EA080: Laboratório 01

Vinícius Esperança Mantovani,
247395.

Introdução)

Neste experimento, foram realizadas atividades no *mininet-wifi* com o objetivo de familiarização e reconhecimento das ferramentas de criação, teste e estudo de topologias e redes. Para tanto, foram utilizados o programa *wireshark*, os comandos *ping* e *iperf*, que também foram alvo de análise e estudo e, outras ferramentas do *mininet*, conforme se nota no decorrer deste documento.

Atividade 1)

Nesta atividade, do modo como é apresentado nas *Figuras 1* e *2* e na tabela de nós abaixo, foi executado inicialmente o comando "*sudo mn --mac*" para a geração de uma rede com dois hosts e um switch. Partindo disso, foram executados os comandos "*nodes*", "*intfs*" e "*pingall*", responsáveis, respectivamente, por apresentar os nós da rede, apresentar as interfaces dos nós e, finalmente, fazer um ping de cada host para os demais hosts, com 0 pacotes *dropados* (aqui há apenas dois *hosts*, então o ping é de um para o outro apenas). Por fim, executou-se um "*xterm*" para cada nó da rede, abrindo os consoles de cada um deles (*Figura 3*).

```
*** 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:
```

Figura 1

```
mininet-wifi> nodes
available nodes are:
c0 h1 h2 s1
mininet-wifi> intfs
h1: h1-eth0
h2: h2-eth0
s1: lo,s1-eth1,s1-eth2
c0:
mininet-wifi> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet-wifi> xterm h1
mininet-wifi> xterm h2
mininet-wifi> xterm s1
```

Figura 2

Nó	Interfaces
h1	h1-eth0
h2	h2-eth0
s1	lo, s1-eth1, s1-eth2
c0 (Controlador)	_____

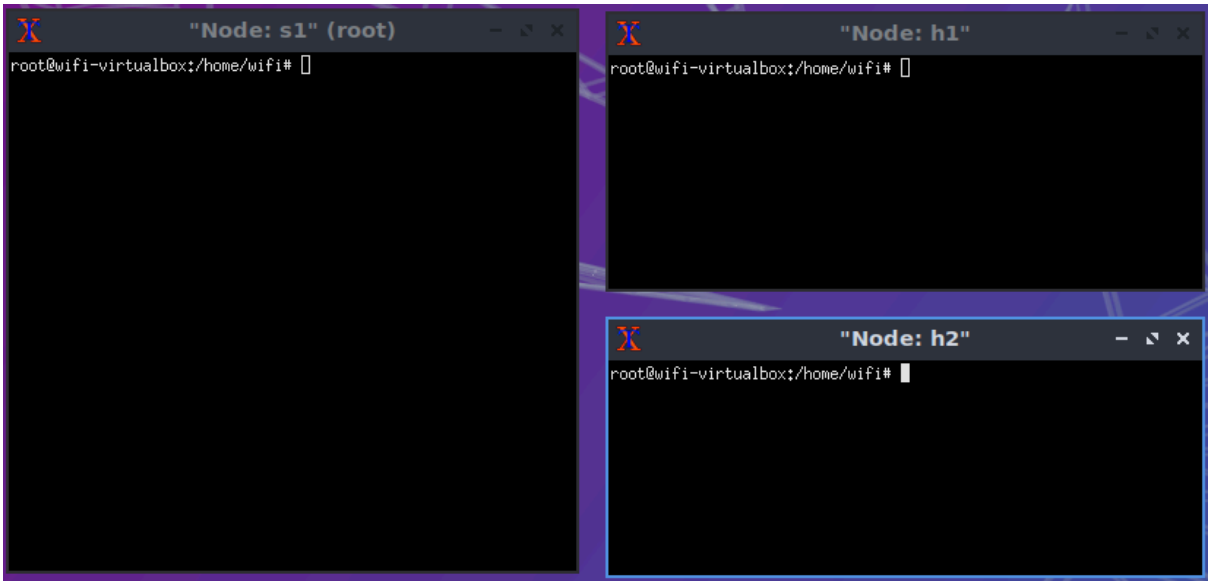

Figura 3

Antes da continuação do procedimento, vale destacar que o controlador c0 não é levado em conta nas análises feitas nesta atividade. Tendo isso em vista, na sequência, usando-se o terminal de *h2*, foi aberta a ferramenta "*Wireshark*" no sistema deste host, por meio do uso do comando "sudo Wireshark". Este processo e o "*Wireshark*" aberto são apresentados respectivamente nas *Figuras 4 e 5* exibida abaixo.

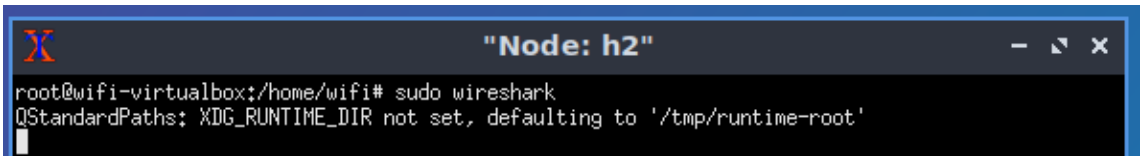

Figura 4

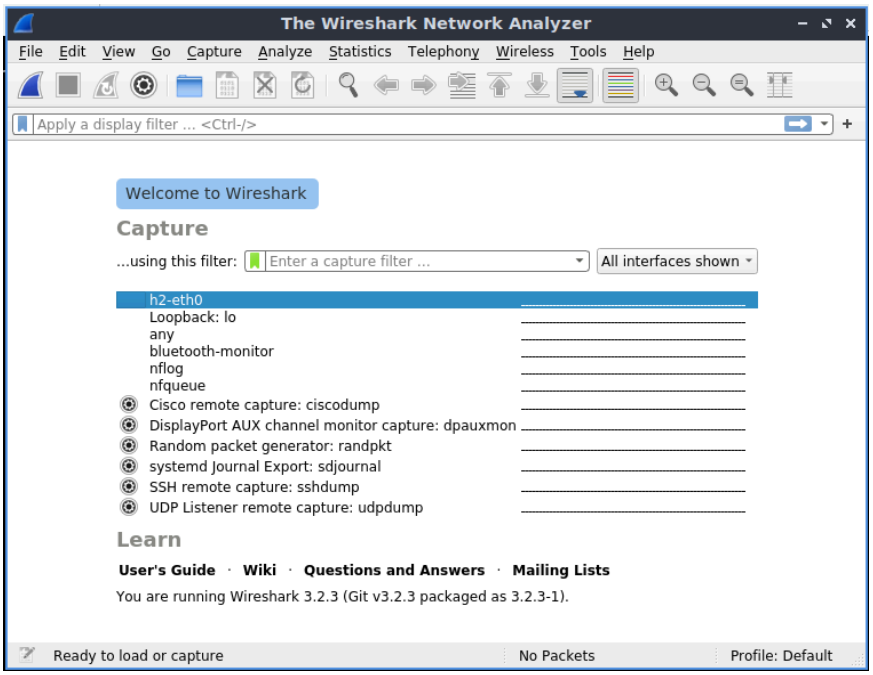

Figura 5

Usando o "*Wireshark*" e executando um ping para *h2*, tem-se o que se apresenta na Figura 6 abaixo, na qual se pode notar um broadcast *ARP* partindo de 10.0.0.1 (*h1*) para encontrar o *MAC* de *h2* e, na sequência disso, vê-se uma resposta *ARP* de *h2* direcionada explicitamente ao *MAC* de *h1*. Finalmente, o *h1* envia o *ping request* e o *h2* retorna um *ping reply*, conforme esperado.

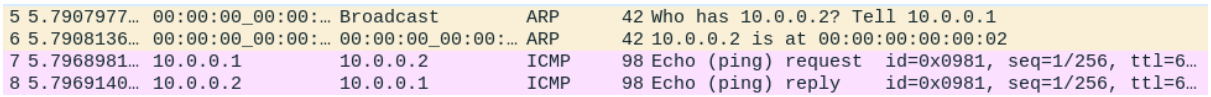

Figura 6

Por fim, executando-se os comando "*ifconfig*" nos terminais de cada um dos nós, conforme as *Figuras 7, 8 e 9*, foram encontrados os *IPs* e *MACs* de *h1* e *h2* e, os *MAC* e IPV6 de *s0* (usados apenas para identificação do switch, uma vez que este deve ser tranparente na rede, logo, não há pacotes que o contenham como destino nem como fonte). Seguindo, por meio dos comandos "*arp*" e "*route*", conforme as *Figuras 10, 11, 12* e *13* exibem, foram encontradas as tabelas *ARP* e de roteamento, respectivamente na sequência das figuras citadas, de *h1 e h2*.

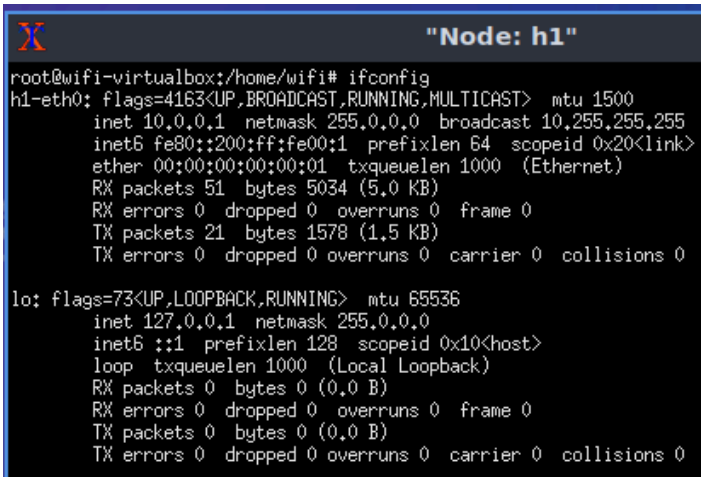

Figura 7

$IPV4_{h1} \rightarrow 10.0.0.1;$ $IPV6_{h1} \rightarrow fe80::200:ff:fe00:1;$ $MAC_{h1} \rightarrow 00:00:00:00:00:01;$

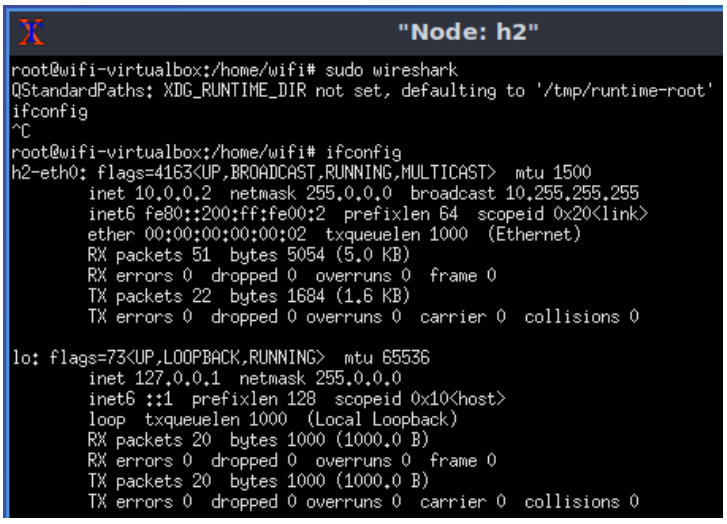

Figura 8

$IPV4_{h2}$ → 10.0.0.2; $IPV6_{h2}$ → fe80::200:ff:fe00:2; MAC_{h2} → 00:00:00:00:00:02;

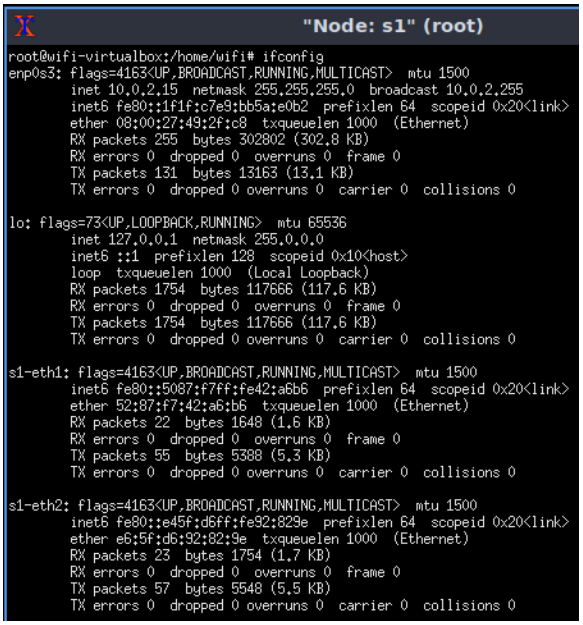

Figura 9

$IPV6_{s1\text{-}eth1}$ → fe80::5087:f7ff:fe42:a6b6; $MAC_{s1\text{-}eth1}$ → 52:87:f7:42:a6:b6;
$IPV6_{s1\text{-}eth2}$ → fe80::e45f:d6ff:fe92:829e; $MAC_{s1\text{-}eth2}$ → e6:5f:d6:92:82:9e;

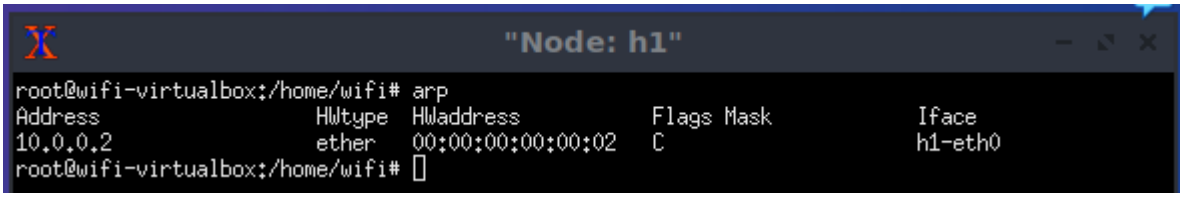

Figura 10

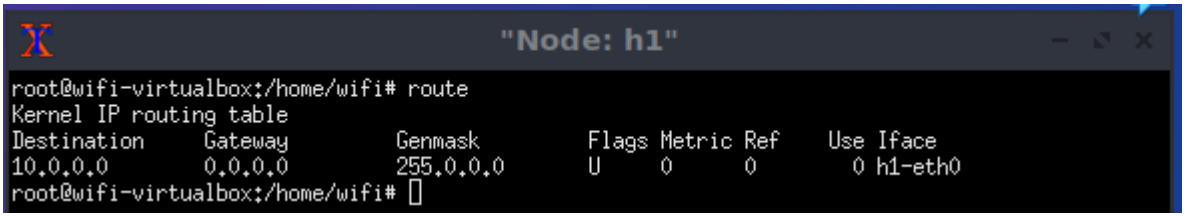

Figura 11

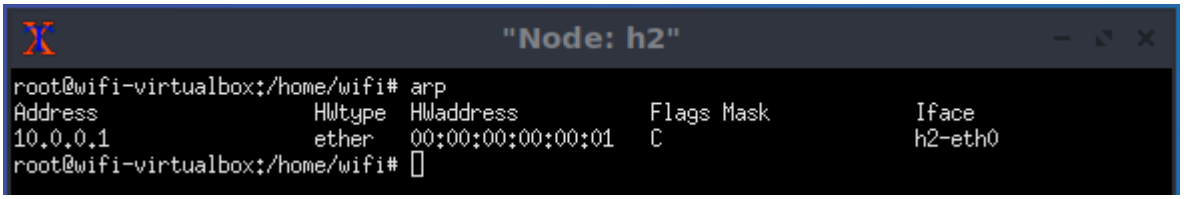

Figura 12

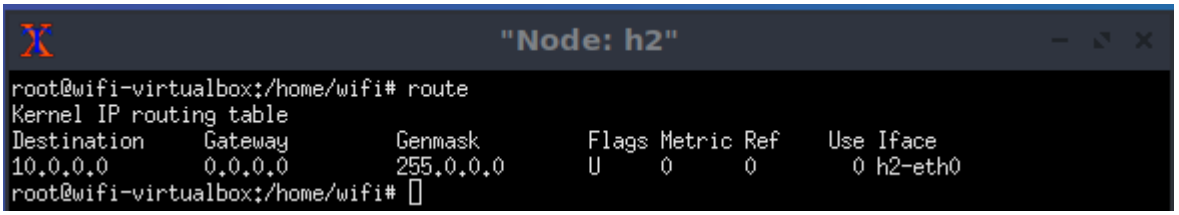

Figura 13

Atividade 2)

Nesta atividade, ao executar o *script python* "*pratica-1-II.py*" clonado do git, obteve-se o que se expõe na *Figura 14* abaixo. Nela, pode-se ver a geração de uma topologia conforme a da *Figura 15*, com bolas representando switches e quadrados representando hosts. Daí, nota-se que o *host 4* não está conectado a nenhum switch, havendo uma tentativa de *link* entre ele e o *host 3* apenas. Por esse motivo, ao usar o comando "*pingall*", ainda conforme a *Figura 14*, nota-se que nenhum *host* consegue se comunicar com o 4 e, consequentemente, nem ele consegue se comunicar com os demais.

```
wifi@wifi-virtualbox:~/EA080-2S2021/lab1$ sudo python pratica-1-II.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s2) (h2, s2) (h3, s3) (h4, h3) (s1, s2) (s1, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X
h2 -> h1 h3 X
h3 -> h1 h2 X
h4 -> X X X
*** Results: 50% dropped (6/12 received)
```

Figura 14

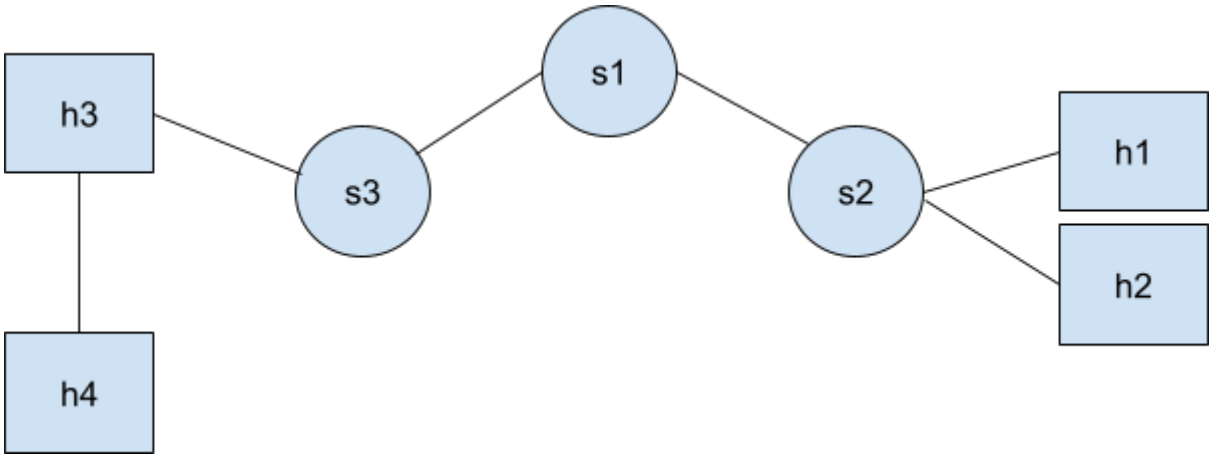

Figura 15

Para corrigir o problema destacado anteriormente, seria necessário criar um *link* de *h4* com algum *switch*. Por isso, foi criado um *link* de *h4* com *s3* que substituiu o *link* de *h4* com *h3*.

```
topo.addLink('h4', 'h3')    ⟶    topo.addLink('h4', 's3')
```

```
wifi@wifi-virtualbox:~/EA080-2S2021/lab1$ sudo python pratica-1-II.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s2) (h2, s2) (h3, s3) (h4, s3) (s1, s2) (s1, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figura 16

Finalmente, após a geração da nova topologia na *Figura 16* acima, que apresenta também um "*pingall*" sem problemas de conexão, tem-se o que se esquematiza abaixo:

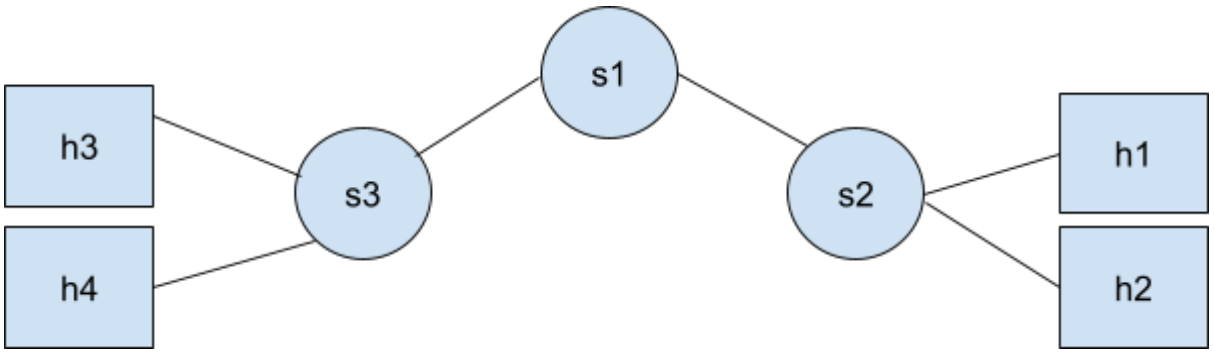

Figura 17

Atividade 3)

Nesta atividade, utilizou-se um *script python* para criação de uma topologia igual à da *Figura 17*, mas com alguns parâmetros passados ao enlace entre os *switches 1* e *2*. Neste contexto, sob análise superficial, nota-se que os parâmetros dos enlaces no script são passados na função de criação do enlace, do modo como se vê abaixo:

```
topo.addLink('s1','s2',bw=100,delay='100ms',loss=10)
```

Com base no que se expõe acima, sabe-se que a taxa de perda no enlace entre o *switch* 1 e o 2 é de 10%, que o delay desse enlace é de 100ms e que a largura de banda nesse enlace é de 100Mbps. Por esse motivo, ao se fazer *ping* de *h1* para *h4*, tem-se tempo de resposta médio próximo de 200ms até a recepção da resposta (ida e volta de sinal) e, uma taxa de perda de pacote que deve convergir para 19% (100 - 90%*90%*100) com um número infinito de *pings* , enquanto que, ao *"pingar"* de *h1* para *h2*, não há perda, pois os pacotes não passam por tal enlace. Sabendo disso, pode-se observar tais expectativas serem atendidas respectivamente nas *Figuras 18* e *19* a seguir, que apresentam o resultados dos comandos "*h1 ping -c100 h2*" e "*h1 ping -c100 h4*".

```
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101325ms
rtt min/avg/max/mdev = 0.047/0.229/8.967/1.040 ms
```

Figura 18

```
--- 10.0.0.4 ping statistics ---
100 packets transmitted, 79 received, 21% packet loss, time 99355ms
rtt min/avg/max/mdev = 200.144/207.416/497.521/42.474 ms
```

Figura 19

Dando continuidade ao experimento, para testar a largura de banda entre os enlaces que levam pacotes de *h1* para *h2* e para *h4*, foram usados os comandos "*iperf h1 h2*" e "*iperf h1 h4*", de maneira a se obter os resultados das *Figuras 20* e *21*, que mostram a largura de banda no caminho de ida e de volta respectivamente (*h1* → *h2, h2* → *h1* ; *h1* → *h4, h4* → *h1*). Nelas, nota-se que há uma velocidade extremamente alta entre os *hosts h1* e *h2*, conectados diretamente pelo *switch* 2, enquanto que a conexão entre *h1* e *h4* tem uma largura de banda muito menor. Isso se dá, devido ao fato de que, conforme explicado na questão do *ping* tratada anteriormente, entre *h1* e *h2* não há perdas de pacotes, então a velocidade da transmissão se dá em toda sua capacidade, com o *TCP* dobrando sua janela e enviando, por consequência, o dobro de pacotes a cada envio, enquanto que entre *h1* e *h4*, a velocidade é extremamente reduzida por conta dos 10% de perda de pacotes entre os *switches* 1 e 2. Isso porque, a cada pacote perdido, tem-se o retorno ao "*slow start*" do *TCP*, que volta ao início do processo, transmitindo apenas um pacote e, incrementando esse número a cada transmissão em 2 vezes (1 → 2 → 4 → 8…) até chegar no ponto de "*congestion avoidance*", em que o incremento é linear. Com isso, a transmissão de pacotes fica continuamente sendo reduzida e incrementada, o que ocasiona uma velocidade muito menor de transmissão quando comparada àquela entre *h1* e *h2*, em que a janela somente aumenta.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['33.8 Gbits/sec', '33.9 Gbits/sec']
```

Figura 20

```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['202 Kbits/sec', '478 Kbits/sec']
```

Figura 21

Conforme discutido anteriormente, os resultados obtidos são facilmente entendidos quando se leva em conta as perdas de pacotes implementada pelo *script* no enlace entre os *switches* 1 e 2. Sendo este, um fator determinante tanto para o número de pacotes perdidos apresentado nos comandos *ping* como para a pequena largura de banda medida por meio do comando *iperf* entre *h1* e *h4*.

Atividade 4)

Inicialmente, alterou-se a função "*topo*" do *script* da atividade anterior para que gerasse a topologia desejada (*Figura 22*). Com isso, obteve-se a função "*topo*" apresentada nas *Figuras 23* e *24*, a qual já contém os parâmetros desejados de largura de banda, *delay* e perda de pacotes desejados apresentados abaixo:

- Core para Distribuição (*d1, d2*): 10 Gbps, 1 ms;
- Distribuição para Acesso (*a1, a2, a3, a4*): 1 Gbps, 3 ms;
- Acesso para Hosts: 100 Mbps, 5 ms;
- Taxa de erro de 15% entre *h8* e *a4*.

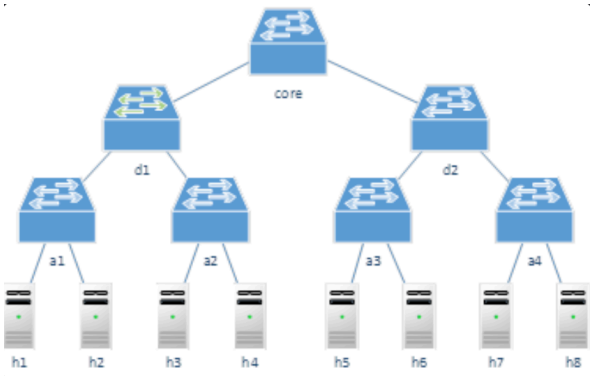

Figura 22

```
def createTopo():
    topo=Topo()
    #Create Nodes
    topo.addHost("h1")
    topo.addHost("h2")
    topo.addHost("h3")
    topo.addHost("h4")
    topo.addHost("h5")
    topo.addHost("h6")
    topo.addHost("h7")
    topo.addHost("h8")
    topo.addSwitch('a1')
    topo.addSwitch('a2')
    topo.addSwitch('a3')
    topo.addSwitch('a4')
    topo.addSwitch('d1')
    topo.addSwitch('d2')
    topo.addSwitch('core1')
```

Figura 23

```
#Create links
topo.addLink('core1','d1',bw=10000,delay='1ms')
topo.addLink('core1','d2',bw=10000,delay='1ms')
topo.addLink('d1','a1',bw=1000,delay='3ms')
topo.addLink('d1','a2',bw=1000,delay='3ms')
topo.addLink('d2','a3',bw=1000,delay='3ms')
topo.addLink('d2','a4',bw=1000,delay='3ms')
topo.addLink('a1','h1',bw=100,delay='5ms')
topo.addLink('a1','h2',bw=100,delay='5ms')
topo.addLink('a2','h3',bw=100,delay='5ms')
topo.addLink('a2','h4',bw=100,delay='5ms')
topo.addLink('a3','h5',bw=100,delay='5ms')
topo.addLink('a3','h6',bw=100,delay='5ms')
topo.addLink('a4','h7',bw=100,delay='5ms')
topo.addLink('a4','h8',bw=100,delay='5ms',loss=10
return topo
```

Figura 24

Para medir a perda de pacotes para *h8* poderia ser usado o comando *ping* de *h7* para o host em questão, uma vez que só há perda de pacotes no enlace entre *a4* e *h8*, logo, teríamos, com tal experimento, um resultado que deveria convergir para aproximadamente 27,75% (100 - 85%*85%*100), por conta das perdas na ida e na volta do sinal, com um número de *pings* tendendo ao infinito. No entanto, por obviedade, não seria possível executar o experimento com infinitos *pings*, portanto, optando-se por 200 *pings*, tem-se o que se mostra na *Figura 25*, que é seguida por uma explicação de seus resultados.

```
--- 10.0.0.8 ping statistics ---
200 packets transmitted, 148 received, 26% packet loss, time 200093ms
rtt min/avg/max/mdev = 20.074/20.494/58.706/3.190 ms
```

Figura 25

De acordo com a figura acima, percebe-se que o número de pacotes perdidos representa 26% dos pacotes enviados em ping. Esse percentual é bastante próximo daquele esperado teoricamente para um número infinito de pacotes, 27,75%, porém, não é idêntico a ele. Isso se dá, por conta de a taxa definida ser probabilística, ou seja, os pacotes terem uma probabilidade de 15% de serem perdidos, o que faz com que o número de pacotes perdidos, a não para infinitos testes, não represente necessariamente 15% de perda, mas sim um número próximo disso. Desse modo, neste caso apresentado, esse percentual aproximado foi de aproximadamente 13,98%, valor consideravelmente próximo a 15%.

Seguindo, usando o comando *iperf* para medir as larguras de banda entre *h1* e *h2/h3/h4*, temos os resultados das *Figura 26*. Em tal figura, podemos perceber que, conforme as expectativas, as velocidades de upload e download de *h1* em conexão com os três *hosts* em questão foram muito semelhantes, o que se deve ao fato de a largura de banda limitante ser a dos enlaces que ligam os *switches a1, a2, a3* e *a4* aos *hosts.* Além disso, ainda por causa dessa limitação de tais enlaces, os valores todos estão próximos de 100 Mbps, valor estabelecido para tais enlaces destacados. Vale destacar ainda, que as diferenças entre os valores de *upload* e *download* podem existir, talvez, por conta do modo como se considera a ida e vinda (a partir de quando se consideram os pacotes) de pacotes pelo canal, pois calculando a média entre os valores da largura de banda de ida e vinda, tem-se um valor mais próximo de 100 Mbps.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['94.4 Mbits/sec', '110 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['92.8 Mbits/sec', '108 Mbits/sec']
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['92.4 Mbits/sec', '109 Mbits/sec']
```

Figura 26

Atividade 5)

Nesta atividade, foram alterados alguns valores do *script* usado na atividade anterior para que fossem atendidas as seguintes requisições:

• Core para Distribuição (d1, d2): 1 Gbps, 2 ms
• Distribuição para Acesso (a1, a2, a3, a4): 100 Mbps, 2 ms
• Acesso para Hosts: 10 Mbps, 2 ms

Do modo citado, obteve-se, portanto, o que se segue na *Figura 27* como *links* na função "*topo*" do *script*:

```
#Create links
topo.addLink('core1','d1',bw=1000,delay='2ms')
topo.addLink('core1','d2',bw=1000,delay='2ms')
topo.addLink('d1','a1',bw=100,delay='2ms')
topo.addLink('d1','a2',bw=100,delay='2ms')
topo.addLink('d2','a3',bw=100,delay='2ms')
topo.addLink('d2','a4',bw=100,delay='2ms')
topo.addLink('a1','h1',bw=10,delay='2ms')
topo.addLink('a1','h2',bw=10,delay='2ms')
topo.addLink('a2','h3',bw=10,delay='2ms')
topo.addLink('a2','h4',bw=10,delay='2ms')
topo.addLink('a3','h5',bw=10,delay='2ms')
topo.addLink('a3','h6',bw=10,delay='2ms')
topo.addLink('a4','h7',bw=10,delay='2ms')
topo.addLink('a4','h8',bw=10,delay='2ms',loss=15)
```

Figura 27

Em sequência ao experimento, foi executado novamente o comando *iperf* para os mesmo *hosts* da atividade anterior, conforme resultados a seguir (*Figura 28*):

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.56 Mbits/sec', '11.7 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.52 Mbits/sec', '11.7 Mbits/sec']
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['9.55 Mbits/sec', '11.8 Mbits/sec']
```

Figura 28

Mais uma vez, do mesmo modo como na atividade anterior, em que os enlaces tinham propriedades diferentes, os valores dos três testes estão muito próximos uns dos outros e, os limitante na largura de banda entre *h1* e os demais foram, novamente, os enlaces de acesso para os hosts, que têm todos uma banda de 10 Mbps. Além disso, vale ressaltar que o "problema" da diferença entre banda de *upload* e *download* de *h1* estão, do mesmo modo que anteriormente, diferentes entre si, sendo que, na média, dão um valor próximo de 10 Mbps (banda determinada para tais enlaces por meio do *script*). Finalmente, comparando os valores de cada um destes três testes com os três da atividade anterior, nota-se que, por conta do limite mais baixo destes da atividade 5, os anteriores (atividade 4) tiveram melhor desempenho quanto à largura de banda.

Além do exposto acima, tem-se o resultado de um comando *ping* com 200 repetições de *h7* para *h8*, como se fez na atividade anterior. Neste experimento, conforme esperado, mais uma vez, o

percentual de perdas foi próximo de 27,75% (32%), porém, o tempo médio de *ping* foi menor. Isso ocorreu, pois a latência nos enlaces que ligam *h7* a *h8* é menor nesta atividade que na anterior. Sendo assim, embora a largura de banda nesta última topologia seja menor que na anterior, no geral, a latência nesta é menor que na anterior (em todos os enlaces e, portanto, no *ping* entre quaisquer dois *hosts*).

Ademais, realizou-se uma captura de pacotes com o intuito de compreender as métricas usadas para a análise anterior a respeito do desempenho da rede. Para tanto, foi utilizado o *wireshark*, de forma a capturar os pacotes enviados em um *iperf* entre *h1* e *h2*, sendo o *wireshark* aberto no sistema de *h1*. Desse modo, pode-se notar um bombardeio de dados causado pelo *iperf* e, em pesquisa breve pela internet, é possível reconhecer que o *iperf* opera enviando muitos pacotes de dados para calcular, durante um período conhecido, a quantia de dados enviados e, dessa maneira, calcular dados/tempo = largura de banda. Logo, conforme a *Figura 29* mostra, é possível confirmar o modo de operação do *iperf* e a sua precisão por meio da ferramenta de estatísticas de entrada e saída do *wireshark*.

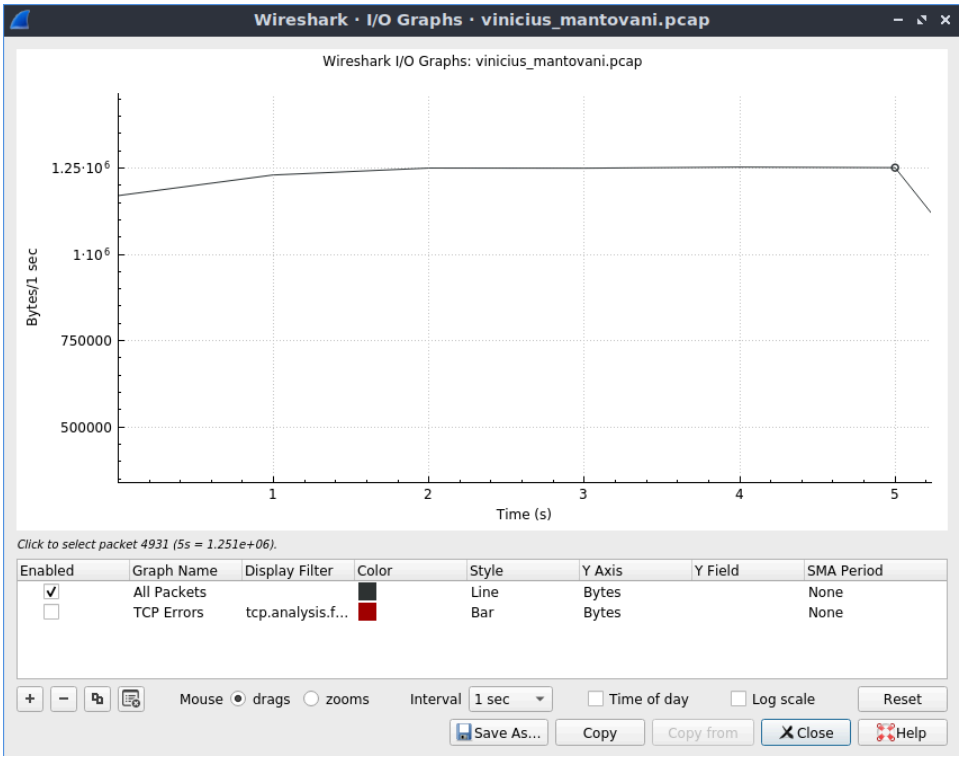

Figura 29

Na figura acima, tem-se o gráfico de bytes de entrada e saída em *h1* acumulados durante um segundo (amostrados em intervalos de 1 segundo) e, pode-se perceber neste gráfico que o número de bytes estabiliza em aproximadamente $1{,}25*10^6$. Logo, como sabemos que o intervalo de tempo entre amostras é de 1 segundo, pode-se afirmar que tem-se no máximo $1{,}25*10^6$ bytes por segundo em transmissão e, por consequência, que a largura de banda é de $1{,}25*10^6*8$ = 10000000 bps = 10 Mbps. Dessa maneira, fica simples compreender o funcionamento do *iperf*, que satura a rede com dados para calcular qual a maior quantia de bits transferidos dentro de um período determinado de tempo e, na sequência, calcular o número de bits por segundo de largura de banda entre os *hosts* envolvidos.

Além do *iperf*, é possível e mais simples analisar o funcionamento do comando *ping* (*Figura 30*). Para tanto, basta capturar os pacotes de *request* e *reply* envolvidos em tal comando. Com isso feito, é possível perceber que o comando é simples e funciona com a ida do pacote *request* de um *host* emissor até o *host* receptor (no caso, respectivamente *h1* e *h2*) e, a volta de um pacote do *host* receptor ao *host* emissor do *ping*. Dessa forma, trabalhando em camada de rede com uso do protocolo *ICMP*, o *ping* usa o tempo da ida do pacote *request* até a chegada do *reply* para apresentar uma o *RTT* entre os *hosts* envolvidos. Por fim, vale ressaltar que, para calcular o percentual de pacotes perdidos no caso de mais de um *ping*, o comando divide o número de pacotes retornados e divide pelo número de pacotes enviados.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x6c23, seq=1/256, ttl=6…
2	0.0120663…	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x6c23, seq=1/256, ttl=6…

Figura 30

Finalmente, para se tomar conhecimento a respeito do programa *miniedit*, foi executado o programa e, já de início, observou-se a interface exibida na *Figura 31*, na qual é possível listar os seguintes dispositivos disponíveis para criação de topologia:

- Host
- Estação
- Switch
- AP → Access Point
- Legacy Switch
- Legacy Router
- NetLink
- Controller

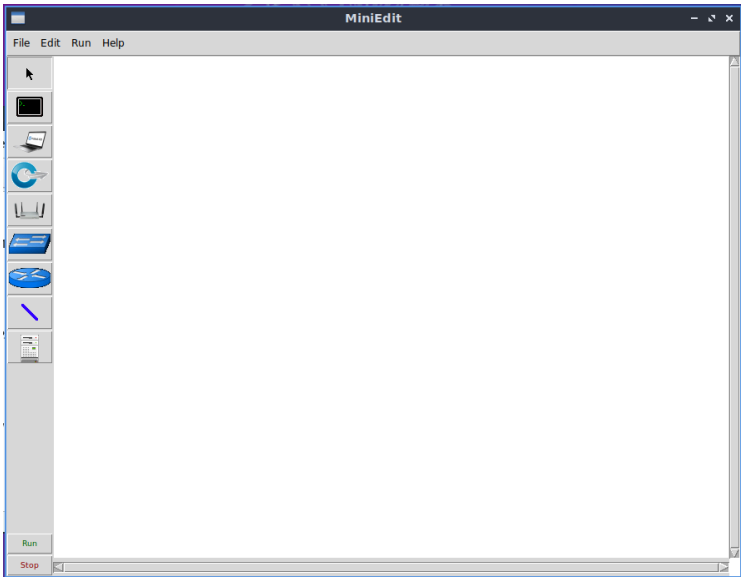

Figura 31

Além disso, criando uma topologia simples (*Figura 35*), que envolve apenas um roteador, dois switches e quatro hosts, nota-se a possibilidade de se definir parâmetros como:

- **Para Hosts:**
  - Aqui, é possível notar que as possibilidades de manipulação dos *hosts* é bastante ampla, o que é um fator importante para o estudo de topologias desejadas.

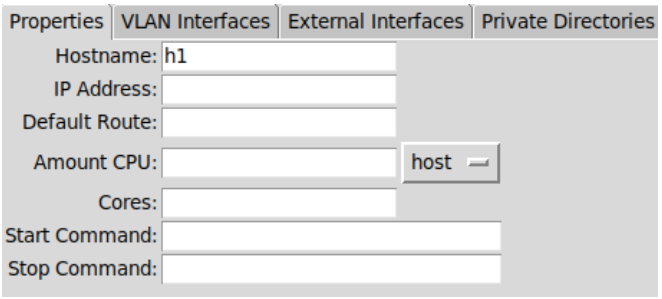

Figura 32

- **Para Links:**
  - Do mesmo modo que se tem uma vasta possibilidade de manipulação dos *hosts*, tem-se também uma vasta gama de parâmetros para os *links*, podendo ser definidas questões de largura de banda, atraso no enlace, perda percentual de pacotes, limite do tamanho de filas, alteração no atraso (*Jitter*) e aumento de velocidade no enlace. Isso, além do tipo de enlace, que pode ser escolhido entre as opções da *Figura 34*.

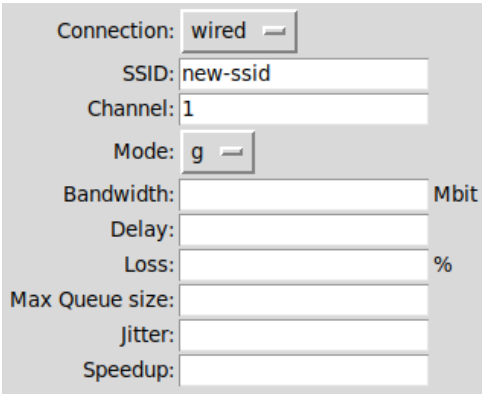

Figura 33

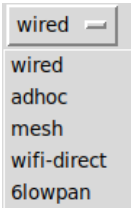

Figura 34

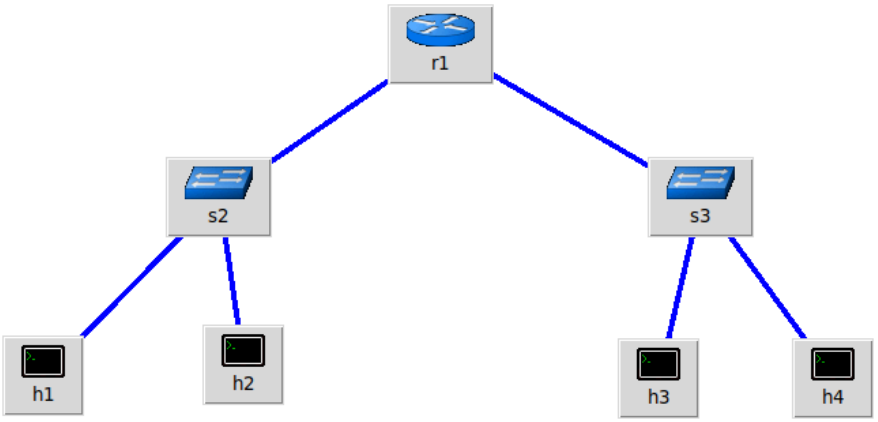

Figura 35

Conclusão)

Em suma, foi possível aprender a respeito das ferramentas e comandos utilizados no decorrer deste experimento. Tendo sido, portanto, o objetivo por trás deste relatório muito bem sucedido, de forma que foi-se obtido conhecimento a respeito do *wireshark*, e da criação de topologias de rede tanto por *python,* no contexto em questão, como pelo programa *miniedit*, Além disso, foram desenvolvidos, também, conhecimentos a respeito dos comandos de teste *iperf* e *ping* e, de comandos básicos do *mininet* utilizados.