Лабораторная работа № 3. Измерение и тестирование пропускной способности сети. Воспроизводимый эксперимент

3.1. 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.

3.2. Предварительные сведения

3.2.1. API Mininet

Application Programming Interface (API) — программный интерфейс приложения, или интерфейс программирования приложений) представляет собой специальный протокол для взаимодействия компьютерных программ, который позволяет использовать функции одного приложения внутри другого.

API Mininet построен на трех основных уровнях:

- *Низкоуровневый API* состоит из базовых узлов и классов ссылок (таких как `Host`, `Switch`, `Link` и их подклассы), которые на самом деле могут быть созданы по отдельности и использоваться для создания сети, но это немного громоздко.
- *API среднего уровня* добавляет объект Mininet, который служит контейнером для узлов и ссылок. Он предоставляет ряд методов (`addHost()`, `addSwitch()`, `addLink()`) для добавления узлов и ссылок в сеть, а также настройки сети, запуска и завершения работы (`start()`, `stop()`).
- *Высокоуровневый API* добавляет абстракцию шаблона топологии (класс `Topo`), который предоставляет возможность создавать повторно используемые параметризованные шаблоны топологии. Эти шаблоны можно передать команде `mn` (через параметр `--custom`) и использовать из командной строки.

Низкоуровневый API используется, когда требуется управлять узлами и коммутаторами напрямую. API среднего уровня применяют при запуске и остановке сети (в частности используется класс `Mininet`).

Полноценные сети могут быть созданы с использованием любого из уровней API, но обычно для создания сетей выбирают либо API среднего уровня (например, `Mininet.add*()`), либо API высокого уровня (`Topo. add*()`).

3.2.2. Примеры API Mininet

Пример сценария низкоуровневого API с описанием узлов и связывающих их сетевых соединений:

```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
c0.start()
s1.start( [ c0 ] )
print( h1.cmd( 'ping -c1', h2.IP() ) )
s1.stop()
c0.stop()
```

Пример сценария API среднего уровня с описанием сетевого объекта:

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
net.start()
print( h1.cmd( 'ping -c1', h2.IP() ) )
CLI( net )
net.stop()
```

Пример API высокого уровня с шаблоном топологии:

```
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def build( self, count=1 ):
        hosts = [ self.addHost( 'h%d' % i )
                  for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )

net = Mininet( topo=SingleSwitchTopo( 3 ) )
net.start()
CLI( net )
net.stop()
```

API среднего уровня на самом деле является самым простым и лаконичным для этого примера, поскольку не требует создания класса топологии. API низкого и среднего уровня являются гибкими и мощными, но могут быть менее удобными для повторного использования по сравнению с высокоуровневым API Topo и шаблонами топологии.

3.3. Задание

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

3.4. Последовательность выполнения работы

1. С помощью API Mininet создайте 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 создайте подкаталог `lab_iperf3_topo` и скопируйте в него файл с примером скрипта `mininet/examples/emptynet.py`, описывающего стандартную простую топологию сети mininet:

```
cd ~/work/lab_iperf3
mkdir lab_iperf3_topo
cd ~/work/lab_iperf3/lab_iperf3_topo
cp ~/mininet/examples/emptynet.py
  ↪  ~/work/lab_iperf3/lab_iperf3_topo
mv emptynet.py lab_iperf3_topo.py
```

   - Изучите содержание скрипта `lab_iperf3_topo.py`:

```
#!/usr/bin/env python

"""
Simple topology
"""

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."
```

```
15
16      net = Mininet( controller=Controller,
            ↪  waitConnected=True )
17
18      info( '*** Adding controller\n' )
19      net.addController( 'c0' )
20
21      info( '*** Adding hosts\n' )
22      h1 = net.addHost( 'h1', ip='10.0.0.1' )
23      h2 = net.addHost( 'h2', ip='10.0.0.2' )
24
25      info( '*** Adding switch\n' )
26      s3 = net.addSwitch( 's3' )
27
28      info( '*** Creating links\n' )
29      net.addLink( h1, s3 )
30      net.addLink( h2, s3 )
31
32      info( '*** Starting network\n')
33      net.start()
34
35      info( '*** Running CLI\n' )
36      CLI( net )
37
38      info( '*** Stopping network' )
39      net.stop()
40
41  if __name__ == '__main__':
42      setLogLevel( 'info' )
43      emptyNet()
44
```

Основные элементы:

- `addSwitch()`: добавляет коммутатор в топологию и возвращает имя коммутатора;
- `ddHost()`: 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`.

– Запустите скрипт создания топологии `lab_iperf3_topo.py`:

```
sudo python lab_iperf3_topo.py
```

– После отработки скрипта посмотрите элементы топологии и завершите работу mininet:

```
mininet> net
mininet> links
mininet> dump
mininet> exit
```

2. Внесите в скрипт `lab_iperf3_topo.py` изменение, позволяющее вывести на экран информацию о хосте `h1`, а именно имя хоста, его IP-адрес, MAC-адрес. Для этого после строки, задающей старт работы сети, добавьте строку:

```
print( "Host", h1.name, "has IP address", h1.IP(), "and
 ↪  MAC address", h1.MAC() )
```

Здесь:
– `IP()` возвращает IP-адрес хоста или определенного интерфейса;
– `MAC()` возвращает MAC-адрес хоста или определенного интерфейса.

3. Проверьте корректность отработки изменённого скрипта.
4. Измените скрипт `lab_iperf3_topo.py` так, чтобы на экран выводилась информация об имени, IP-адресе и MAC-адресе обоих хостов сети. Проверьте корректность отработки изменённого скрипта.
5. Mininet предоставляет функции ограничения производительности и изоляции с помощью классов `CPULimitedHost` и `TCLink`. Добавьте в скрипт настройки параметров производительности:
   – Сделайте копию скрипта `lab_iperf3_topo.py`:

```
cp lab_iperf3_topo.py lab_iperf3_topo2.py
```

   – В начале скрипта `lab_iperf3_topo2.py` добавьте записи об импорте классов `CPULimitedHost` и `TCLink`:

```
...
from mininet.node import CPULimitedHost
from mininet.link import TCLink
...
```

   – В скрипте `lab_iperf3_topo2.py` измените строку описания сети, указав на использование ограничения производительности и изоляции:

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

- В скрипте `lab_iperf3_topo2.py` измените функцию задания параметров виртуального хоста `h1`, указав, что ему будет выделено 50% от общих ресурсов процессора системы:

```
...
h1 = net.addHost( 'h1', ip='10.0.0.1', cpu=50 )
...
```

- Аналогичным образом для хоста `h2` задайте долю выделения ресурсов процессора в 45%.
- В скрипте `lab_iperf3_topo2.py` измените функцию параметров соединения между хостом `h1` и коммутатором `s3`:

```
...
net.addLink( h1, s3, bw=10, delay='5ms',
    max_queue_size=1000, loss=10, use_htb=True )
...
```

Здесь добавляется 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)*.
- Запустите на отработку сначала скрипт `lab_iperf3_topo2.py`, затем `lab_iperf3_topo.py` и сравните результат.

6. Постройте графики по проводимому эксперименту:
   - Сделайте копию скрипта `lab_iperf3_topo2.py` и поместите его в подкаталог `iperf`:

```
cp lab_iperf3_topo2.py lab_iperf3.py
mkdir -p ~/work/lab_iperf3/iperf3
mv ~/work/lab_iperf3/lab_iperf3_topo/lab_iperf3.py
    ~/work/lab_iperf3/iperf3
cd ~/work/lab_iperf3/iperf3
ls -l
```

– В начале скрипта `lab_iperf3.py` добавьте запись

```
...
import time
...
```

– Измените код в скрипте `lab_iperf3.py` так, чтобы:
  – 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 опишите запуск на хосте `h2` сервера `iPerf3`, а на хосте `h1` запуск с задержкой в 10 секунд клиента `iPerf3` с экспортом результатов в JSON-файл, закомментируйте строки, отвечающие за запуск CLI-интерфейса:

```
...
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 )
...
```

– В отчёте поясните синтаксис вызова iPerf3, заданный в скрипте.
– Запустите на отработку скрипт `lab_iperf3.py`:

```
sudo python lab_iperf3.py
```

– Постройте графики из получившегося JSON-файла:

```
plot_iperf.sh iperf_result.json
```

– Создайте `Makefile` для проведения всего эксперимента:

```
touch Makefile
```

– В `Makefile` пропишите запуск скрипта эксперимента, построение графиков и очистку каталога от результатов:

```
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
```

- Проверьте корректность отработки `Makefile`:

```
make clean
make
```

7. Завершите соединение с виртуальной машиной mininet и выключите её.

3.5. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание результатов выполнения задания:
   - скриншоты (снимки экрана), фиксирующие выполнение работы;
   - подробное описание настроек служб в соответствии с заданием;
   - результаты проверки корректности настроек служб в соответствии с заданием (подтверждённые скриншотами);
   - листинги (исходный код) программ (если они есть);
   - результаты выполнения программ (текст, графики или снимки экрана в зависимости от задания).
4. Выводы, согласованные с заданием работы.