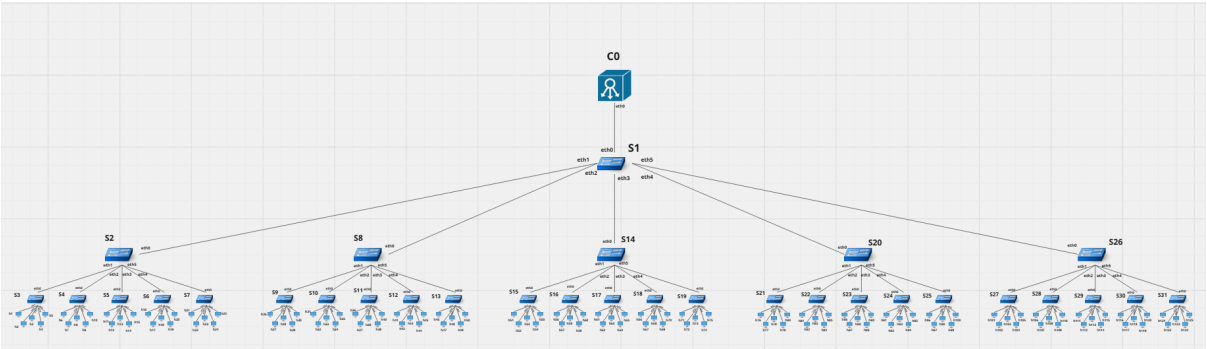
Trabalho Final - Mininet

Lanna Correia e Silva

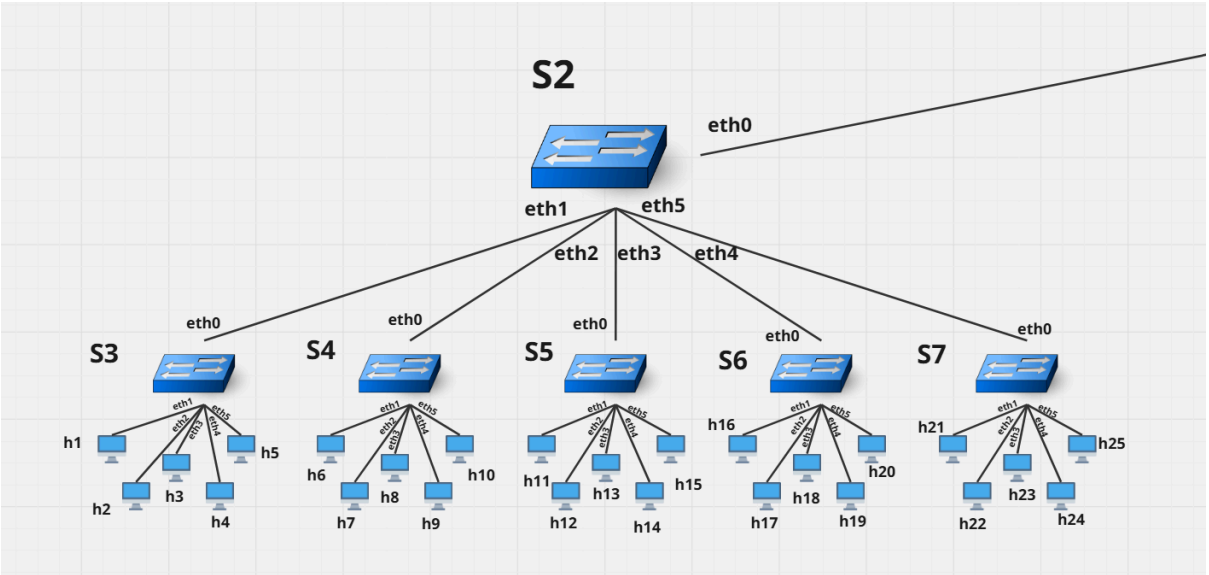
Questão 1.

Topologia:

A primeira topologia utilizada foi a de uma árvore com profundidade três e ramificação cinco, gerando 31 switches e 125 hosts no total. Como especificação, foi utilizado o modo de endereço MAC padronizado e larguras de banda de 30 Mbps. Para criar esta topologia, no terminal do PuTTy, utilizamos o comando “sudo mn –mac –topo=tree,depth=3,fanout=5 –link tc,bw=30”.



Topologia geral.



Topologia aproximada da primeira ramificação do switch s1, com a relação de portas utilizadas em cada conexão dos switches. Em todos os hosts, as portas utilizadas para conexão foram “eth0”.

```
mininet@mininet-vm:~$ sudo mn --mac --topo=tree,depth=3,fanout=5 --link tc,bw=30
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64 h65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81 h82 h
83 h84 h85 h86 h87 h88 h89 h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 h100 h101 h10
2 h103 h104 h105 h106 h107 h108 h109 h110 h111 h112 h113 h114 h115 h116 h117 h11
8 h119 h120 h121 h122 h123 h124 h125
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s
23 s24 s25 s26 s27 s28 s29 s30 s31
*** Adding links:
(30.00Mbit) (30.00Mbit) (s1, s2) _
```

Execução do comando de criação da topologia.

Inspeções:

A fim de verificar se a topologia foi criada corretamente, executamos alguns comandos, como “nodes” para verificar os nós (hosts, switches e controladores) gerados, “net” para visualizar a relação de portas nas ligações entre nós da rede, “dump” a fim de ver os endereços IP dos componentes, e “x ifconfig -a” (considerando x como um host ou switch qualquer) para ver o endereço MAC de cada nó e as configurações de suas portas.

```
mininet> nodes
available nodes are:
c0 h1 h10 h100 h101 h102 h103 h104 h105 h106 h107 h108 h109 h11 h110 h111 h112 h
113 h114 h115 h116 h117 h118 h119 h12 h120 h121 h122 h123 h124 h125 h13 h14 h15
h16 h17 h18 h19 h2 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h3 h30 h31 h32 h33 h3
4 h35 h36 h37 h38 h39 h4 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h5 h50 h51 h52
h53 h54 h55 h56 h57 h58 h59 h6 h60 h61 h62 h63 h64 h65 h66 h67 h68 h69 h7 h70 h7
1 h72 h73 h74 h75 h76 h77 h78 h79 h8 h80 h81 h82 h83 h84 h85 h86 h87 h88 h89 h9
h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 s1 s10 s11 s12 s13 s14 s15 s16 s17 s18 s
19 s2 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s3 s30 s31 s4 s5 s6 s7 s8 s9
```

A rede possui 31 switches, um controlador e 125 hosts.

```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s3-eth3
h4 h4-eth0:s3-eth4
h5 h5-eth0:s3-eth5
h6 h6-eth0:s4-eth1
h7 h7-eth0:s4-eth2
h8 h8-eth0:s4-eth3
h9 h9-eth0:s4-eth4
h10 h10-eth0:s4-eth5
h11 h11-eth0:s5-eth1
h12 h12-eth0:s5-eth2
h13 h13-eth0:s5-eth3
h14 h14-eth0:s5-eth4
```

Links entre as portas eth0 dos hosts h1-h5 em s3, h6-h10 em s4, h11-h15 em s5, e assim por diante.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3193>
<Host h2: h2-eth0:10.0.0.2 pid=3195>
<Host h3: h3-eth0:10.0.0.3 pid=3197>
<Host h4: h4-eth0:10.0.0.4 pid=3199>
<Host h5: h5-eth0:10.0.0.5 pid=3201>
<Host h6: h6-eth0:10.0.0.6 pid=3203>
<Host h7: h7-eth0:10.0.0.7 pid=3205>
<Host h8: h8-eth0:10.0.0.8 pid=3207>
<Host h9: h9-eth0:10.0.0.9 pid=3209>
<Host h10: h10-eth0:10.0.0.10 pid=3211>
<Host h11: h11-eth0:10.0.0.11 pid=3213>
<Host h12: h12-eth0:10.0.0.12 pid=3215>
<Host h13: h13-eth0:10.0.0.13 pid=3217>
<Host h14: h14-eth0:10.0.0.14 pid=3219>
<Host h15: h15-eth0:10.0.0.15 pid=3221>
<Host h16: h16-eth0:10.0.0.16 pid=3223>
<Host h17: h17-eth0:10.0.0.17 pid=3225>
<Host h18: h18-eth0:10.0.0.18 pid=3227>
<Host h19: h19-eth0:10.0.0.19 pid=3229>
<Host h20: h20-eth0:10.0.0.20 pid=3231>
<Host h21: h21-eth0:10.0.0.21 pid=3233>
<Host h22: h22-eth0:10.0.0.22 pid=3235>
<Host h23: h23-eth0:10.0.0.23 pid=3237>
<Host h24: h24-eth0:10.0.0.24 pid=3239>
```

Endereços IP dos 25 primeiros hosts.

```
<Host h120: h120-eth0:10.0.0.120 pid=3431>
<Host h121: h121-eth0:10.0.0.121 pid=3433>
<Host h122: h122-eth0:10.0.0.122 pid=3435>
<Host h123: h123-eth0:10.0.0.123 pid=3437>
<Host h124: h124-eth0:10.0.0.124 pid=3439>
<Host h125: h125-eth0:10.0.0.125 pid=3441>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None,
s1-eth5:None pid=3446>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None,
s2-eth5:None,s2-eth6:None pid=3449>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None,
s3-eth5:None,s3-eth6:None pid=3452>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None,s4-eth4:None,
s4-eth5:None,s4-eth6:None pid=3455>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None,s5-eth4:None,
s5-eth5:None,s5-eth6:None pid=3458>
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None,s6-eth4:None,
s6-eth5:None,s6-eth6:None pid=3461>
<OVSSwitch s7: lo:127.0.0.1,s7-eth1:None,s7-eth2:None,s7-eth3:None,s7-eth4:None,
s7-eth5:None,s7-eth6:None pid=3464>
<OVSSwitch s8: lo:127.0.0.1,s8-eth1:None,s8-eth2:None,s8-eth3:None,s8-eth4:None,
s8-eth5:None,s8-eth6:None pid=3467>
<OVSSwitch s9: lo:127.0.0.1,s9-eth1:None,s9-eth2:None,s9-eth3:None,s9-eth4:None,
s9-eth5:None,s9-eth6:None pid=3470>
<OVSSwitch s10: lo:127.0.0.1,s10-eth1:None,s10-eth2:None,s10-eth3:None,s10-eth4:
None,s10-eth5:None,s10-eth6:None pid=3473>
<OVSSwitch s11: lo:127.0.0.1,s11-eth1:None,s11-eth2:None,s11-eth3:None,s11-eth4:
None,s11-eth5:None,s11-eth6:None pid=3476>
<OVSSwitch s12: lo:127.0.0.1,s12-eth1:None,s12-eth2:None,s12-eth3:None,s12-eth4:
```

Endereços IP e informações de portas dos switches.

```
mininet> h25 ifconfig -a
h25-eth0  Link encap:Ethernet  HWaddr 00:00:00:00:00:19
          inet addr:10.0.0.25  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Informações específicas referentes às portas do host h25; MAC address = 00:00:00:00:00:19.

```
mininet> s1 ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:7e:0f:24
          inet addr:192.168.56.104  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1180 (1.1 KB)  TX bytes:684 (684.0 B)

eth1      Link encap:Ethernet  HWaddr 08:00:27:a7:c9:40
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:346 errors:0 dropped:0 overruns:0 frame:0
          TX packets:351 errors:0 dropped:0 overruns:0 carrier:0
          collisions:351 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:32884 (32.8 KB)  TX bytes:31110 (31.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:25479 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25479 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1362720 (1.3 MB)  TX bytes:1362720 (1.3 MB)

ovs-system Link encap:Ethernet  HWaddr 92:f9:86:5f:ac:5e
```

Informações específicas referentes às portas do switch s1; MAC address = 08:00:27:7e:0f:24.

Testes de conexão:

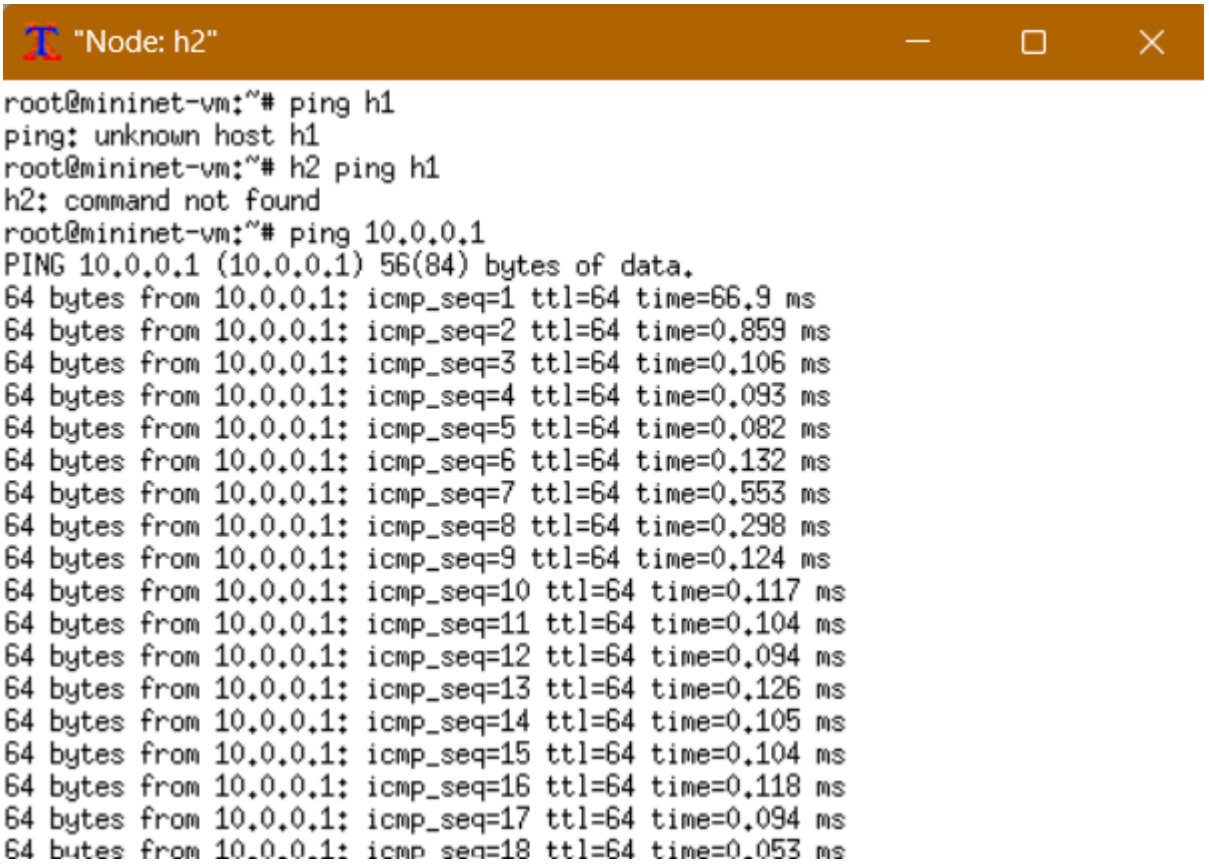
Para averiguar o funcionamento da rede na troca de dados, realizamos um teste de ping com 5 pacotes entre os hosts h1 e h2 e um teste de tcpdump entre os hosts h1 e h2, h25 e h26.

```
mininet> h1 ping -c 5 h2
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=13.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.138 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.044 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.043/2.835/13.902/5.533 ms
mininet>
```

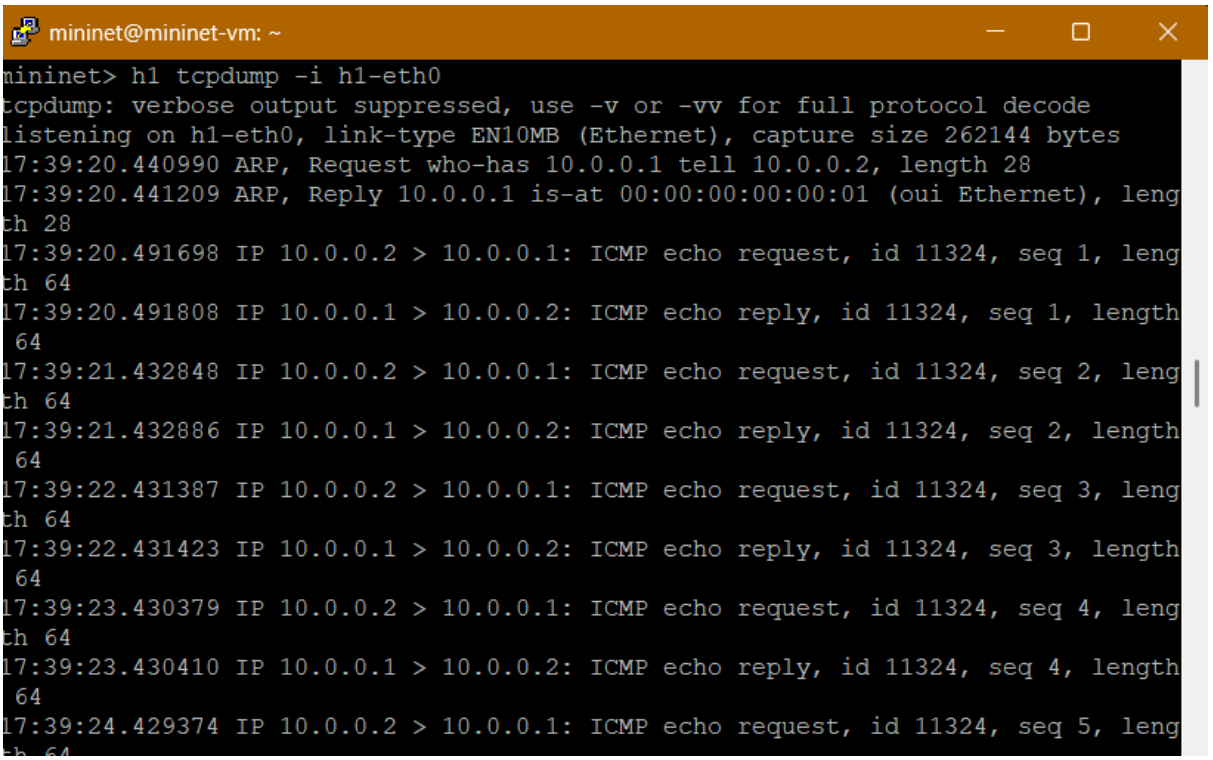
Conexão bem-sucedida no envio de dados de h1 para h2.

```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:574 (574.0 B)  TX bytes:574 (574.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> h1 tcpdump -i h1-eth0
```

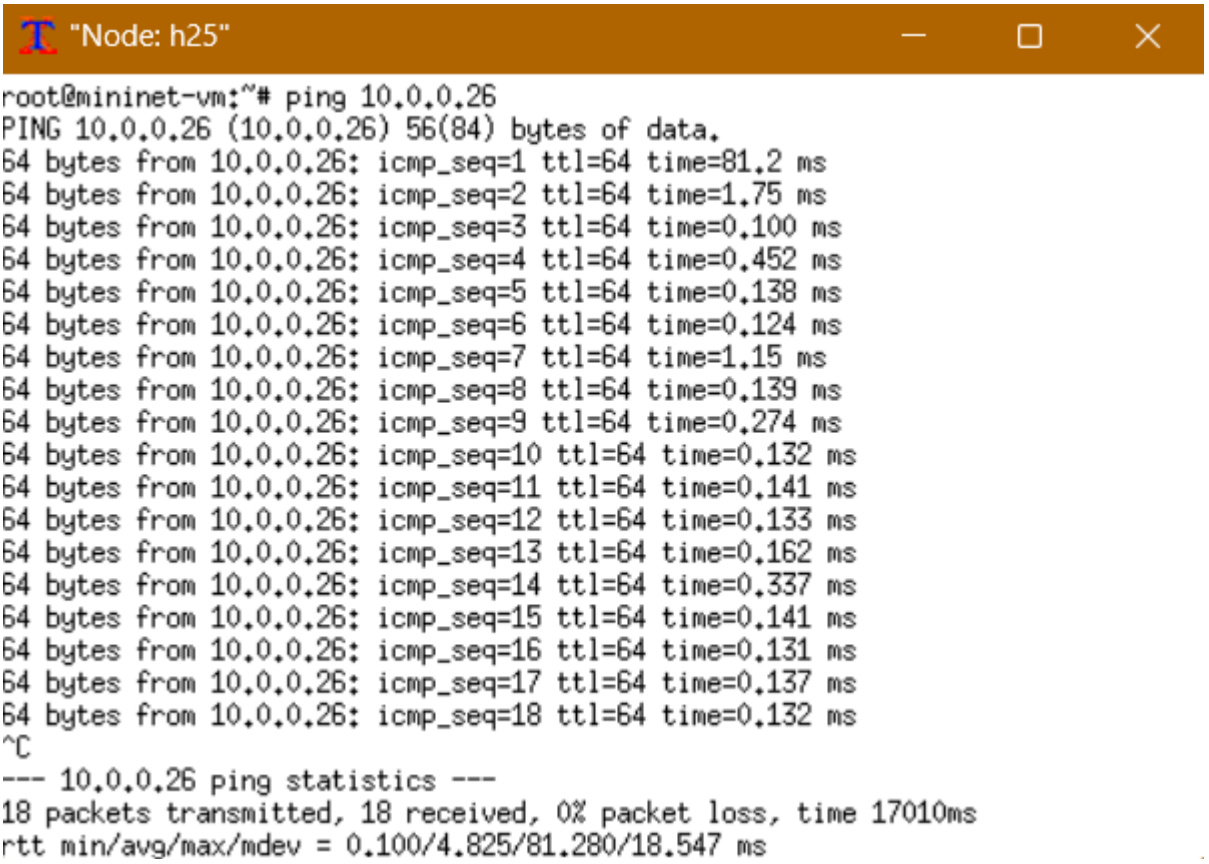
Verificação do endereço IP de h1 para possibilitar o envio de pacotes por parte de h25 e a escuta em tcpdump no host 1.



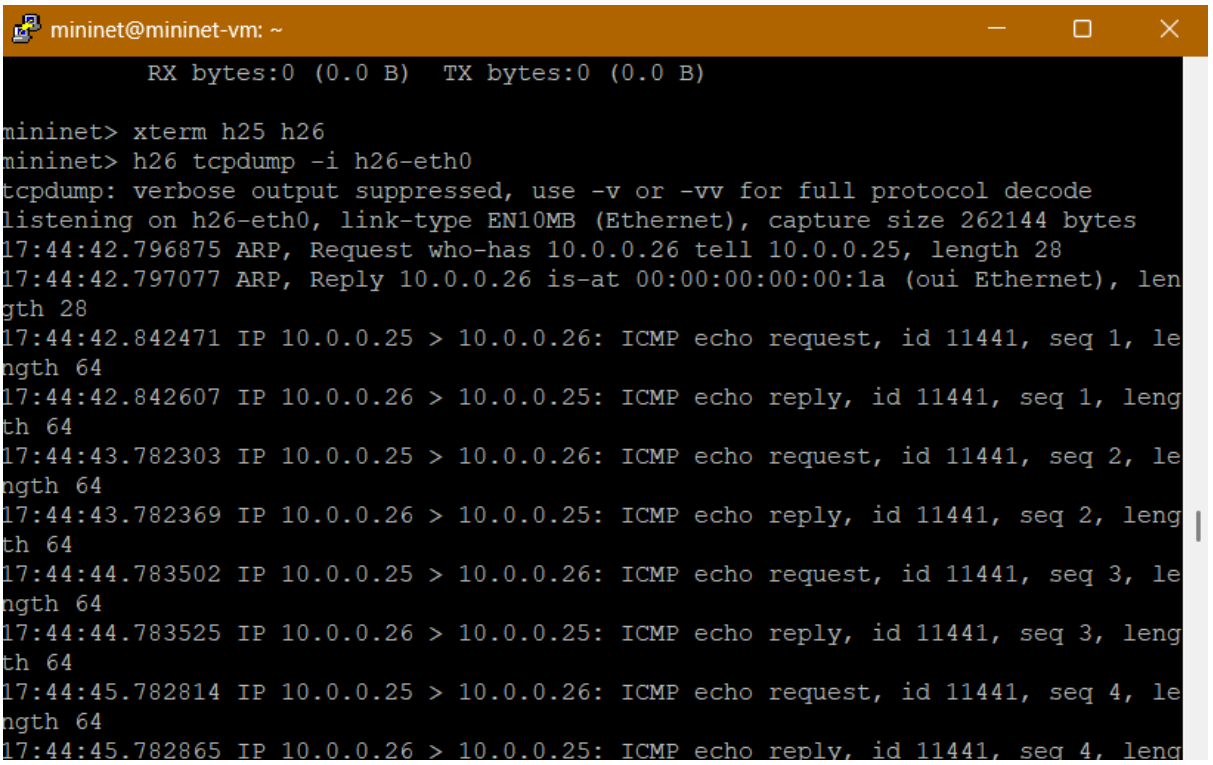
Execução do comando de ping para o IP 10.0.0.1 (host h1) pelo nó h2.



Recepção dos pacotes enviados pelo host h2.



Envio de pacotes do nó h25 para o host h26.



Recebimento dos pacotes de h25 por h26 no modo de escuta tcpdump.

Além dos testes de envio de pacotes, executamos dois testes de iperf (um de 30Mbps e o outro de 40Mbps de largura de banda) com o host h1 atuando como servidor TCP na porta 5555 e h2 como seu cliente.

30 Mbps:

"Node: h1"

```
root@mininet-vm:~# h1 tcpdump -i h1-eth0
h1: command not found
root@mininet-vm:~# iperf -s -p 5555 -i 1
------------------------------------------------------------
Server listening on TCP port 5555
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[320] local 10.0.0.1 port 5555 connected with 10.0.0.2 port 44968
[ ID] Interval       Transfer     Bandwidth
[320]  0.0- 1.0 sec  2.59 MBytes  21.7 Mbits/sec
[320]  1.0- 2.0 sec  3.16 MBytes  26.5 Mbits/sec
[320]  2.0- 3.0 sec  3.08 MBytes  25.9 Mbits/sec
[320]  3.0- 4.0 sec  3.15 MBytes  26.4 Mbits/sec
[320]  4.0- 5.0 sec  3.09 MBytes  25.9 Mbits/sec
[320]  5.0- 6.0 sec  3.11 MBytes  26.1 Mbits/sec
[320]  6.0- 7.0 sec  3.17 MBytes  26.6 Mbits/sec
[320]  7.0- 8.0 sec  3.17 MBytes  26.6 Mbits/sec
[320]  8.0- 9.0 sec  3.17 MBytes  26.6 Mbits/sec
[320]  9.0-10.0 sec  3.09 MBytes  26.0 Mbits/sec
[320] 10.0-11.0 sec  3.11 MBytes  26.1 Mbits/sec
[320] 11.0-12.0 sec  3.10 MBytes  26.0 Mbits/sec
[320] 12.0-13.0 sec  3.17 MBytes  26.6 Mbits/sec
[320] 13.0-14.0 sec  3.14 MBytes  26.4 Mbits/sec
```

"Node: h2"

```
50 packets transmitted, 50 received, 0% packet loss, time 49006ms
rtt min/avg/max/mdev = 0.053/1.483/66.996/9.359 ms
root@mininet-vm:~# iperf -c 10.0.0.1 -p 5555 -i 1 -t 20
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5555
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[319] local 10.0.0.2 port 44968 connected with 10.0.0.1 port 5555
[ ID] Interval       Transfer     Bandwidth
[319]  0.0- 1.0 sec  2.75 MBytes  23.1 Mbits/sec
[319]  1.0- 2.0 sec  3.38 MBytes  28.3 Mbits/sec
[319]  2.0- 3.0 sec  3.00 MBytes  25.2 Mbits/sec
[319]  3.0- 4.0 sec  3.12 MBytes  26.2 Mbits/sec
[319]  4.0- 5.0 sec  3.25 MBytes  27.3 Mbits/sec
[319]  5.0- 6.0 sec  3.00 MBytes  25.2 Mbits/sec
[319]  6.0- 7.0 sec  3.12 MBytes  26.2 Mbits/sec
[319]  7.0- 8.0 sec  3.12 MBytes  26.2 Mbits/sec
[319]  8.0- 9.0 sec  3.25 MBytes  27.3 Mbits/sec
[319]  9.0-10.0 sec  3.12 MBytes  26.2 Mbits/sec
[319] 10.0-11.0 sec  3.12 MBytes  26.2 Mbits/sec
[319] 11.0-12.0 sec  3.25 MBytes  27.3 Mbits/sec
[319] 12.0-13.0 sec  3.12 MBytes  26.2 Mbits/sec
[319] 13.0-14.0 sec  3.12 MBytes  26.2 Mbits/sec
```

Resultados do teste bem-sucedido.

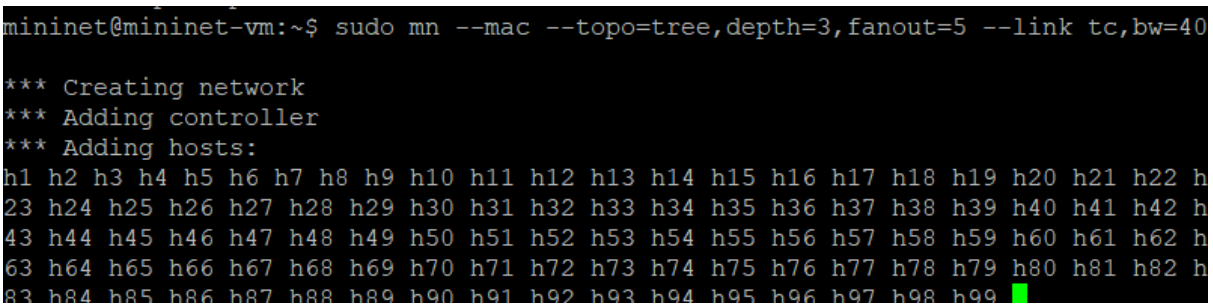
40 Mbps:

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 8 terms
*** Stopping 155 links
.....................................................................................
.....................................................................
*** Stopping 31 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s
23 s24 s25 s26 s27 s28 s29 s30 s31
*** Stopping 125 hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64 h65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81 h82 h
83 h84 h85 h86 h87 h88 h89 h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 h100 h101 h10
2 h103 h104 h105 h106 h107 h108 h109 h110 h111 h112 h113 h114 h115 h116 h117 h11
8 h119 h120 h121 h122 h123 h124 h125
*** Done
completed in 1799.746 seconds
mininet@mininet-vm:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
```

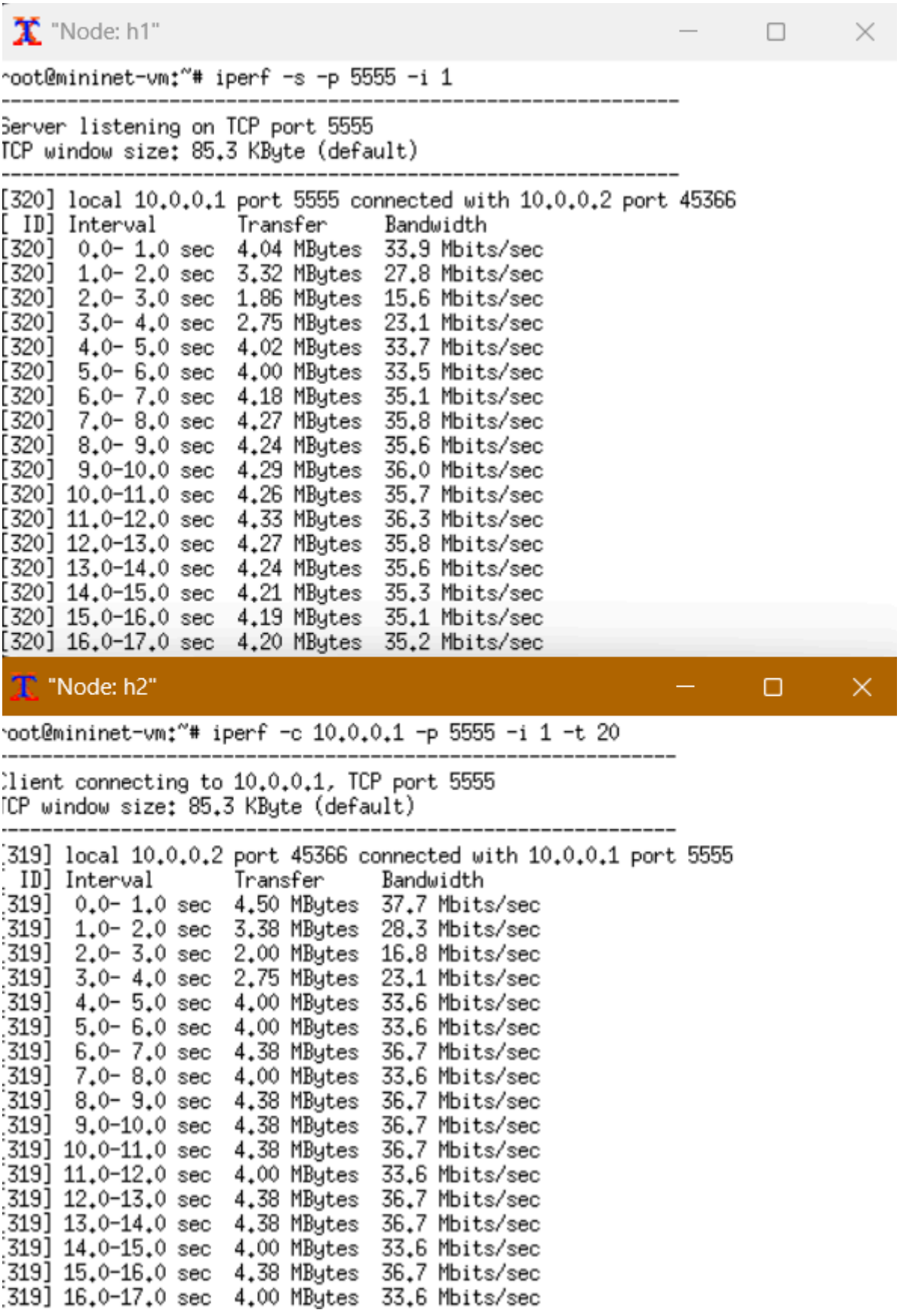
Reinicialização da topologia.

```
mininet@mininet-vm: ~
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd
 ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openfl
owd ovs-controller udpbwtest mnexec ivs 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 --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
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.
mininet@mininet-vm:~$
```

Topologia anterior apagada, vista em “Cleanup complete”.



Recriação da mesma topologia, mas agora com largura de banda de 40Mbps.



Resultado bem-sucedido do novo teste de iperf (com a mesma lógica do anterior de 30Mbps).

Conclusão:

Através destes testes bem-sucedidos, vê-se que a topologia foi criada corretamente e a rede é operante assim como o esperado.

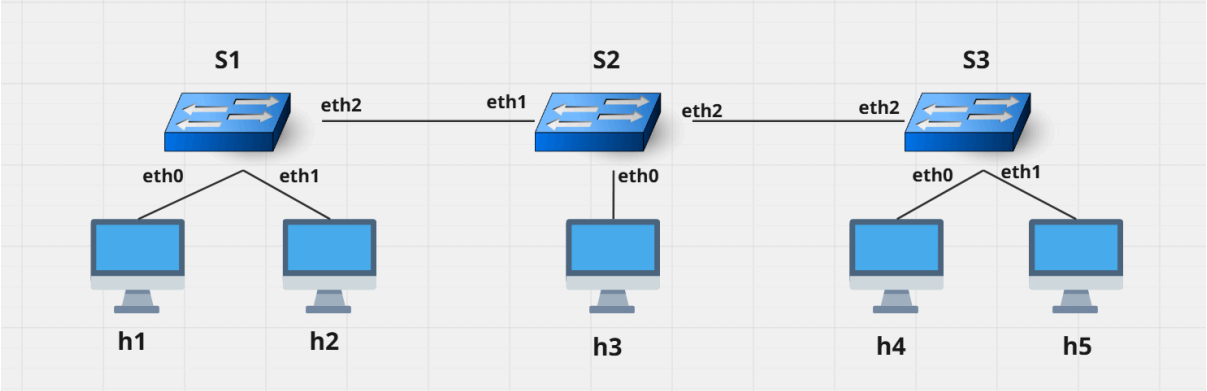
Questão 2.

Abordagem:

Agora, ao invés de utilizarmos puramente linhas de comando em terminal para gerar a rede com Mininet, usaremos a linguagem de programação Python como auxílio para administração da criação e gerenciamento da rede. O código principal referente a essa questão foi nomeado “trabalho-final-mininet_LannaC.py”.

Topologia:

Nesta segunda questão, é feito o uso de uma nova topologia. A mesma é composta por três switches (s1, s2 e s3) e cinco hosts (h1-h5) e as ligações entre nós são feitas de acordo com a imagem a seguir.



Para a replicação da mesma em código, utilizamos a classe “NetworkTopo(Topo)” para adicionar switches e hosts, além de estabelecer as conexões entre eles, com endereços MAC padronizados e controlador manual (visto em “controller=None”) .

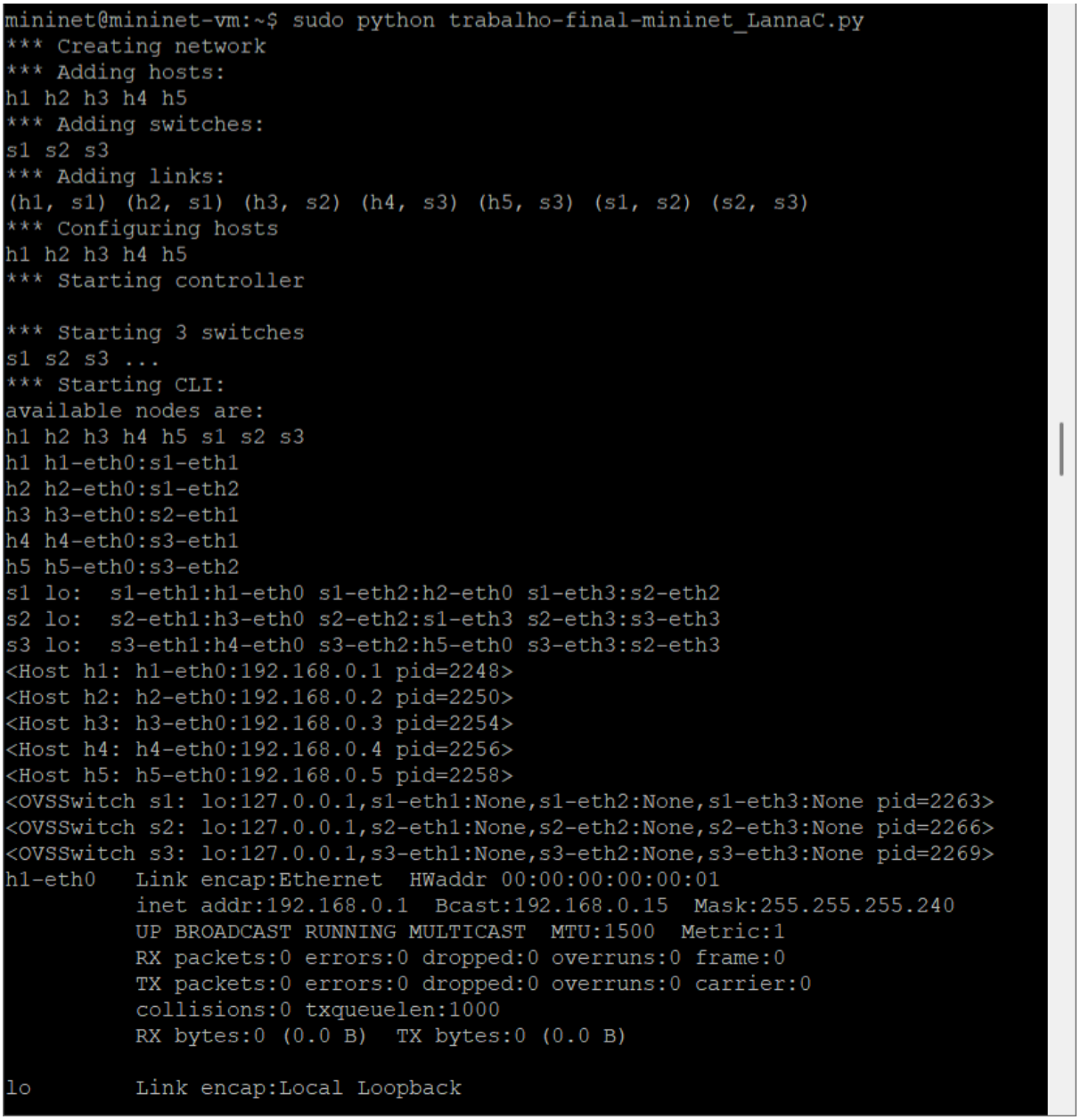
```python
class NetworkTopo( Topo ):
    # Builds network topology
    def build(self, **_opts):

        s1 = self.addSwitch('s1', failMode='standalone')
        s2 = self.addSwitch('s2', failMode='standalone')
        s3 = self.addSwitch('s3', failMode='standalone')

        # Adding hosts
        h1 = self.addHost('h1',ip='192.168.0.1/28',mac='00:00:00:00:00:01')
        h2 = self.addHost('h2',ip='192.168.0.2/28',mac='00:00:00:00:00:02')
        h3 = self.addHost('h3',ip='192.168.0.3/28',mac='00:00:00:00:00:03')
        h4 = self.addHost('h4',ip='192.168.0.4/28',mac='00:00:00:00:00:04')
        h5 = self.addHost('h5',ip='192.168.0.5/28',mac='00:00:00:00:00:05')

        # Connecting hosts to switches
        for d, s in [(h1, s1), (h2, s1), (h3,s2), (h4,s3), (h5,s3)]:
            self.addLink(d, s)

        #Connecting switches
        self.addLink(s1, s2)
        self.addLink(s2, s3)
```

Configuração dos switches e hosts.

```python
def run():
    topo = NetworkTopo()
    net = Mininet(topo=topo, controller=None)
    #net.addController('c0')
    net.start()
    CLI( net, script="nodes.sh")
    h1 = net.get('h1')

    print(h1.cmd('sudo ovs-ofctl del-flows s1'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s1'))
    print(h1.cmd('sudo ovs-ofctl del-flows s2'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s2'))
    print(h1.cmd('sudo ovs-ofctl del-flows s3'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s3'))
    #print(h1.cmd('sudo mn --custom topo-3s-5h.py --topo mytopo'))


    #h1 e h2
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:2')
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:1')

    #h1 e h3
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,actions=output:3')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,actions=output:1')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,actions=output:2')
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,actions=output:1')

    #h1 e h4
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,actions=output:3')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,actions=output:3')
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,actions=output:1')
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:3')
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:1')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:2')
```

```python
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:1')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:2')

    #h4 e h5
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:05,actions=output:2')
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:04,actions=output:1')


    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,action=flood')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_type=0x806,nw_proto=1,action=flood')
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_type=0x806,nw_proto=1,action=flood')

    print(h1.cmd('sudo ovs-ofctl dump-flows s1'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s2'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s3'))


    CLI( net, script="ping.sh")

    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    run()
```

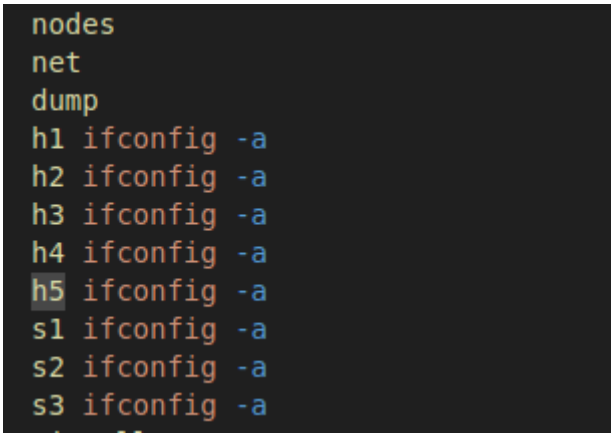
Ativação da topologia, execução do script e comandos para terminal.



Criação bem-sucedida da topologia da rede.

Inspeções:

Para verificar se a topologia foi criada como o esperado, executamos alguns comandos, como “nodes” para verificar os nós (hosts e switches) gerados, “net” para visualizar a relação de portas nas conexões da rede, “dump” a fim de ver os endereços IP dos dispositivos, e “x ifconfig -a” (considerando x como um host ou switch qualquer) para ver o endereço MAC de cada nó e as configurações de suas portas.

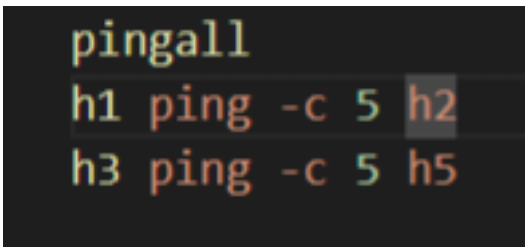


Comandos que serão executados no terminal.

Exemplo de execução do comando “ifconfig”.

Testes de conexão:

Para analisar o funcionamento da rede na troca de dados, realizamos testes de ping através do comando “pingall” e do “ping” com o envio de 5 pacotes entre os hosts h1 e h2, h3 e h5.



Sequência de comandos de envio de pacotes de teste.

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.041 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.049 ms
64 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=0.038 ms

--- 192.168.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.038/0.047/0.071/0.013 ms
PING 192.168.0.5 (192.168.0.5) 56(84) bytes of data.
64 bytes from 192.168.0.5: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 192.168.0.5: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 192.168.0.5: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 192.168.0.5: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 192.168.0.5: icmp_seq=5 ttl=64 time=0.041 ms

--- 192.168.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3998ms
rtt min/avg/max/mdev = 0.027/0.044/0.072/0.014 ms

NXST_FLOW reply (xid=0x4):


NXST_FLOW reply (xid=0x4):


NXST_FLOW reply (xid=0x4):

NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.127s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
```

Respostas ao comando de “pingall” e pings específicos entre h1 e h2 e h3 e h5.

A seguir, apagamos todas as regras que haviam sobre tráfego de pacotes nos switches com a reinicialização de todos os links da topologia.

```
h1 = net.get('h1')

print(h1.cmd('sudo ovs-ofctl del-flows s1'))
print(h1.cmd('sudo ovs-ofctl dump-flows s1'))
print(h1.cmd('sudo ovs-ofctl del-flows s2'))
print(h1.cmd('sudo ovs-ofctl dump-flows s2'))
print(h1.cmd('sudo ovs-ofctl del-flows s3'))
print(h1.cmd('sudo ovs-ofctl dump-flows s3'))
```

Exclusão apenas dos links da topologia, sem deletá-la.

```
NXST_FLOW reply (xid=0x4):


NXST_FLOW reply (xid=0x4):


NXST_FLOW reply (xid=0x4):

NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.127s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=0.194s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=0.138s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=0.182s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=0.171s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=0.082s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=0.034s, table=0, n_packets=0, n_bytes=0, idle_age=0, arp,a
rp_op=1 actions=FLOOD

NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=0.132s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=output:3
 cookie=0x0, duration=0.165s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:2
 cookie=0x0, duration=0.176s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:1
 cookie=0x0, duration=0.088s, table=0, n_packets=0, n_bytes=0, idle_age=0, dl_sr
c=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:2
```

Resultados da exclusão.

E, assim, escrevemos algumas novas regras de tráfego com base nos endereços MAC dos hosts para execução no terminal.

```
#h1 e h2
net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:2')
net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:1')

#h1 e h3
net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,actions=output:3')
net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,actions=output:1')
net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,actions=output:2')
net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,actions=output:1')

#h1 e h4
net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,actions=output:3')
net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,actions=output:3')
net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,actions=output:1')
net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:3')
net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:1')
net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:2')
```

```
    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:1')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01,actions=output:2')

    #h4 e h5
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:05,actions=output:2')
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:04,actions=output:1')


    net['s1'].cmd('sudo ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,action=flood')
    net['s2'].cmd('sudo ovs-ofctl add-flow s2 dl_type=0x806,nw_proto=1,action=flood')
    net['s3'].cmd('sudo ovs-ofctl add-flow s3 dl_type=0x806,nw_proto=1,action=flood')

    print(h1.cmd('sudo ovs-ofctl dump-flows s1'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s2'))
    print(h1.cmd('sudo ovs-ofctl dump-flows s3'))


    CLI( net, script="ping.sh")

    net.stop()
if __name__ == '__main__':
    setLogLevel( 'info' )
    run()
```

Novas regras de tráfego nos switches s1, s2 e s3.

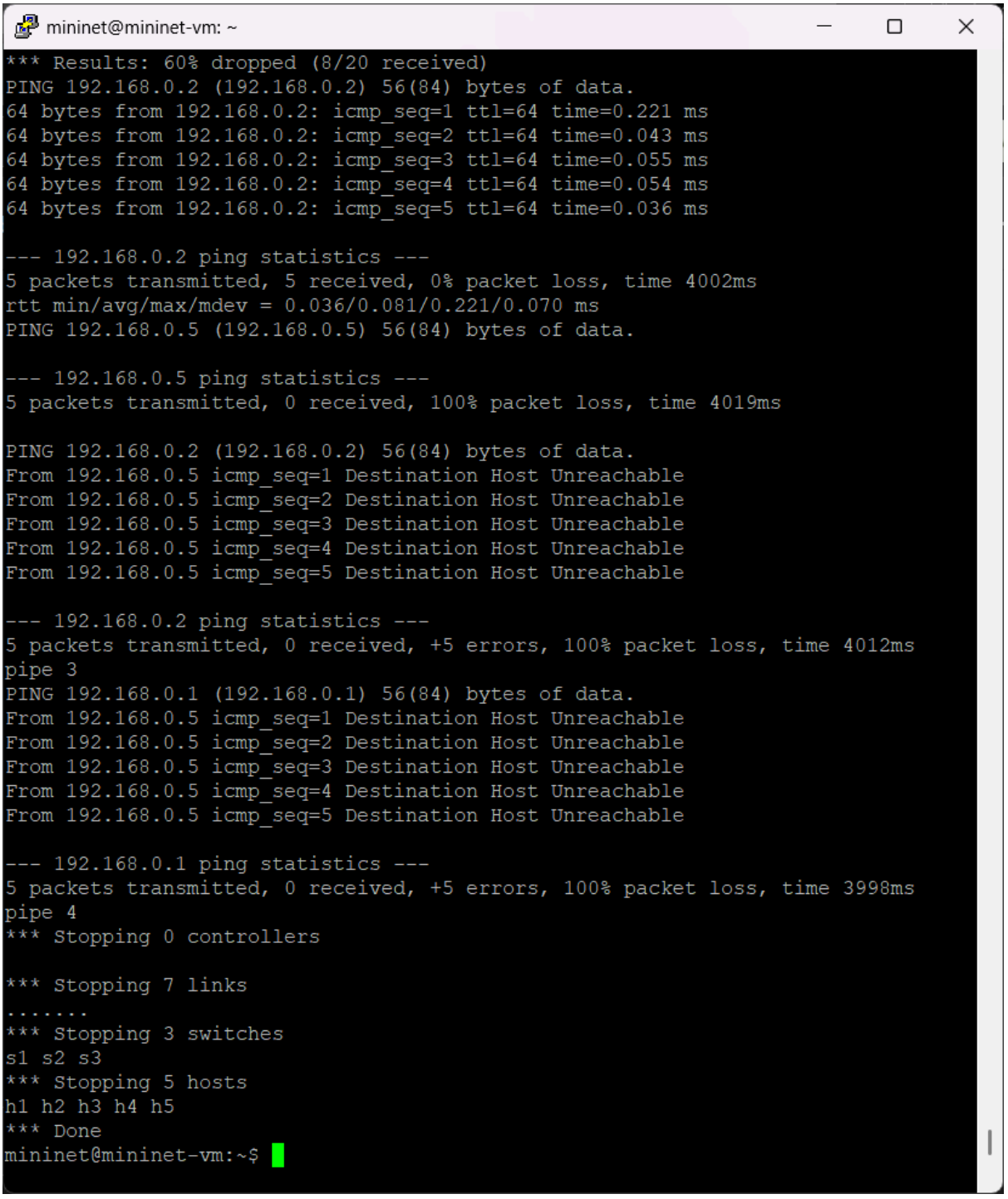
Por fim, para garantir que as novas regras foram contabilizadas e as anteriores foram apagadas, executamos novos testes de ping.

```
pingall
h1 ping -c 5 h2
h3 ping -c 5 h5
h5 ping -c 5 h2
h5 ping -c 5 h1
```

Novos testes de ping.

É importante ressaltar que nem todos os casos do “pingall” devem ser bem-sucedidos, como por exemplo o envio de pacotes de h5 à h1 (“h5 ping -c 5 h1”), já que nem todos os links entre hosts foram restabelecidos. Isso é confirmado no resultado da execução desses comandos, exposto a seguir.

```
*** Starting CLI:
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X
h2 -> h1 X X X
h3 -> h1 X X X
h4 -> h1 X X h5
h5 -> X X X h4
*** Results: 60% dropped (8/20 received)
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.221 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 192.168.0.2: icmp_seq=5 ttl=64 time=0.036 ms

--- 192.168.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.036/0.081/0.221/0.070 ms
PING 192.168.0.5 (192.168.0.5) 56(84) bytes of data.

--- 192.168.0.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4019ms

PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
From 192.168.0.5 icmp_seq=1 Destination Host Unreachable
From 192.168.0.5 icmp_seq=2 Destination Host Unreachable
From 192.168.0.5 icmp_seq=3 Destination Host Unreachable
From 192.168.0.5 icmp_seq=4 Destination Host Unreachable
From 192.168.0.5 icmp_seq=5 Destination Host Unreachable
```



Execução bem-sucedida com base no esperado para o que foi criado de conexões MAC entre hosts.

Referências:

https://vecta.io/symbols/240/cisco-network-topology-icons-3015/241/system-controller
https://www.freeiconspng.com/img/27057