# EA080: Laboratório 04

**Vinícius Esperança Mantovani,**
**247395.**

## Introdução)

Neste experimento, objetivou-se conhecer a topologia criada na sala de aula de EA080 da FEEC, por meio de emulação com o *mininet*. Para tanto, visando-se ainda explorar ferramentas do *mininet*, foram criadas as topologias deste laboratório usando o *miniedit*, conforme se nota ao longo do desenvolvimento. Vale ressaltar que pôde-se aprofundar em conhecimentos de redes, seguindo o procedimento recomendado, de modo a se montar aquilo que se via em hardware na sala em software com o emulador.

## Atividade 1)

Nesta primeira atividade, foi montada uma topologia, exposta na *Figura 1*, na qual se tem um switch que liga um conjunto de switches e hosts a um conjunto de roteadores. Isso se fez com o uso da ferramenta *miniedit* e a figura citada é uma imagem da topologia em tal ferramenta.

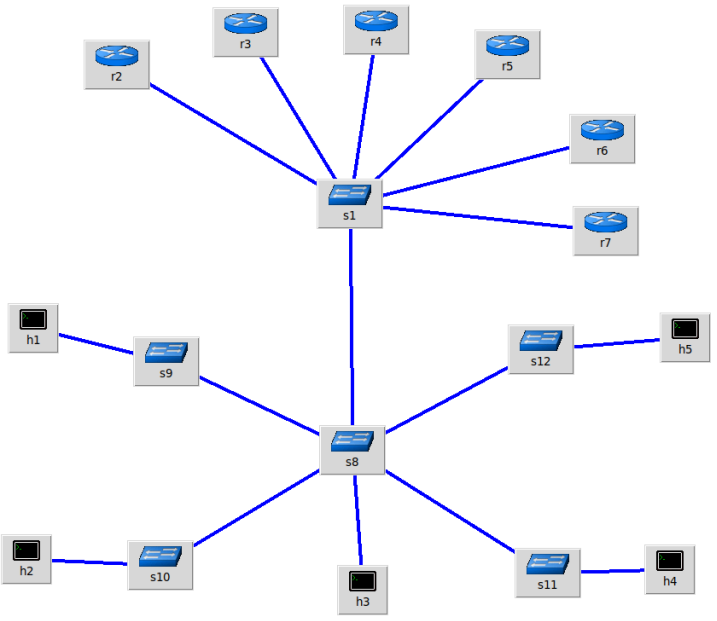


*Figura 1: topologia de rede montada na atividade 1*

Após a criação da topologia e sua exportação como *script python*, foi alterado o "*ipbase*" nesse *script* para que o endereçamento dos nós fosse semelhante àquele da topologia que se busca representar (*Figura 2*):

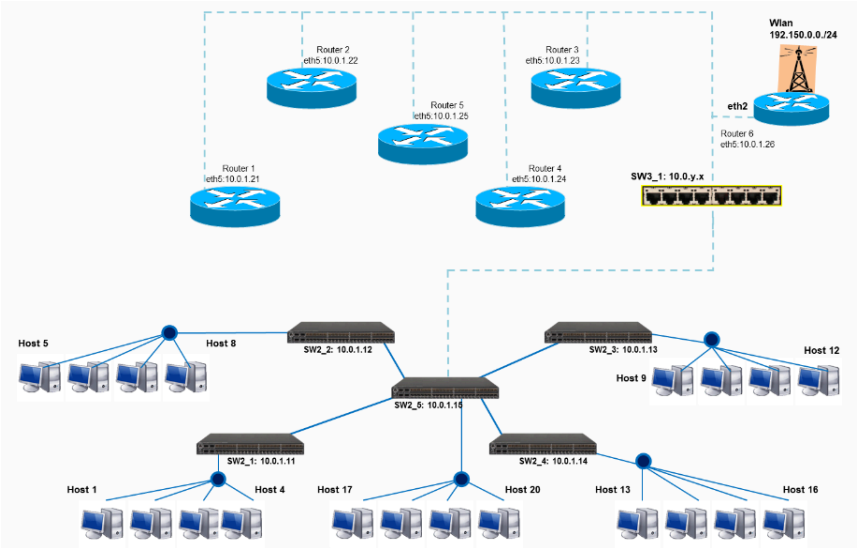


*Figura 2: Topologia a ser emulada na atividade 1*

Seguindo, foi usado o comando ping entre pares de hosts para análise de conectividade e, ao passo em que se fez isso, foram também *printadas* as tabelas *ARP* de cada nó que contém tal tabela (hosts). Isso se expõe nas figuras a seguir, em que se apresenta o teste de *h1* com rumo aos outros nós:

```
mininet> h1 ping h2 -c2
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=0.766 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.116 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 0.116/0.441/0.766/0.325 ms
```

*Figura 3: Ping de h1 para h2*

```
mininet> h1 arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.2                 ether   c2:de:9a:44:9b:c6   C                     h1-eth0
```

*Figura 4: ARP h1 após ping com h2*

```
mininet> h2 arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.1                 ether   02:aa:7e:d9:f2:22   C                     h2-eth0
```

*Figura 5: ARP h2*

```
mininet> h1 ping h3 -c2
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.13 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.086 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.086/0.607/1.129/0.521 ms
```

*Figura 6: Ping de h1 para h3*

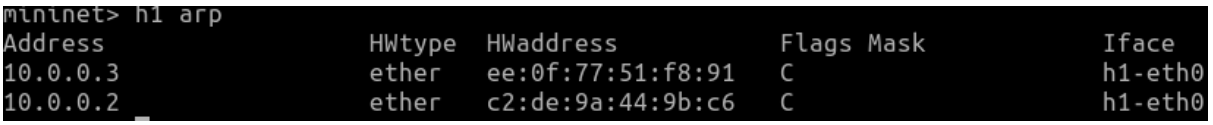


*Figura 7: ARP h1 após ping com h3*


*Figura 8: ARP h3*

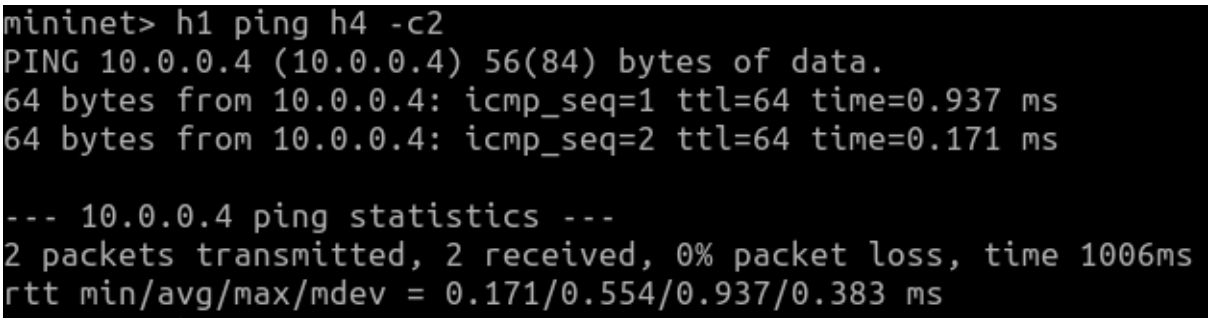


*Figura 9: Ping de h1 para h4*

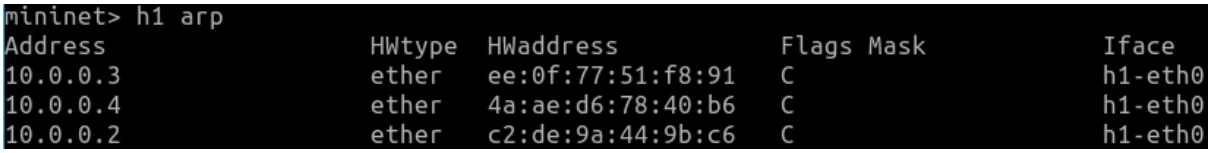


*Figura 10: ARP h1 após ping com h4*

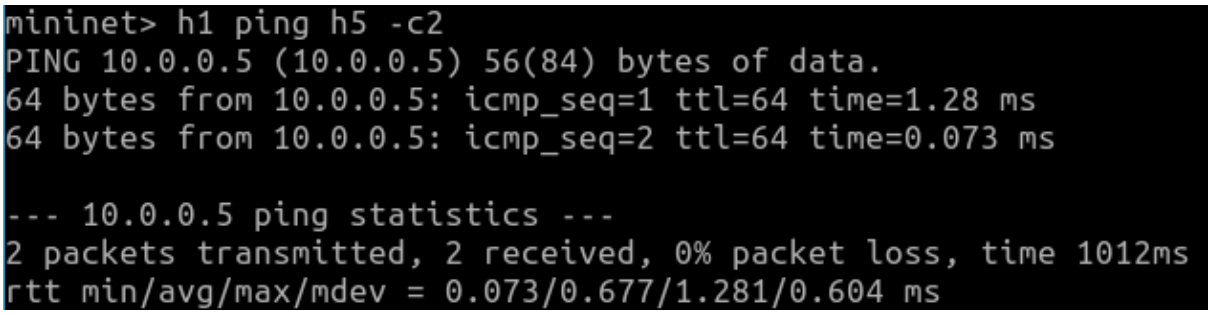


*Figura 11: Ping de h1 para h5*

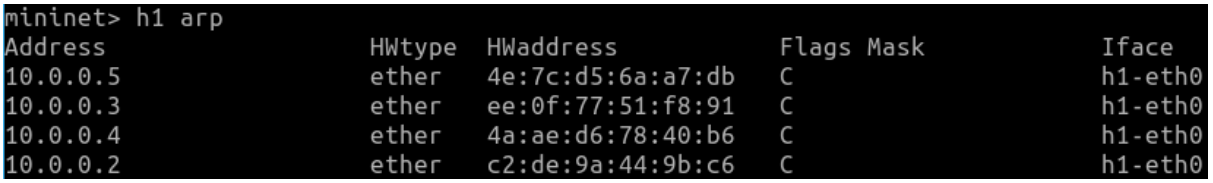


*Figura 12: ARP h1 após ping com h5*

Observando as figuras acima, é possível notar que os hosts estão todos conectados entre sí e que a tabela *ARP* de *h1* é incrementada a cada *ping*, o que se esperava teoricamente, uma vez que a os hosts ainda não tiveram contato entre si. Desse modo, quando *h1* busca por um outro host usando o IP, ele precisa aprender o *MAC* de tal host e, nesse momento, *h1* adiciona esse MAC em sua tabela ARP

para lembrá-lo em próximas conexões. Além disso, nota-se o mesmo com os demais hosts, que ao receberem *ping* de *h1* adicionam os endereços IP e MAC deste host em suas tabelas ARP (não foram apresentadas todas, pois são iguais, possuem apenas endereço de *h1*). Por fim, vale ressaltar que os roteadores estão todos causando problemas de conectividade quando se executa um comando *ping*, conforme figura abaixo, que apresenta o resultado do *ping* de h1 para um roteador.

```
mininet> h1 ping r4 -c2
ping: r4: Temporary failure in name resolution
```

*Figura 13: Ping falho entre h1 e r4*

Na figura acima, pode-se verificar o fenômeno citado e, analisando-se o *script* gerado pelo *miniedit*, nota-se que os roteadores estão todos com IP = 0.0.0.0, o que ocasiona o problema em questão. Na figura seguinte, vê-se o IP com que é definido um dos roteadores, o que se expande a todos os demais:

```
r5 = net.addHost('r5', cls=Node, ip='0.0.0.0')
r5.cmd('sysctl -w net.ipv4.ip_forward=1')
```

*Figura 14: Definição de um roteador no script da topologia*

Seguindo, ainda com os roteadores inalterados, *printou*-se a tabela de roteamento do host 1, conforme se segue:

```
mininet> h1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 h1-eth0
```

*Figura 15: Tabela de roteamento de h1*

Ressaltando-se que somente a do host 1 foi impressa, por causa de serem todas iguais (com exceção da interface), percebe-se que o valor de *Gateway* está como “0.0.0.0” e o de *Destination* está como “10.0.0.0”, enquanto a máscara indica que 24 bits são referentes à subrede. Isso, no entanto, não apresenta informações relevantes a respeito dos roteadores, uma vez que o *Gateway* apresentado representa mais que um roteador e, nenhum dos roteadores da rede, conforme explicado anteriormente, tem endereço IP relevante, logo, nenhum deles participa de conexão alguma (mesmo que houvessem hosts entre redes) e, o *Gateway* apresentado, provavelmente só indica que não há conexão com roteadores, não o IP de um deles.

Finalmente, observando a *Figura 14*, novamente, pode-se notar que os roteadores são definidos como um host e, em seguida, são providos da capacidade de repasse de pacotes por meio do comando “sysctl -w net.ipv4.ip_foward=1”. Logo, é necessária a segunda linha de sua definição, para que não sejam mantidos apenas como hosts, mas sim como roteadores.

**Atividade 2)**

Obs: Vale destacar que o roteador 1 do modelo de topologia desejado foi representado no roteador 6 da topologia montada e, o host 101, no host 6 da topologia montada!

Analisando a *Figura 16*, pode-se notar que a topologia contém duas subredes, o que implica a existência de dois domínios *broadcast*, um para cada subrede. Ademais, quanto às faixas de endereços de cada uma das redes, tem-se:

Rede 10.0.2.0/23 → Faixa de 10.0.2.1 a 10.0.3.254; IP de broadcast = 10.0.3.255;
Rede 10.0.0.0/23 → Faixa de 10.0.0.1 a 10.0.1.254; IP de broadcast = 10.0.1.255;

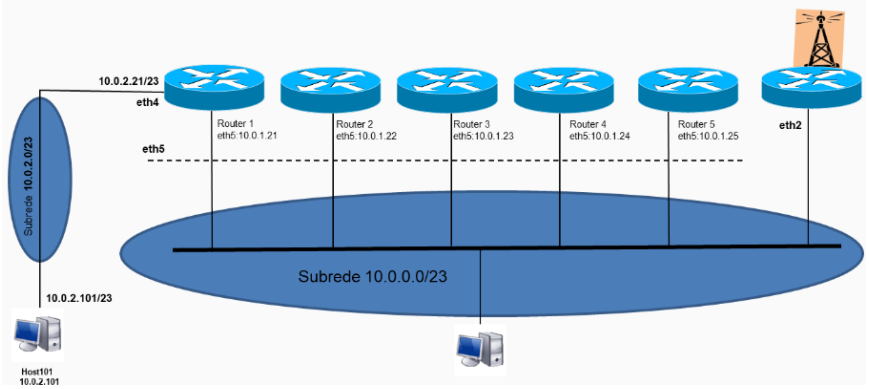


*Figura 16: Topologia modelo na atividade 2*

Buscando-se uma representação desta topologia da figura acima no emulador *mininet*, foi criado no *miniedit* a topologia apresentada abaixo:

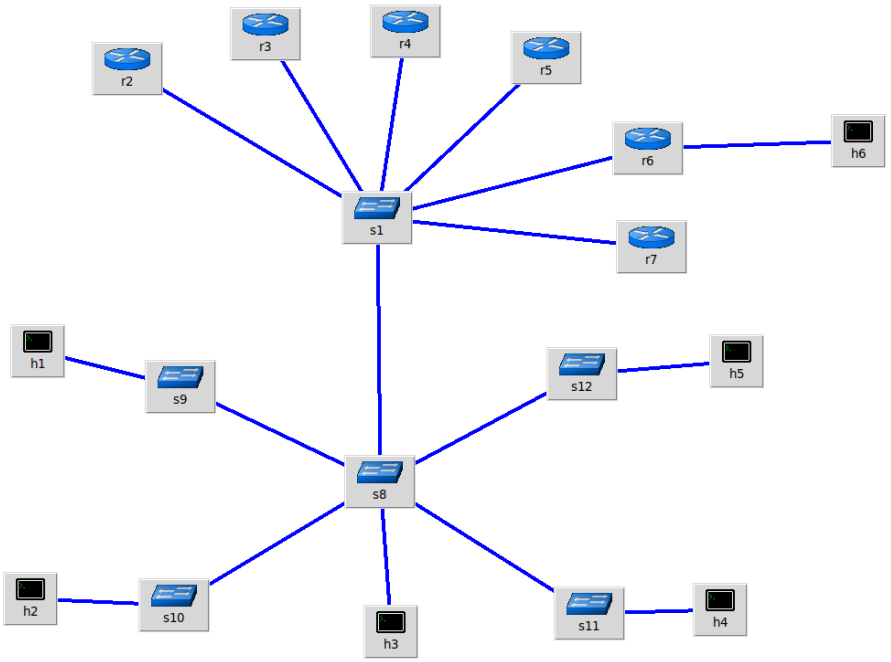


*Figura 17: Topologia montada na atividade 2*

Conforme se percebe, essa topologia é a mesma da usada na atividade 1, mas com um novo host conectado a um dos roteadores (aqui o h6 conectado a r6). Nesse sentido, para definir a rede a que o novo host pertence, é feita uma série de mudanças em h6 e r6, conforme se segue:

```
h6 = net.addHost('h6', cls=Host, ip='10.0.2.1', defaultRoute=None)
```

*Figura 18: Alteração do IP de h6*

Inicialmente, como na *Figura 18*, foi alterado o IP de *h6*. Em seguida, pelo sistema do roteador *r6*, foi alterado o IP de suas interfaces com h6 e com a outra subrede, conforme se segue:

```
r6.cmd("ifconfig r6-eth0 10.0.2.21/23")
r6.cmd("ifconfig r6-eth1 10.0.1.21/23")
```

*Figura 19: Alteração do IP de r6 nas interfaces de r6*

```
h6 = net.addHost('h6', cls=Host, ip='10.0.2.1', defaultRoute='via 10.0.2.21')
```

*Figura 20: Alteração do IP de r6 na segunda interface (r6-eth1)*

Seguindo, após as alterações citadas, foram encontradas as tabelas de roteamento do host 6 adicionado e do roteador conectado a ele, conforme abaixo:

```
root@wifi-virtualbox:/home/wifi/EA080-2S2021# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.254.0   U     0      0        0 r6-eth1
10.0.2.0        0.0.0.0         255.255.254.0   U     0      0        0 r6-eth0
```

*Figura 21: Tabela de roteamento de r6*

```
root@wifi-virtualbox:/home/wifi/EA080-2S2021# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.21       0.0.0.0         UG    0      0        0 h6-eth0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 h6-eth0
```

*Figura 22: Tabela de roteamento de h6*

Nessas figuras, é possível notar que o host está devidamente “conectado” ao roteador e que esse roteador tem duas interfaces. Isso, de modo que o *Gateway* padrão para h6 foi devidamente *setado* como sendo o roteador 6 e os caminhos para os quais seguem os dados que chegam ao roteador estão, também, corretamente especificados.

Por fim, executando-se um *ping* do host 6 para o host 1, tem-se o seguinte:

```
mininet> h6 ping h1 -c2
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.2.1 icmp_seq=1 Destination Host Unreachable
From 10.0.2.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1004ms
pipe 2
```

*Figura 23: Ping de h6 para h1*

Na figura acima, conforme se percebe, os dois pacotes ping enviados não chegaram ao destino, pois ele está “inalcançável”. Isso se dá, pelo fato de que, embora as duas subredes estejam definidas pelo roteador e os IPs dele e do host 6 estejam corretamente designados, tem-se o desconhecimento dos demais hosts quanto ao *Gateway* que devem usar, uma vez que não têm como padrão nenhum roteador e, nenhum dos roteadores (além do r6) tem IPs configurados nem *Gateway* padrão para eles próprios. Assim, embora se consiga enviar pacotes de h1 para r6 e de h6 para r6, não é possível mandar de h1 para h6 nem ao contrário, pois o pacote não transitará entre redes.

Prosseguindo, o problema foi solucionado adicionando os devidos endereços IP a cada um dos roteadores e hosts de acordo com a *Figura 16*, adicionando o roteador 6 como *Gateway* padrão dos outros roteadores e cada um dos roteadores a cada um dos hosts (h1 → *Gateway r2, h2* → *Gateway r3*...), de modo a se obter o que se segue:

```
info( '*** Add hosts/stations\n')
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3/23', defaultRoute='10.0.1.21')
h4 = net.addHost('h4', cls=Host, ip='10.0.0.4/23', defaultRoute='10.0.1.21')
h5 = net.addHost('h5', cls=Host, ip='10.0.0.5/23', defaultRoute='10.0.1.21')
h6 = net.addHost('h6', cls=Host, ip='10.0.2.101/23', defaultRoute='10.0.2.21')
h2 = net.addHost('h2', cls=Host, ip='10.0.0.2/23', defaultRoute='10.0.1.21')
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1/23', defaultRoute='10.0.1.21')
```

*Figura 24: Código corrigido, parte 1*

```
r2.cmd("ifconfig r2-eth0 10.0.1.22/23")
r3.cmd("ifconfig r3-eth0 10.0.1.23/23")
r4.cmd("ifconfig r4-eth0 10.0.1.24/23")
r5.cmd("ifconfig r5-eth0 10.0.1.25/23")
r7.cmd("ifconfig r7-eth0 10.0.1.26/23")
r6.cmd("ifconfig r6-eth0 10.0.2.21/23")
r6.cmd("ifconfig r6-eth1 10.0.1.21/23")

r2.cmd("route add default gw 10.0.1.21")
r3.cmd("route add default gw 10.0.1.21")
r4.cmd("route add default gw 10.0.1.21")
r5.cmd("route add default gw 10.0.1.21")
r7.cmd("route add default gw 10.0.1.21")

h1.cmd("route add default gw 10.0.1.23")
h2.cmd("route add default gw 10.0.1.24")
h3.cmd("route add default gw 10.0.1.22")
h4.cmd("route add default gw 10.0.1.25")
h5.cmd("route add default gw 10.0.1.26")
h6.cmd("route add default gw 10.0.2.21")
```

*Figura 25: Código corrigido, parte 2*

Feito isso, tendo o problema resolvido, usando *tracepath* e capturando os pacotes com wireshark, encontrou-se o TTL entre hosts na topologia desta atividade e da primeira, conforme figuras abaixo:

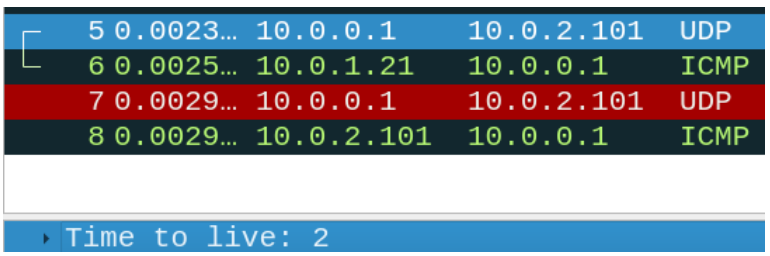


*Figura 26: TTL entre h1 e h6 (captura em h1), atividade 2*

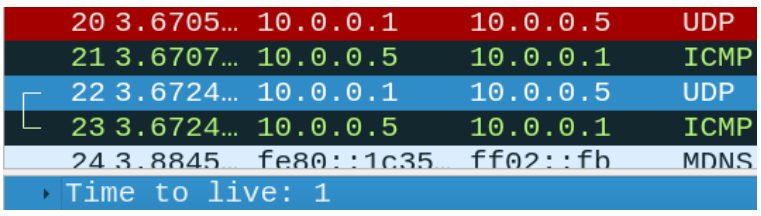


*Figura 27: TTL entre h1 e h5 (captura em h1), atividade 1*

Nas figuras acima, é possível observar que o TTL do pacote UDP de tracepath feito de h1 para h6 tem TTL maior que aquele feito de h1 a h5 na atividade 1 (topologia da atividade 1). Isso se dá, porque h1 e h5 estão na mesma rede, logo, o pacote vai diretamente de um host a outro, passando apenas por switches, enquanto que h1 e h6 estão em redes diferentes, portanto, o pacote precisa passar por um roteador para chegar ao destino. Desse modo, mesmo que fossem feitas ambas as capturas na topologia da atividade 2, o resultado seria o mesmo, para h1 → h6 ocorreriam dois hops (TTL = 2) e para h1 → h5 ocorreria um único hop (TTL = 1).

Por fim, conforme apresentado na comparação entre a topologia criada e a modelo, r6 é equivalente a r1 da topologia modelo, logo, foi analisada a tabela de roteamento de r6, do modo como se segue:

```
mininet> r6 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.254.0   U     0      0        0 r6-eth1
10.0.2.0        0.0.0.0         255.255.254.0   U     0      0        0 r6-eth0
```

*Figura 28: Tabela de roteamento r6 (r1 do modelo)*

A tabela acima, pode ser entendida facilmente. Isso porque, o roteador 6 é o que liga as redes 10.0.0.0/23 e 10.0.2.0/23, consequentemente, ele repassa pacotes de destino à rede 10.0.0.0 para a interface r6-eth1 e aqueles com destino a 10.0.2.0 para a interface r6-eth0, do modo como deve ser feito. Dessa forma, esse roteador atua como ponte entre as duas redes.

Quanto aos hosts, cabe ressaltar que cada um deles tem como *Gateway* padrão um roteador diferente e, todos esses roteadores, têm como *Gateway* padrão o roteador 6. Por esse motivo, são apresentadas abaixo as tabelas de um host (h1) e um roteador atrelado a ele (r3).

```
mininet> h1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.1.23       0.0.0.0         UG    0      0        0 h1-eth0
10.0.0.0        0.0.0.0         255.255.254.0   U     0      0        0 h1-eth0
```

*Figura 29: Tabela de roteamento h1*

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.1.21       0.0.0.0         UG    0      0        0 r3-eth0
10.0.0.0        0.0.0.0         255.255.254.0   U     0      0        0 r3-eth0
```

*Figura 30: Tabela de roteamento r3*

As tabelas acima são usadas para repassar os pacotes que saem de h1 para as interfaces corretas de acordo com o IP de destino do pacote. Para tanto, essas tabelas possuem uma relação entre conjunto de IPs, ou rede, e interface para a qual um pacote deve ser repassado. Nesse sentido, o uso de "-n" se dá como forma de esconder nomes de possíveis redes (faixas de IPs) para que se possa analisá-las por IP. Nesse caso, no experimento conduzido, isso foi útil para analisar quais IPs são repassados para o roteador que é *Gateway* padrão de um host ou mesmo de outro roteador, de modo a confirmar que ele repassa pacotes com qualquer IP de fato, ou seja, que ele atua de fato como *Gateway* padrão.

**Atividade 3)**

Nesta atividade, a topologia utilizada é outra, conforme as figura abaixo (destaque para a diferença de número entre os roteadores da topologia modelo comparado aos da topologia montada).

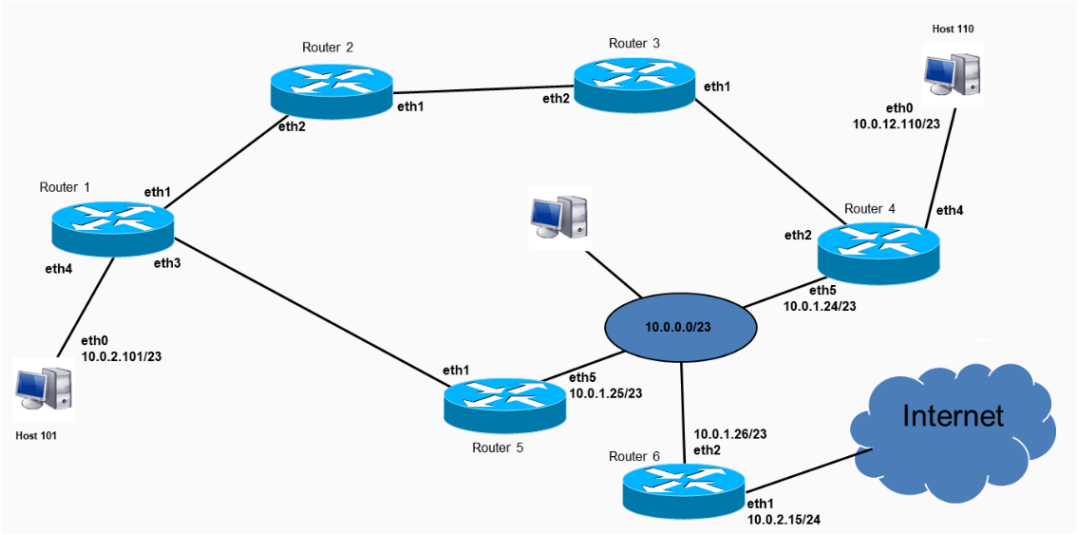


*Figura 31: Topologia modelo atividade 3*

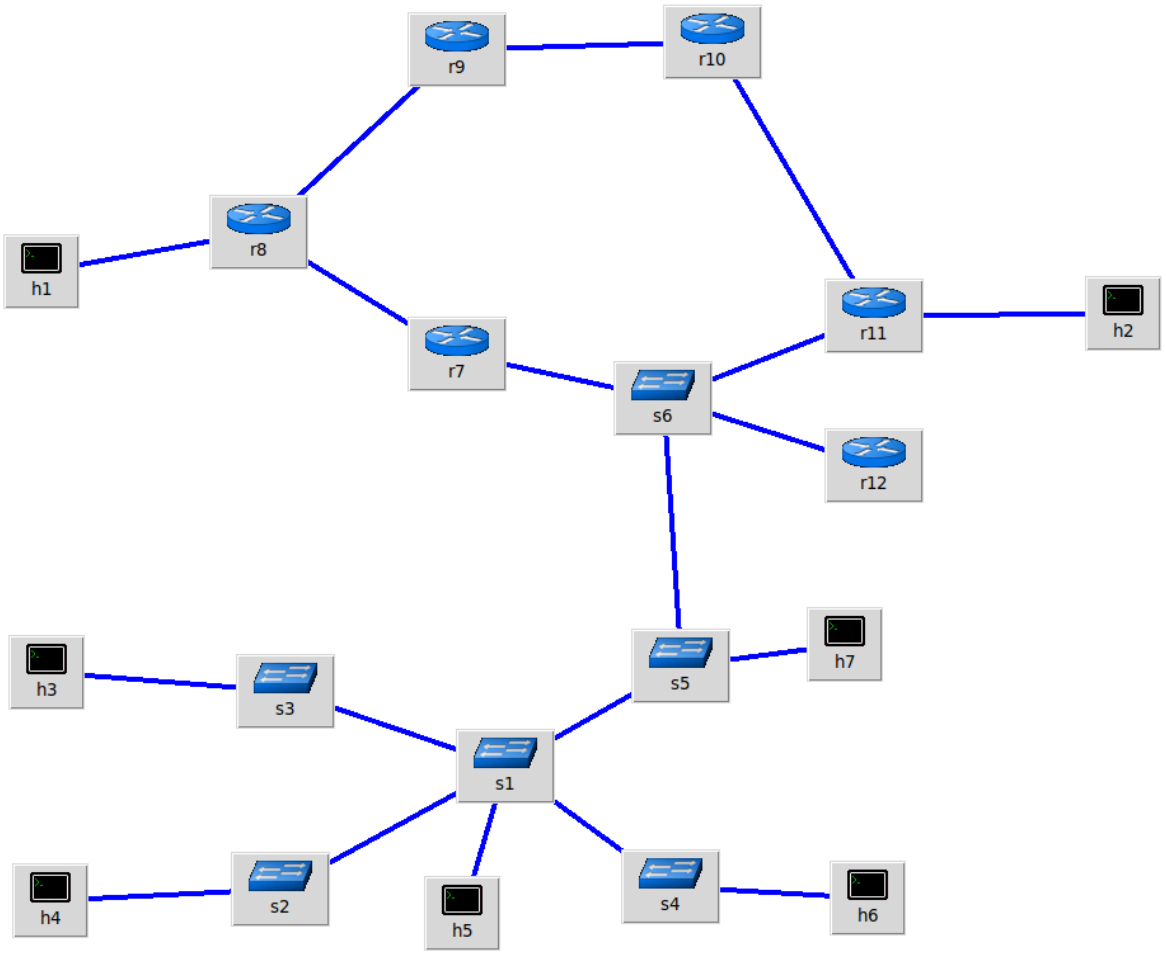


*Figura 32: Topologia montada atividade 3*

Após ajustar as rotas estáticas da rede, tem-se o seguinte código:

```
info( '*** Add hosts/stations\n')
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3/23', defaultRoute=None)
h4 = net.addHost('h4', cls=Host, ip='10.0.0.4/23', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.0.0.5/23', defaultRoute=None)
h1 = net.addHost('h1', cls=Host, ip='10.0.2.101/23', defaultRoute=None)
h7 = net.addHost('h7', cls=Host, ip='10.0.0.7/23', defaultRoute=None)
h6 = net.addHost('h6', cls=Host, ip='10.0.0.6/23', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.12.110/23', defaultRoute=None)

info( '*** Add links\n')
net.addLink(s1, s5)
net.addLink(s5, s6)
net.addLink(s6, r7, intfName2='r7-eth0')
net.addLink(r7, r8, intfName1='r7-eth1',intfName2='r8-eth0')
net.addLink(r8, r9, intfName1='r8-eth1',intfName2='r9-eth0')
net.addLink(r9, r10, intfName1='r9-eth1',intfName2='r10-eth0')
net.addLink(r10, r11, intfName1='r10-eth1',intfName2='r11-eth0')
net.addLink(r11, s6, intfName1='r11-eth1')
net.addLink(s6, r12, intfName2='r12-eth0')
net.addLink(r11, h2, intfName1='r11-eth2',intfName2='h2-eth0')
net.addLink(s3, h3, intfName2='h3-eth0')
net.addLink(s2, h4, intfName2='h4-eth0')
net.addLink(s1, h5, intfName2='h5-eth0')
net.addLink(s4, h6, intfName2='h6-eth0')
net.addLink(s5, h7, intfName2='h7-eth0')
net.addLink(r8, h1, intfName1='r8-eth2',intfName2='h1-eth0')
net.addLink(s1, s2)
net.addLink(s1, s3)
net.addLink(s1, s4)
```

*Figura 33: Código topologia 3, parte 1*

```
info( '*** Post configure nodes\n')

r7.cmd("ifconfig r7-eth0 10.0.1.25/23")
r7.cmd("ifconfig r7-eth1 10.0.4.25/23")
r8.cmd("ifconfig r8-eth0 10.0.4.21/23")
r8.cmd("ifconfig r8-eth1 10.0.6.21/23")
r8.cmd("ifconfig r8-eth2 10.0.2.21/23")
r9.cmd("ifconfig r9-eth0 10.0.6.29/23")
r9.cmd("ifconfig r9-eth1 10.0.8.29/23")
r10.cmd("ifconfig r10-eth0 10.0.8.30/23")
r10.cmd("ifconfig r10-eth1 10.0.10.30/23")
r11.cmd("ifconfig r11-eth0 10.0.10.24/23")
r11.cmd("ifconfig r11-eth1 10.0.1.24/23")
r11.cmd("ifconfig r11-eth2 10.0.12.24/23")
r12.cmd("ifconfig r12-eth0 10.0.1.32/23")

h1.cmd("route add default gw 10.0.2.21")
h2.cmd("route add default gw 10.0.12.24")
h3.cmd("route add default gw 10.0.1.25")
h4.cmd("route add default gw 10.0.1.25")
h5.cmd("route add default gw 10.0.1.25")
h6.cmd("route add default gw 10.0.1.24")
h7.cmd("route add default gw 10.0.1.24")
```

*Figura 34: Código topologia 3, parte 2*

A partir disso, obteve-se as tabelas de roteamento dos roteadores envolvidos em transmissões entre hosts, conforme as figuras abaixo:

```
mininet> r8 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.4.25       255.255.254.0   UG    0      0        0 r8-eth0
10.0.2.0        0.0.0.0         255.255.254.0   U     0      0        0 r8-eth2
10.0.4.0        0.0.0.0         255.255.254.0   U     0      0        0 r8-eth0
10.0.6.0        0.0.0.0         255.255.254.0   U     0      0        0 r8-eth1
10.0.12.0       10.0.4.25       255.255.254.0   UG    0      0        0 r8-eth0
```

*Figura 35: Tabela de roteamento de r8, roteador 1 do modelo (Figura 31)*

```
mininet> r7 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.254.0   U     0      0        0 r7-eth0
10.0.2.0        10.0.4.21       255.255.254.0   UG    0      0        0 r7-eth1
10.0.4.0        0.0.0.0         255.255.254.0   U     0      0        0 r7-eth1
10.0.12.0       10.0.1.24       255.255.254.0   UG    0      0        0 r7-eth0
```

*Figura 36: Tabela de roteamento de r7, roteador 5 do modelo (Figura 31)*

```
mininet> r11 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.254.0   U     0      0        0 r11-eth1
10.0.2.0        10.0.1.25       255.255.254.0   UG    0      0        0 r11-eth1
10.0.10.0       0.0.0.0         255.255.254.0   U     0      0        0 r11-eth0
10.0.12.0       0.0.0.0         255.255.254.0   U     0      0        0 r11-eth2
```

*Figura 37: Tabela de roteamento de r11, roteador 4 do modelo (Figura 31)*

Observando as figuras acima, pode-se notar que há caminhos suficientes para a comunicação entre todos os hosts e, como eles têm *Gateways* padrão, eles são todos "acessíveis" pelos roteadores, conforme se nota nas próximas figuras, que apresentam *ping's* e *tracepath's*.

```
mininet> h1 ping h2 -c2
PING 10.0.12.110 (10.0.12.110) 56(84) bytes of data.
64 bytes from 10.0.12.110: icmp_seq=1 ttl=61 time=0.166 ms
64 bytes from 10.0.12.110: icmp_seq=2 ttl=61 time=0.125 ms

--- 10.0.12.110 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1033ms
rtt min/avg/max/mdev = 0.125/0.145/0.166/0.020 ms
```

*Figura 38: Resposta de ping entre h1 e h2 (hosts 101 e 110, respectivamente, da Figura 31)*

```
mininet> h1 tracepath h2
 1?: [LOCALHOST]                      pmtu 1500
 1:  ???                                                    0.065ms
 1:  ???                                                    0.023ms
 2:  ???                                                    0.021ms
 3:  ???                                                    0.264ms
 4:  ???                                                    0.055ms reached
     Resume: pmtu 1500 hops 4 back 4
```

*Figura 39: Resposta de tracepath entre h1 e h2 (hosts 101 e 110, respectivamente, da Figura 31)*

```
mininet> h1 ping h5 -c2
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=62 time=0.602 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=62 time=0.121 ms

--- 10.0.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.121/0.361/0.602/0.240 ms
```

*Figura 40: Resposta de ping entre h1 e h5 (hosts 101 e de 10.0.0.0, respectivamente, da Figura 31)*

```
mininet> h1 tracepath h5
 1?: [LOCALHOST]                      pmtu 1500
 1:  ???                                                    0.086ms
 1:  ???                                                    0.020ms
 2:  ???                                                    0.023ms
 3:  ???                                                    0.396ms reached
     Resume: pmtu 1500 hops 3 back 3
```

*Figura 41: Resposta de tracepath entre h1 e h5 (hosts 101 e de 10.0.0.0, respectivamente, da Figura 31)*

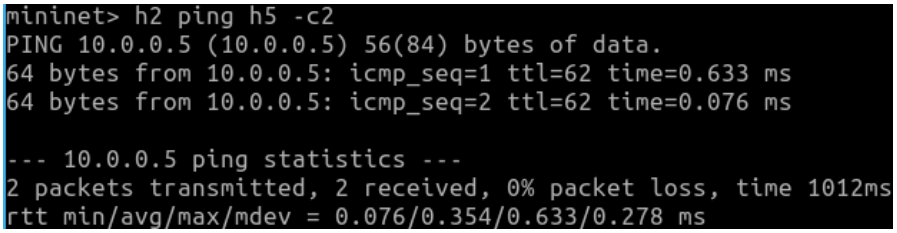


*Figura 42: Resposta de ping entre h2 e h5 (hosts 110 e de 10.0.0.0, respectivamente, da Figura 31)*

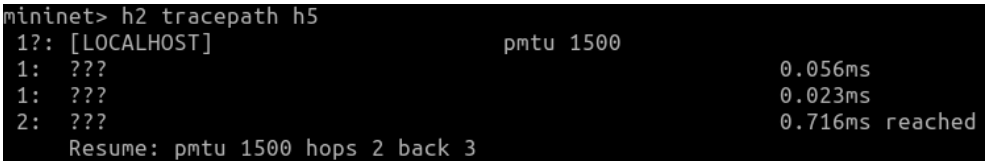


*Figura 43: Resposta de tracepath entre h2 e h5 (hosts 110 e de 10.0.0.0, respectivamente, da Figura 31)*

Nos comandos ping, pode-se observar um TTL muito alto quando comparado ao número de passos do *tracepath*. Isso porque, nos resultados de *tracepath* obtidos, o número de passos mais alto foi de 4, já que o pacote passou pelos roteadores 8, 7 e 11 para chegar de h1 em h2. Enquanto que, no *ping*, o TTL é muito alto, visando a garantir a chegada do pacote ping ao destino, embora no caso presente esse TTL seja muito superestimado.

Como complemento, vale ressaltar os caminhos em cada um dos *ping* e *tracepath*:
1°) h1 → h2: Sai de h1, passa por r8, segue para r7 que o repassa para r11 e, por fim, segue para h2;
2°) h1 → h5: Sai de h1, passa por r8, segue para r7 que o repassa diretamente para h5;
3°) h2 → h5: Sai de h2, passa por r11, segue diretamente h5.

## Conclusão)

Finalmente, pode-se afirmar que os objetivos deste experimento foram satisfeitos, à medida que se pôde usar o *miniedit* para construir as topologias usadas e, foi possível se aprender mais profundamente a respeito dos *scripts* geradores das redes usadas. Além disso, foi estudado ainda formas de se configurar as tabelas de roteamento dos roteadores e hosts manualmente, o que permitiu, também, aprofundamento dos conhecimentos relacionados a repasse de pacotes, TTL e números de saltos.