производительности и без нее	13
4.8	Создание копии скрипта lab_iperf3_topo2.py	13
4.9	Изменения кода в скрипте lab_iperf3.py	14
4.10	Запуск скрипта lab_iperf3.py	15
4.11	Создание Makefile	15
4.12	Проверка работы Makefile	16

1 Цель работы

Основной целью работы является знакомство с инструментом для измерения пропускной способности сети в режиме реального времени — iPerf3, а также получение навыков проведения воспроизводимого эксперимента по измерению пропускной способности моделируемой сети в среде Mininet.

2 Задание

1. Воспроизвести посредством API Mininet эксперименты по измерению пропускной способности с помощью iPerf3.
2. Построить графики по проведённому эксперименту.

3 Теоретическое введение

Mininet[1] – это эмулятор компьютерной сети. Под компьютерной сетью подразумеваются простые компьютеры — хосты, коммутаторы, а так же OpenFlow-контроллеры. С помощью простейшего синтаксиса в примитивном интерпретаторе команд можно разворачивать сети из произвольного количества хостов, коммутаторов в различных топологиях и все это в рамках одной виртуальной машины(ВМ). На всех хостах можно изменять сетевую конфигурацию, пользоваться стандартными утилитами(`ifconfig`, `ping`) и даже получать доступ к терминалу. На коммутаторы можно добавлять различные правила и маршрутизировать трафик.

iPerf3[2] представляет собой кроссплатформенное клиент-серверное приложение с открытым исходным кодом, которое можно использовать для измерения пропускной способности между двумя конечными устройствами. iPerf3 может работать с транспортными протоколами TCP, UDP и SCTP:

- TCP и SCTP:
 - измеряет пропускную способность;
 - позволяет задать размер MSS/MTU;
 - отслеживает размер окна перегрузки TCP (CWnd).
- UDP:
 - измеряет пропускную способность;
 - измеряет потери пакетов;
 - измеряет колебания задержки (jitter);
 - поддерживает групповую рассылку пакетов (multicast).

4 Выполнение лабораторной работы

С помощью API Mininet создадим простейшую топологию сети, состоящую из двух хостов и коммутатора с назначенной по умолчанию mininet сетью 10.0.0.0/8.

В каталоге /work/lab_iperf3 для работы над проектом создадим подкаталог lab_iperf3_topo и скопируем в него файл с примером скрипта mininet/examples/emphynet.py, описывающего стандартную простую топологию сети mininet (рис. 4.1).

```
mininet@mininet-vm:~$ cd ~/work/lab_iperf3
mininet@mininet-vm:~/work/lab_iperf3$ mkdir lab_iperf3_topo
mininet@mininet-vm:~/work/lab_iperf3$ 
mininet@mininet-vm:~/work/lab_iperf3$ cd ~/work/lab_iperf3/lab_iperf3_topo
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ cp ~/mininet/examples/emphynet.py ~/work/lab_iperf3/lab_iperf3_topo
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ mv emphynet.py lab_iperf3_topo.py
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$
```

Рис. 4.1: Копирование файла emphynet.py

Изучим содержание скрипта lab_iperf3_topo.py (рис. 4.2).

В нем написан скрипт по созданию простейшей топологии из двух хостов h1 и h2, а также коммутатора s3 и контроллера c0. В начале файла видим импорт необходимых библиотек.

```
GNU nano 4.8 lab_iperf3_topo.py
#!/usr/bin/env python

"""
This example shows how to create an empty Mininet object
(without a topology object) and add nodes to it manually.
"""

from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def emptyNet():

    "Create an empty network and add nodes to it."

    net = Mininet( controller=Controller, waitConnected=True )

    info( '*** Adding controller\n' )
    net.addController( 'c0' )

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1', ip='10.0.0.1' )
    h2 = net.addHost( 'h2', ip='10.0.0.2' )

    info( '*** Adding switch\n' )
    s3 = net.addSwitch( 's3' )

    info( '*** Creating links\n' )
    net.addLink( h1, s3 )
    net.addLink( h2, s3 )
```

Рис. 4.2: Содержание файла lab_iperf3_topo.py

Основные элементы:

- addSwitch(): добавляет коммутатор в топологию и возвращает имя коммутатора;
- addHost(): добавляет хост в топологию и возвращает имя хоста;
- addLink(): добавляет двунаправленную ссылку в топологию (и возвращает ключ ссылки; ссылки в Mininet являются двунаправленными, если не указано иное);
- Mininet: основной класс для создания и управления сетью;
- start(): запускает сеть;
- pingAll(): проверяет подключение, пытаясь заставить все узлы пинговать друг друга;
- stop(): останавливает сеть;

- `net.hosts`: все хосты в сети;
- `dumpNodeConnections()`: сбрасывает подключения к/от набора узлов;
- `setLogLevel('info' | 'debug' | 'output')`: устанавливает уровень вывода Mininet по умолчанию; рекомендуется `info`.

Запустим скрипт создания топологии `lab_iperf3_topo.py` и посмотрим ее основные параметры (рис. 4.3).

```
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ sudo python lab_ip
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s3 ...
*** Waiting for switches to connect
s3
*** Running CLI
*** Starting CLI:
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
s3 lo: s3-eth1:h1-eth0 s3-eth2:h2-eth0
c0
mininet> links
h1-eth0<->s3-eth1 (OK OK)
h2-eth0<->s3-eth2 (OK OK)
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=856>
<Host h2: h2-eth0:10.0.0.2 pid=859>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=864>
<Controller c0: 127.0.0.1:6653 pid=849>
```

Рис. 4.3: Создание топологии и ее основные параметры

Внесем в скрипт `lab_iperf3_topo.py` изменение, позволяющее вывести на экран информацию обоих хостов сети, а именно имя хоста, его IP-адрес, MAC-адрес (рис. 4.4).

```
def emptyNet():
    "Create an empty network and add nodes to it."

    net = Mininet( controller=Controller, waitConnected=True )

    info( '*** Adding controller\n' )
    net.addController( 'c0' )

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1', ip='10.0.0.1' )
    h2 = net.addHost( 'h2', ip='10.0.0.2' )

    info( '*** Adding switch\n' )
    s3 = net.addSwitch( 's3' )

    info( '*** Creating links\n' )
    net.addLink( h1, s3 )
    net.addLink( h2, s3 )

    info( '*** Starting network\n' )
    net.start()
    print( "Host", h1.name, "has IP address", h1.IP(), "and MAC address", h1.MAC() )
    print( "Host", h2.name, "has IP address", h2.IP(), "and MAC address", h2.MAC() )

    info( '*** Running CLI\n' )
    CLI( net )

    info( '*** Stopping network\n' )
    net.stop()
```

Рис. 4.4: Изменение скрипта lab_iperf3_topo.py

Здесь:

- IP() возвращает IP-адрес хоста или определенного интерфейса;
- MAC() возвращает MAC-адрес хоста или определенного интерфейса.

Проверим корректность отработки изменённого скрипта (рис. 4.5).

```
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ nano lab_iperf3_topo.py
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ sudo python lab_iperf3_topo.py
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s3 ...
*** Waiting for switches to connect
s3
Host h1 has IP address 10.0.0.1 and MAC address 06:75:a4:da:79:ec
Host h2 has IP address 10.0.0.2 and MAC address ca:39:81:33:44:7a
*** Running CLI
*** Starting CLI:
```

Рис. 4.5: Проверка работы внесенных изменений

Действительно, нам вывелась информация об IP и mac адресах хостов.

Mininet предоставляет функции ограничения производительности и изоляции с помощью классов CPULimitedHost и TCLink. Добавим в скрипт настройки параметров производительности (рис. 4.6).

В скрипте `lab_iperf3_topo2.py` изменим строку описания сети, указав на использование ограничения производительности и изоляции. Также изменим функцию задания параметров виртуального хоста `h1`, указав, что ему будет выделено 50% от общих ресурсов процессора системы. Аналогичным образом для хоста `h2` зададим долю выделения ресурсов процессора в 45%. В скрипте изменим функцию параметров соединения между хостом `h1` и коммутатором `s3`. А именно добавим двунаправленный канал с характеристиками пропускной способности, задержки и потерь:

- параметр пропускной способности (`bw`) выражается числом в Мбит;
- задержка (`delay`) выражается в виде строки с заданными единицами измерения (например, `5ms`, `100us`, `1s`);
- потери (`loss`) выражаются в процентах (от 0 до 100);
- параметр максимального значения очереди (`max_queue_size`) выражается в пакетах;
- параметр `use_htb` указывает на использование ограничителя интенсивности входящего потока Hierarchical Token Bucket (HTB)

```

GNU nano 4.8                                lab iperf3_topo2.py                                Modified
#!/usr/bin/env python

"""
This example shows how to create an empty Mininet object
(without a topology object) and add nodes to it manually.
"""

from mininet.net import Mininet
from mininet.node import Controller
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def emptyNet():

    "Create an empty network and add nodes to it."

    net = Mininet( controller=Controller, waitConnected=True, host = CPULimitedHost, link = TCLink

    info( '*** Adding controller\n' )
    net.addController( 'c0' )

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1', ip='10.0.0.1', cpu=50 )
    h2 = net.addHost( 'h2', ip='10.0.0.2', cpu=45 )

    info( '*** Adding switch\n' )
    s3 = net.addSwitch( 's3' )

    info( '*** Creating links\n' )
    net.addLink( h1, s3, bw=10, delay='5ms', max_queue_size=1000, loss=10, use_htb=True )
    net.addLink( h2, s3 )

    info( '*** Starting network\n' )
    net.start()
    print( "Host", h1.name, "has IP address", h1.IP(), "and MAC address", h1.MAC() )
    print( "Host", h2.name, "has IP address", h2.IP(), "and MAC address", h2.MAC() )

    info( '*** Running CLI\n' )
    CLI( net )

    info( '*** Stopping network' )
    net.stop()

```

Рис. 4.6: Настройка параметров производительности

Запустим на отработку сначала скрипт lab_iperf3_topo2.py, затем lab_iperf3_topo.py и сравним результат (рис. 4.7). Увидим, что в первом случае у нас создалась сеть с настроенными параметрами, а во втором случае дефолтная сеть без этих параметров.

```

mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ sudo python lab_iperf3_topo2.py
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
(10.00Mbit 5ms delay 10.00000% loss) (10.00Mbit 5ms delay 10.00000% loss) *** Starting network
*** Configuring hosts
h1 (cfs 5000000/1000000us) h2 (cfs 4500000/1000000us)
*** Starting controller
c0
*** Starting 1 switches
s3 (10.00Mbit 5ms delay 10.00000% loss) ... (10.00Mbit 5ms delay 10.00000% loss)
*** Waiting for switches to connect
s3
Host h1 has IP address 10.0.0.1 and MAC address 36:6b:1b:32:54:c0
Host h2 has IP address 10.0.0.2 and MAC address b6:93:11:e6:e9:8a
*** Running CLI
*** Starting CLI:
mininet> exit
*** Stopping network*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s3
*** Stopping 2 hosts
h1 h2
*** Done
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ sudo python lab_iperf3_topo.py
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s3 ...
*** Waiting for switches to connect
s3
Host h1 has IP address 10.0.0.1 and MAC address 26:9c:7d:3f:02:1f
Host h2 has IP address 10.0.0.2 and MAC address 72:65:85:a7:2f:70
*** Running CLI
*** Starting CLI:

```

Рис. 4.7: Запуск скрипта с настройкой параметров производительности и без нее

Построим графики по проводимому эксперименту.

Сделаем копию скрипта lab_iperf3_topo2.py и поместим его в подкаталог iperf (рис. 4.8).

```

mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ cp lab_iperf3_topo2.py lab_iperf3.py
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ mkdir -p ~/work/lab_iperf3/iperf3
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ mv ~/work/lab_iperf3/lab_iperf3_topo/lab_i
per3.py ~/work/lab_iperf3/iperf3
mininet@mininet-vm:~/work/lab_iperf3/lab_iperf3_topo$ cd ~/work/lab_iperf3/iperf3
mininet@mininet-vm:~/work/lab_iperf3/iperf3$ ls
lab_iperf3.py
mininet@mininet-vm:~/work/lab_iperf3/iperf3$ ls -l
total 4
-rwxrwxr-x 1 mininet mininet 1345 Nov 25 13:12 lab_iperf3.py
mininet@mininet-vm:~/work/lab_iperf3/iperf3$

```

Рис. 4.8: Создание копии скрипта lab_iperf3_topo2.py

Изменим код в скрипте lab_iperf3.py так, чтобы (рис. 4.9).:

- на хостах не было ограничения по использованию ресурсов процессора;
- каналы между хостами и коммутатором были по 100 Мбит/с с задержкой 75 мс, без потерь, без использования ограничителей пропускной способности

и максимального размера очереди.

- После функции старта сети опишем запуск на хосте h2 сервера iPerf3, а на хосте h1 запуск с задержкой в 10 секунд клиента iPerf3 с экспортом результатов в JSON-файл, закомментируем строки, отвечающие за запуск CLI-интерфейса:

```
"""
This example shows how to create an empty Mininet object
(without a topology object) and add nodes to it manually.
"""
import time
from mininet.net import Mininet
from mininet.node import Controller
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def emptyNet():
    "Create an empty network and add nodes to it."

    net = Mininet( controller=Controller, waitConnected=True, host = CPULimitedHost, link =
    info( '*** Adding controller\n' )
    net.addController( 'c0' )

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1', ip='10.0.0.1' )
    h2 = net.addHost( 'h2', ip='10.0.0.2' )

    info( '*** Adding switch\n' )
    s3 = net.addSwitch( 's3' )

    info( '*** Creating links\n' )
    net.addLink( h1, s3, bw=100, delay='75ms' )
    net.addLink( h2, s3, bw=100, delay='75ms' )

    info( '*** Starting network\n' )
    net.start()
    info( '*** Starting network\n' )

    info( '*** Traffic generation\n' )
    h2.cmdPrint( 'iperf3 -s -D -1' )
    time.sleep(10) # Wait 10 seconds for servers to start
    h1.cmdPrint( 'iperf3 -c', h2.IP(), '-J > iperf_result.json' )

    # info( '*** Running CLI\n' )
    # CLI( net )
```

Рис. 4.9: Изменен ия кода в скрипте lab_iperf3.py

Запустим на отработку скрипт lab_iperf3.py (рис. 4.10).

```

mininet@mininet-vm:~/work/lab_iperf3/iperf3$ sudo python lab_iperf3.py
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
(100.00Mbit 75ms delay) (100.00Mbit 75ms delay) (100.00Mbit 75ms delay) (100.00Mbit 75ms del
*** Starting network
*** Configuring hosts
h1 (cfs -l/1000000us) h2 (cfs -l/1000000us)
*** Starting controller
c0
*** Starting 1 switches
s3 (100.00Mbit 75ms delay) (100.00Mbit 75ms delay) ... (100.00Mbit 75ms delay) (100.00Mbit 75
delay)
*** Waiting for switches to connect
s3
*** Starting network
*** Traffic generation
*** h2 : ('iperf3 -s -D -l',)
*** h1 : ('iperf3 -c', '10.0.0.2', '-J > iperf_result.json')
*** Stopping network*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s3
*** Stopping 2 hosts
h1 h2
*** Done
mininet@mininet-vm:~/work/lab_iperf3/iperf3$ ls
iperf_result.json lab_iperf3.py

```

Рис. 4.10: Запуск скрипта lab_iperf3.py

Построим графики из получившегося JSON-файла. Создадим Makefile для проведения всего эксперимента. В Makefile пропишем запуск скрипта эксперимента, построение графиков и очистку каталога от результатов (рис. 4.11).

```

GNU nano 4.8 Makefile
all: iperf_result.json plot

iperf_result.json:
    sudo python lab_iperf3.py

plot: iperf_result.json
    plot_iperf.sh iperf_result.json

clean:
    -rm -f *.json *.csv
    rm -rf results

```

Рис. 4.11: Создание Makefile

Проверьте корректность отработки Makefile (рис. 4.12).

```

mininet@mininet-vm:~/work/lab_iperf3/iperf3$ nano Makefile
mininet@mininet-vm:~/work/lab_iperf3/iperf3$ make clean
rm -f *.json *.csv
rm -rf results
mininet@mininet-vm:~/work/lab_iperf3/iperf3$ make
sudo python lab_iperf3.py
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
(100.00Mbit 75ms delay) (100.00Mbit 75ms delay) (100.00Mbit 75ms delay)
*** Starting network
*** Configuring hosts
h1 (cfs -l/1000000us) h2 (cfs -l/1000000us)
*** Starting controller
c0
*** Starting 1 switches
s3 (100.00Mbit 75ms delay) (100.00Mbit 75ms delay) ... (100.00Mbit 75ms
delay)
*** Waiting for switches to connect
s3
*** Starting network
*** Traffic generation
*** h2 : ('iperf3 -s -D -l',)
*** h1 : ('iperf3 -c', '10.0.0.2', '-J > iperf_result.json')
*** Stopping network*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s3
*** Stopping 2 hosts
h1 h2
*** Done
plot_iperf.sh iperf_result.json
mininet@mininet-vm:~/work/lab_iperf3/iperf3$ █

```

Рис. 4.12: Проверка работы Makefile

5 Выводы

В результате выполнения данной лабораторной работы я познакомилась с инструментом для измерения пропускной способности сети в режиме реального времени — iPerf3, а также получила навыки проведения воспроизводимого эксперимента по измерению пропускной способности моделируемой сети в среде Mininet.

Список литературы

1. Mininet [Электронный ресурс]. Mininet Project Contributors. URL: <http://mininet.org/> (дата обращения: 17.11.2024).
2. IPerff [Электронный ресурс]. URL: <https://iperf.fr/>.