

Санкт-Петербургский национальный исследовательский университет  
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

## **ПРАКТИЧЕСКАЯ РАБОТА**

Выполнил:

Швалов Даниил Андреевич

Группа: К33211

Проверил:

Меркушев Александр Евгеньевич

Санкт-Петербург

2023

## Упражнение 1

В данном упражнении необходимо установить и настроить среду для работы с mininet. В качестве рабочей машины я использую MacBook с процессором M1. В связи с этим у меня возникли проблемы с установкой и использованием виртуальной машины. Поэтому для работы с mininet была использована выделенная виртуальная машина RUVDS (рис. 1)

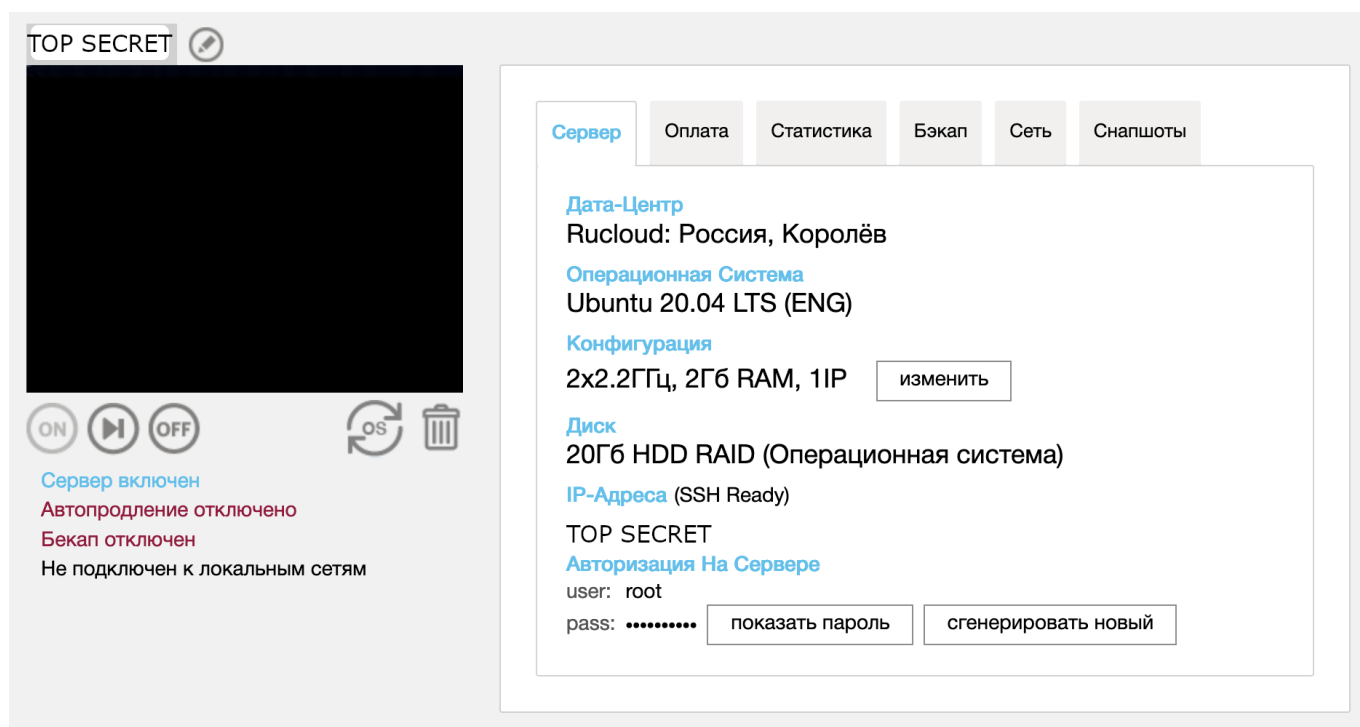


Рисунок 1 – Виртуальная машина RUVDS

Вследствие этого, mininet был установлен в систему нативно. Для этого я проделал следующие шаги:

1. клонировал исходный код mininet с помощью команды  

```
git clone https://github.com/mininet/mininet
```
2. выбрал последнюю стабильную версию mininet с помощью команд  

```
cd mininet  
git tag # list available versions  
git checkout -b mininet-2.3.0 2.3.0
```

```
cd ..
```

3. установил mininet в систему с помощью команды

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

После этого mininet был успешно установлен на мою виртуальную машину (рис. 2).

```
root@ruvds-w39ps:~# sudo mn
sudo: unable to resolve host ruvds-w39ps: Name or service not known
*** 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> |
```

Рисунок 2 – Проверка работоспособности mininet

## Упражнение 2

В этом упражнении необходимо создать собственную топологию с использованием API-интерфейса mininet на языке python. В начале необходимо было создать линейную топологию, изображенную на рис. 3.

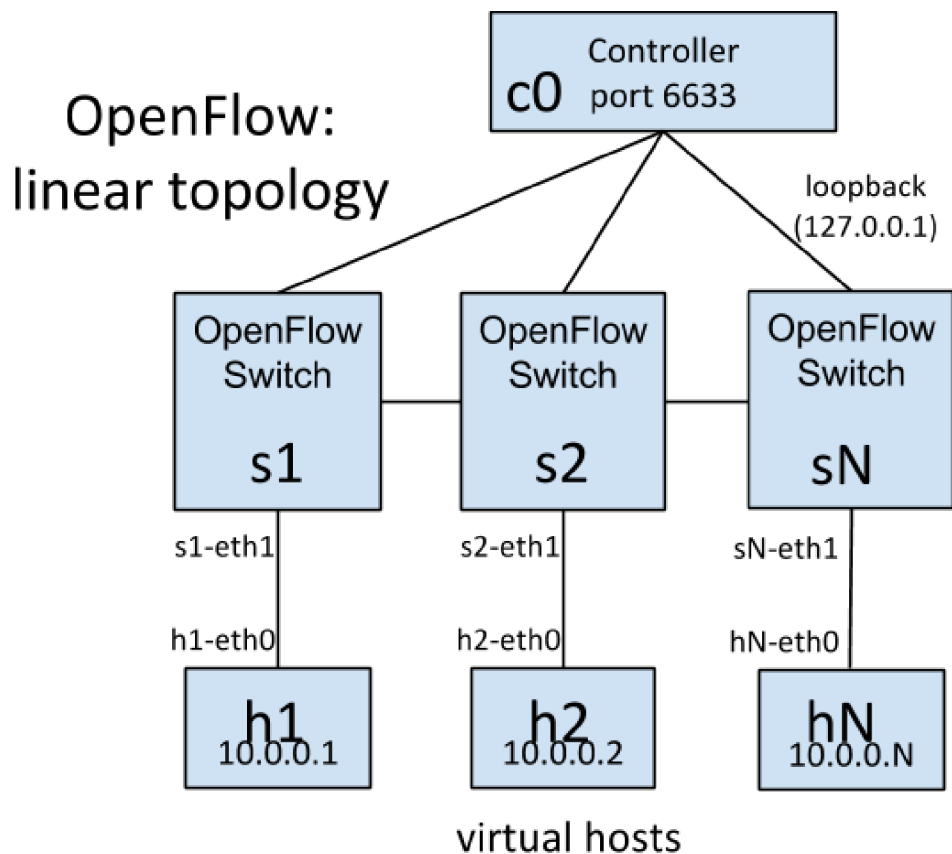


Рисунок 3 – Схема линейной топологии

Исходный код скрипта, реализующего линейную топологию, находится в приложении А. На рис. 4 изображен результат запуска скрипта.

```

root@ruvds-w39ps:~/networks# sudo ./LinearTopo.py
sudo: unable to resolve host ruvds-w39ps: Name or service not known
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (h1, s1) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (h2, s2) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (h3, s3) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (h4, s4) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (s2, s1) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (s3, s2) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (s4, s3)
*** Configuring hosts
h1 (cfs 25000/100000us) h2 (cfs 25000/100000us) h3 (cfs 25000/100000us) h4 (cfs 25000/100000us)
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ... (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 X h3
*** Results: 8% dropped (11/12 received)
*** Stopping 1 controllers
c0
*** Stopping 7 links
.....
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
  
```

Рисунок 4 – Результат запуска скрипта линейной топологии

изображенную на рис. 5.

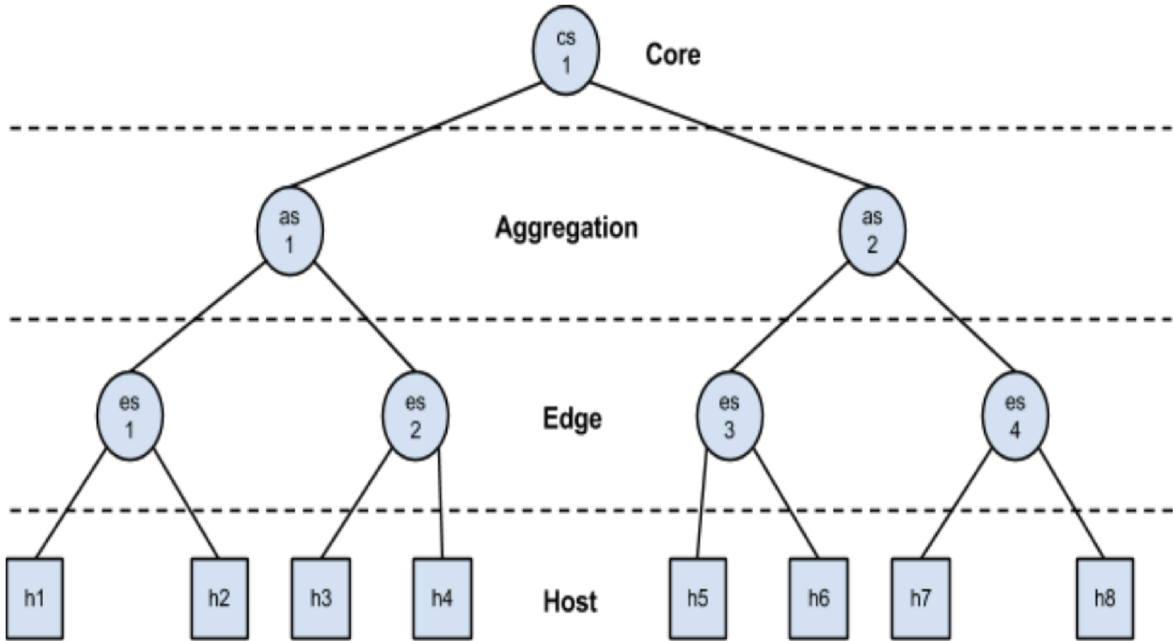


Рисунок 5 – Схема древовидной топологии с разветвлением равным 2

Исходный код скрипта, реализующего древовидную топологию, находится в приложении Б. На рис. 6 изображен результат запуска скрипта.

```
root@rvuds-w39ps:/networks# sudo ./CustomTopo.py
sudo: unable to resolve host rvuds-w39ps: Name or service not known
*** Creating network
*** Adding controller
*** Adding hosts:
hs1 hs2 hs3 hs4 hs5 hs6 hs7 hs8
*** Adding switches:
as1 as2 cs1 es1 es2 es3 es4
*** Adding Links:
(10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (as1, cs1) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (as2, cs1) (10.00Mbit 5ms delay 1.00000% loss) (1
0.00Mbit 5ms delay 1.00000% loss) (es1, as1) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (es2, as1) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (es3
, as2) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (es4, as2) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (hs1, es1) (10.00Mbit 5ms delay 1.00000% l
oss) (10.00Mbit 5ms delay 1.00000% loss) (hs2, es1) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (hs3, es2) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (
hs4, es2) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (hs5, es3) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (hs6, es3) (10.00Mbit 5ms delay 1.0
0000% loss) (10.00Mbit 5ms delay 1.00000% loss) (hs7, es4) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (hs8, es4)
*** Configuring hosts
hs1 (cfs -1/100000us) hs2 (cfs -1/100000us) hs3 (cfs -1/100000us) hs4 (cfs -1/100000us) hs5 (cfs -1/100000us) hs6 (cfs -1/100000us) hs7 (cfs -1/100000us) hs8 (cfs -1/100000us)
*** Starting controller
c0
*** Starting 7 switches
as1 as2 cs1 es1 es2 es3 es4 ... (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000%
loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms de
lay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss) (10.00Mbit 5ms delay 1.00000% loss)
Dumping host connections
hs1 hs1-eth0:es1-eth2
hs2 hs2-eth0:es1-eth3
hs3 hs3-eth0:es2-eth2
hs4 hs4-eth0:es2-eth3
hs5 hs5-eth0:es3-eth2
hs6 hs6-eth0:es3-eth3
hs7 hs7-eth0:es4-eth2
hs8 hs8-eth0:es4-eth3
Testing network connectivity
*** Ping: testing ping reachability
hs1 -> hs2 hs3 hs4 X hs6 hs7 hs8
hs2 -> hs1 hs3 hs4 hs5 hs6 hs7 hs8
hs3 -> hs1 hs2 hs4 hs5 hs6 hs7 hs8
hs4 -> hs1 hs2 hs3 hs5 hs6 hs7 hs8
hs5 -> hs1 hs2 hs3 hs4 hs6 hs7 hs8
hs6 -> hs1 hs2 X hs4 hs5 hs7 X
hs7 -> hs1 hs2 hs3 hs4 hs5 hs6 hs8
hs8 -> hs1 hs2 X hs4 hs5 hs6 hs7
*** Results: 7% dropped (52/56 received)
```

Рисунок 6 – Результат запуска скрипта древовидной топологии

## Упражнение 3

В данном упражнении необходимо использовать POX и протестировать доступность хостов. В начале был выключен предыдущий контроллер и очищен mininet (рис. 7).

```
root@ruvds-w39ps:~# ps -A | grep controller
root@ruvds-w39ps:~# sudo mn -c
sudo: unable to resolve host ruvds-w39ps: Name or service not known
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller ovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller ovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br as1 -- --if-exists del-br as2 -- --if-exists del-br cs1 -- --if-exists del-br es1 -- --if-exists del-br es2 -- --if-exists del-br es3 -- --if-exists del-br es4
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_[:alnum:]]+-eth[[:digit:]]+)'
( ip link del as2-eth1;ip link del cs1-eth2;ip link del as1-eth2;ip link del es1-eth1;ip link del es1-eth1;ip link del as1-eth2;ip link del as1-eth3;ip link del es2-eth1;ip link del es2-eth1;ip link del as1-eth3;ip link del as2-eth2;ip link del es3-eth1;ip link del es3-eth1;ip link del as2-eth2;ip link del as2-eth3;ip link del es4-eth1;ip link del es4-eth1;ip link del as2-eth3;ip link del cs1-eth1;ip link del as1-eth1;ip link del as1-eth1;ip link del cs1-eth1;ip link del cs1-eth2;ip link del as2-eth1 ) 2> /dev/null
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
```

Рисунок 7 – Отключение предыдущего контроллера и очистка mininet

После этого был запущен POX (рис. 8) и mininet (рис. 9).

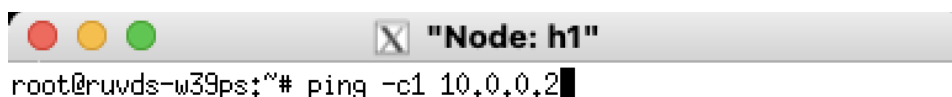
```
root@ruvds-w39ps:~# pox/pox.py log.level --DEBUG forwarding.hub
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:forwarding.hub:Proactive hub running.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.8.10/May 26 2023 14:05:08)
DEBUG:core:Platform is Linux-5.4.0-97-generic-x86_64-with-glibc2.29
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

Рисунок 8 – Запуск POX

```
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> |
```

Рисунок 9 – Запуск mininet

Затем в терминале mininet была выполнена команда `xterm h1 h2 h3`, которая создала три терминала. Во втором терминале была выполнена команда `tcpdump -XX -n -i h2-eth0`, а в третьем — команда `tcpdump -XX -n -i h3-eth0`. Тем самым был запущен `tcpdump` для второго и третьего хостов для печати пакетов, просматриваемых хостом. В первом терминале была выполнена команда `ping -c1 10.0.0.2`, которая проверяет доступность хоста с адресом 10.0.0.2 (рис. 10). На рис. 11 изображено то, что вывел `tcpdump` после `ping`-запроса. Как видно, данный хост существует и доступен.



```
root@ruvds-w39ps:~# ping -c1 10.0.0.2
```

Рисунок 10 – Проверка доступности хоста с адресом 10.0.0.2

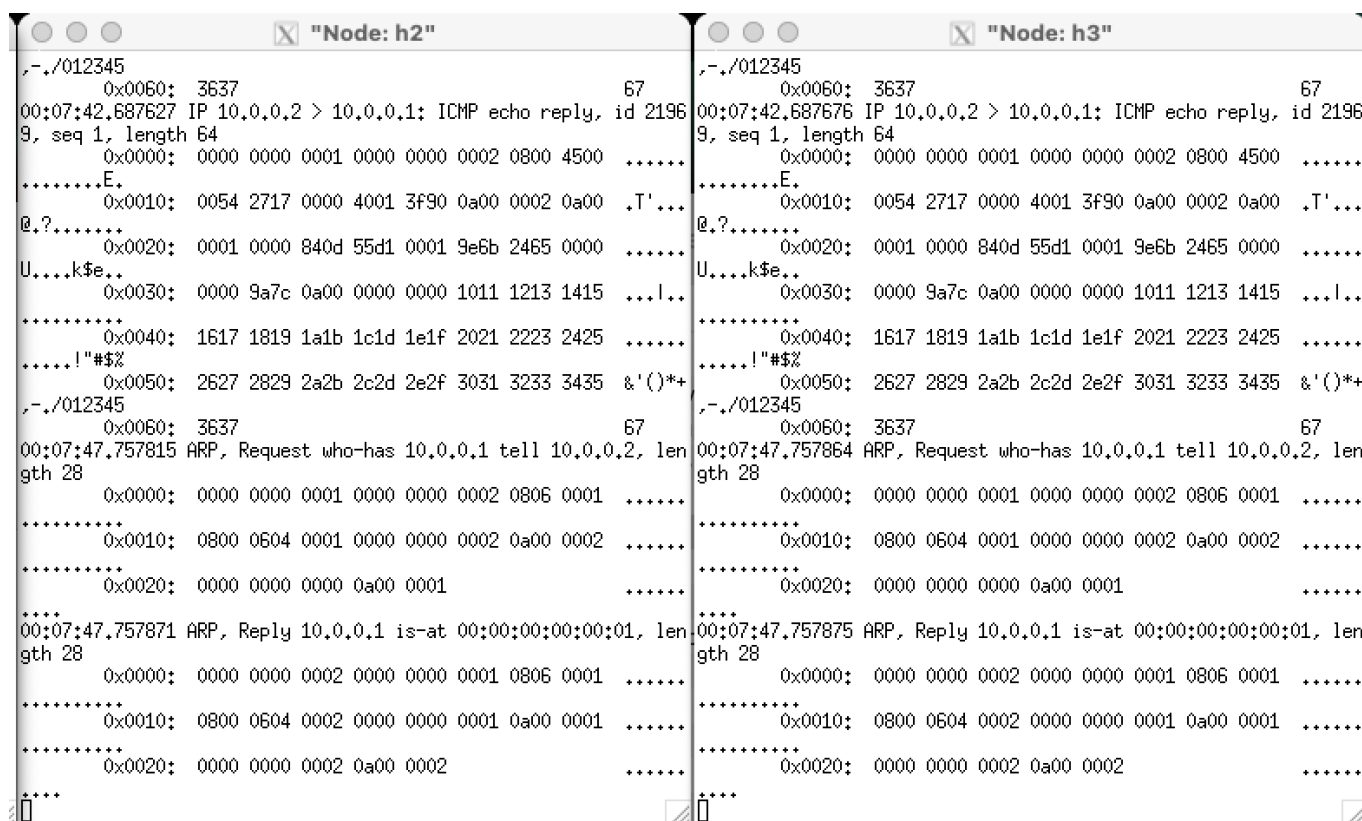


Рисунок 11 – Вывод tcpdump после первого запроса ping

В случае, если проверить доступность несуществующего хоста с адресом 10.0.0.5 (рис. 12), то в выводе tcpdump будет видно, что хост не был найден (рис. 13).

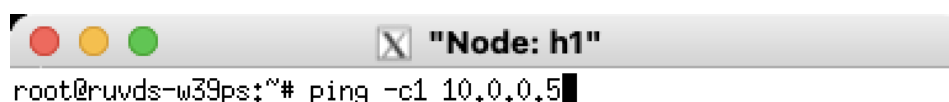


Рисунок 12 – Проверка доступности хоста с адресом 10.0.0.5



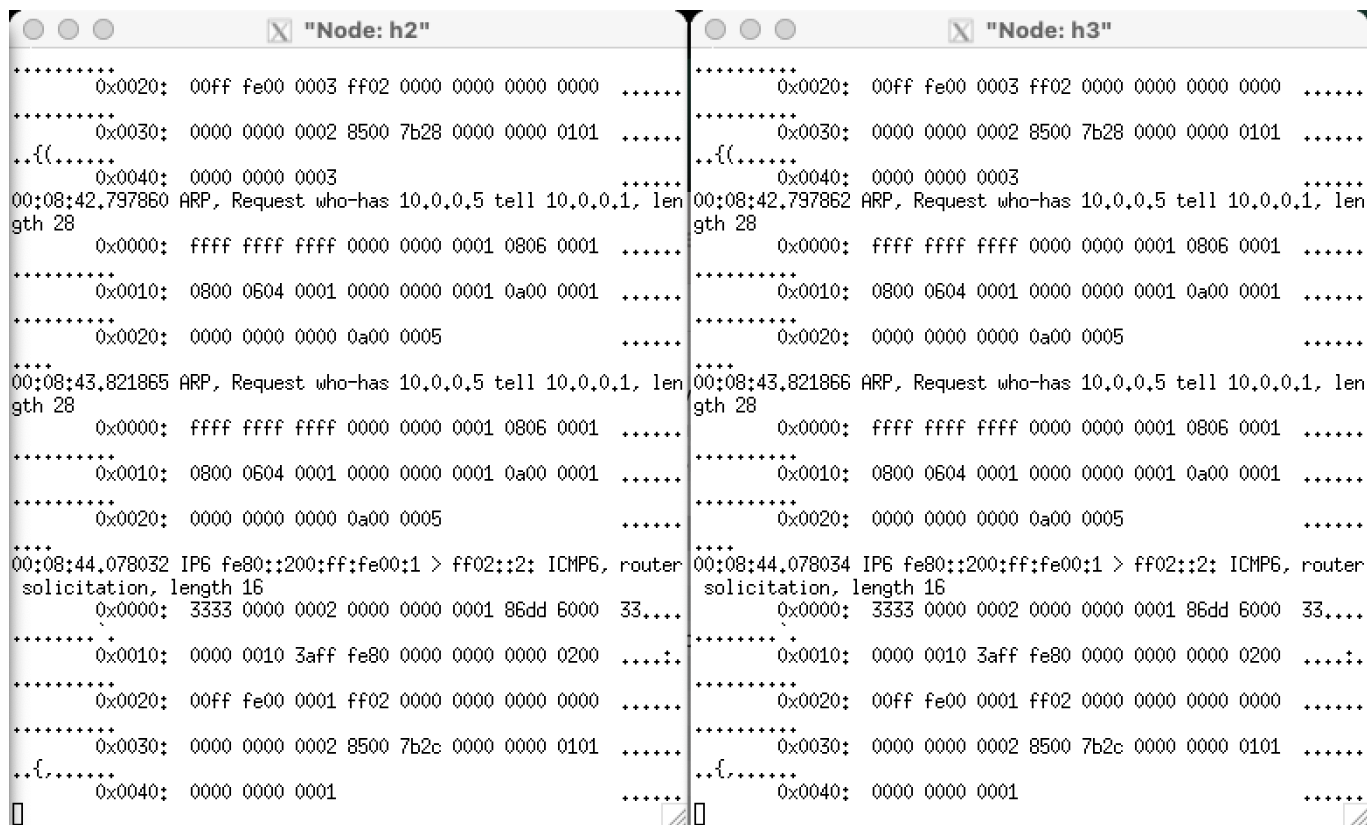


Рисунок 13 – Вывод tcpdump после второго запроса ping

## Заключение

**Вывод:** в данной лабораторной работой я научился работать с mininet, создавать скрипты, моделирующие различные сетевые топологии, а также тестировать доступность различных хостов.

## Приложение А

### Исходный код скрипта линейной топологии

---

```
#!/usr/bin/python3
```

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import irange, dumpNodeConnections
from mininet.log import setLogLevel

class LinearTopo(Topo):
    def __init__(self, k=2, **opts):
        super(LinearTopo, self).__init__(**opts)

        self.k = k

        lastSwitch = None
        for i in irange(1, k):
            host = self.addHost("h%s" % i, cpu=0.5 / k)
            switch = self.addSwitch("s%s" % i)
            self.addLink(
                host,
                switch,
                bw=10,
                delay="5ms",
                loss=1,
                max_queue_size=1000,
                use_htb=True,
            )
            if lastSwitch:
                self.addLink(
                    switch,
                    lastSwitch,
                    bw=10,
                    delay="5ms",
                    loss=1,
```

```
        max_queue_size=1000,  
        use_htb=True,  
    )  
    lastSwitch = switch
```

```
def perfTest():  
    topo = LinearTopo(k=4)  
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink)  
    net.start()  
    print("Dumping host connections")  
    dumpNodeConnections(net.hosts)  
    print("Testing network connectivity")  
    net.pingAll()  
    print("Testing bandwidth between h1 and h4")  
    h1, h4 = net.get("h1", "h4")  
    net.iperf((h1, h4))  
    net.stop()  
  
if __name__ == "__main__":  
    setLogLevel("info")  
    perfTest()
```

---

## Приложение Б

### Исходный код скрипта древовидной топологии

---

```
#!/usr/bin/python3

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class CustomTopo(Topo):
    "Simple Data Center Topology"

    "linkopts - (1:core, 2:aggregation, 3: edge) parameters"
    "fanout - number of child switch per parent switch"

    def __init__(self, linkopts1, linkopts2, linkopts3, fanout=2, **opts):
        Topo.__init__(self, **opts)

        self.__init_tree(
            fanout,
            level=0,
            parent=self.addSwitch("cs1"),
            level_info=[
                {"name": "as", "count": 0, "opts": linkopts1},
                {"name": "es", "count": 0, "opts": linkopts2},
                {"name": "hs", "count": 0, "opts": linkopts3},
            ],
        )

    def __init_tree(self, fanout, level, parent, level_info):
        if level >= len(level_info):
            return

        for index in range(fanout):
            info = level_info[level]
```

```

        info["count"] += 1
        name = info["name"] + str(info["count"])

        if level == len(level_info) - 1:
            device = self.addHost(name)
        else:
            device = self.addSwitch(name)

        self.addLink(device, parent, **info["opts"])

        self.__init_tree(fanout, level + 1, device, level_info)

def perfTest():
    opts = {
        "bw": 10,
        "delay": "5ms",
        "loss": 1,
        "max_queue_size": 1000,
        "use_htb": True,
    }
    topo = CustomTopo(opts, opts, opts)
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
    net.start()
    print("Dumping host connections")
    dumpNodeConnections(net.hosts)
    print("Testing network connectivity")
    net.pingAll()

if __name__ == "__main__":
    setLogLevel("info")
    perfTest()

```

---