Relatório do EP3
MAC0352 – Redes de Computadores e Sistemas Distribuídos – 2/2019

Julio Kenji Ueda (9298281), Ricardo Akira Tanaka (9778856)

1 Passo 0

Na definição do protocolo OpenFlow, o que um switch faz toda vez que ele recebe um pacote que ele nunca recebeu antes?

Um pacote nunca recebido não possui um fluxo definido, então o mesmo é enviado à controladora do switch. Esta pode adicionar um fluxo ao switch para tratar futuros pacotes semelhantes ou ignorar o pacote (drop).

2 Passo 2

Com o acesso à Internet funcionando em sua rede local, instale na VM o programa `traceroute` usando `sudo apt install traceroute` e escreva abaixo a saída do comando `sudo traceroute -I www.inria.fr`. Pela saída do comando, a partir de qual salto os pacotes alcançaram um roteador na Europa? Como você chegou a essa conclusão?

```
<traceroute to www.inria.fr (128.93.162.84), 30 hops max, 60 byte packets
 1  10.0.2.2 (10.0.2.2)  1.451 ms  0.892 ms  0.159 ms
 2  * * *
 3  * * *
 4  core-cce.uspnet.usp.br (143.107.255.225)  2.892 ms  3.381 ms  3.472 ms
 5  border1.uspnet.usp.br (143.107.251.29)  3.094 ms  2.877 ms  2.415 ms
 6  usp-sp.bkb.rnp.br (200.143.255.113)  3.152 ms  3.843 ms  3.572 ms
 7  br-rnp.redclara.net (200.0.204.213)  110.882 ms  111.045 ms  110.601 ms
 8  us-br.redclara.net (200.0.204.9)  111.203 ms  111.132 ms  111.039 ms
 9  redclara-gw.par.fr.geant.net (62.40.125.168)  212.854 ms  212.793 ms  212.873 ms
10  renater-lb1-gw.mx1.par.fr.geant.net (62.40.124.70)  212.777 ms  233.930 ms  233.771 ms
11  te1-1-inria-rtr-021.noc.renater.fr (193.51.177.107)  212.957 ms  212.683 ms  212.783 ms
12  inria-rocquencourt-te1-4-inria-rtr-021.noc.renater.fr (193.51.184.177)  213.021 ms  214.159 ms  214.016 ms
13  unit240-reth1-vfw-ext-dc1.inria.fr (192.93.122.19)  213.152 ms  212.698 ms  213.692 ms
14  ezp3.inria.fr (128.93.162.84)  213.233 ms  213.307 ms  212.836 ms>
```

Na rede Eduroam da USP, a partir do salto 9 foi alcançado um servidor europeu, `redclara-gw.par.fr.geant.net` e indicado pela presença do domínio `fr` e pelo endereço IP.

3 Passo 3 - Parte 1

Execute o comando `iperf`, conforme descrito no tutorial, antes de usar a opção `--switch user`, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado).

```
*** Iperf: testing TCP bandwidth between h1 and h3
```

Teste	Gbits/sec
1	17.0
2	16.6
3	16.1
4	15.8
5	13.9

Média (GBits/sec)	Intervalo de Confiança (GBits/sec)
15.88	(14.39, 17.37)

4 Passo 3 - Parte 2

Execute o comando `iperf`, conforme descrito no tutorial, com a opção `--switch user`, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes menos o da Seção anterior? Qual o motivo dessa diferença?

```
*** Iperf: testing TCP bandwidth between h1 and h3
```

Teste	Mbits/sec
1	489
2	474
3	438
4	474
5	449

Média (MBits/sec)	Intervalo de Confiança (MBits/sec)
464.8	(439.03, 490.57)

Aproximadamente 34 vezes menos. Essa diferença de velocidade é causada pela passagem dos pacotes entre o kernel-space e o user-space. Enquanto na parte 1 o fluxo foi pré-definido e ocorria diretamente no switch, portanto somente dentro do kernel-space, na parte 2 a existência do controlador faz com que os pacotes tenham que passar do kernel-space para o user-space, onde está a controladora, para retornar ao kernel-space.

5 Passo 4 - Parte 1

Execute o comando `iperf`, conforme descrito no tutorial, usando o controlador `of_tutorial.py` original sem modificação, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes menos o da Seção 3? Qual o motivo para essa diferença? Use a saída do comando `tcpdump`, deixando claro em quais computadores virtuais ele foi executado, para justificar a sua resposta.

```
*** Iperf: testing TCP bandwidth between h1 and h3
```

Teste	Mbits/sec
1	21.3
2	31.3
3	31.5
4	22.4
5	31.5

Média (MBits/sec)	Intervalo de Confiança (MBits/sec)
27.6	(21.06, 34.14)

Aproximadamente 575 vezes menos. O comando `tcpdump` rodando no host 2 (`h2-eth0`) mostra pacotes originados e destinados a outros hosts sendo recebidos na interface de rede. Portanto fica claro que todo tráfego está passando pelo controlador, que atua como um hub, enviando os pacotes a todos os hosts conectados nele. A passagem pelo controlador causa a perda de desempenho.

```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0

17:29:21.935960 IP 10.0.0.3.5001 > 10.0.0.1.54578: Flags [.], ack 10020185, win 1952, options [nop,nop,TS val 2801196 ecr 2801111], length 0
        0x0000:  0000 0000 0001 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0034 947d 4000 4006 9243 0a00 0003 0a00  .4.}@.@..C......
        0x0020:  0001 1389 d532 1f84 3092 7d27 cf8b 8010  .....2..0.}'....
        0x0030:  07a0 593c 0000 0101 080a 002a be2c 002a  ..Y<.......*.,.*
        0x0040:  bdd7                                     ..
17:29:21.952843 IP 10.0.0.1.54578 > 10.0.0.3.5001: Flags [.], ack 2, win 58, options [nop,nop,TS val 2801220 ecr 2801210], length 0
        0x0000:  0000 0000 0003 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0034 78c1 4000 4006 adff 0a00 0001 0a00  .4x.@.@.........
        0x0020:  0003 d532 1389 7d28 ea4c 1f84 3093 8010  ...2..}(.L..0...
        0x0030:  003a 4564 0000 0101 080a 002a be44 002a  .:Ed.......*.D.*
        0x0040:  be3a                                     .:
```

6 Passo 4 - Parte 2

Execute o comando `iperf`, conforme descrito no tutorial, usando o seu controlador `switch.py`, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes mais o da Seção anterior? Qual o motivo dessa diferença? Use a saída do comando `tcpdump`, deixando claro em quais computadores virtuais ele foi executado, para justificar a sua resposta.

```
*** Iperf: testing TCP bandwidth between h1 and h3
```

Teste	Mbits/sec
1	30.4
2	31.0
3	22.7
4	30.2
5	28.1

Média (MBits/sec)	Intervalo de Confiança (MBits/sec)
28.48	(24.24, 32.72)

É ligeiramente maior, mas praticamente não há diferença pois todo o tráfego ainda passa pelo controlador, o que causa mau desempenho. O pequeno ganho do `switch.py` é devido à memorização das portas dos endereços conhecidos, o que implica no envio dos pacotes somente ao destino e não a todos os hosts. Assim, as saídas do `tcpdump` estão presentes somente para os `hosts h1` e `h3` (é possível verificar as saídas do `tcpdump` direcionando a saída padrão para um arquivo, pois são muito extensas):

`tcpdump no host h1 (20 primeiros pacotes)`

```
root@mininet-vm:~# tcpdump-n -i h1-eth0 > dump_h1.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

09:46:00.983889 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
09:46:00.991436 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
09:46:00.991442 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [S], seq 1032372313, win 29200,
    options [mss 1460,sackOK,TS val 875252 ecr 0,nop,wscale 9], length 0
09:46:00.992496 IP 10.0.0.3.5001 > 10.0.0.1.45992: Flags [S.], seq 798501729, ack 1032372314, win 28960,
    options [mss 1460,sackOK,TS val 875254 ecr 875252,nop,wscale 9], length 0
09:46:00.992511 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [.], ack 1, win 58,
    options [nop,nop,TS val 875254 ecr 875254], length 0
09:46:00.993366 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [F.], seq 1, ack 1, win 58, options [nop,nop,TS val 875254 ecr 875254], length 0
09:46:00.995805 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [S], seq 2225526446, win 29200,
    options [mss 1460,sackOK,TS val 875254 ecr 0,nop,wscale 9], length 0
09:46:01.003606 IP 10.0.0.3.5001 > 10.0.0.1.45992: Flags [.], ack 2, win 57, options [nop,nop,TS val 875257 ecr 875254], length 0
09:46:01.010078 IP 10.0.0.3.5001 > 10.0.0.1.45994: Flags [S.], seq 4120615762, ack 2225526447, win 28960,
    options [mss 1460,sackOK,TS val 875257 ecr 875254,nop,wscale 9], length 0
09:46:01.010092 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 0
09:46:01.010321 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [P.], seq 1:25, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 24
09:46:01.010543 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 25:2921, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 2896
09:46:01.010562 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 2921:5817, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 2896
09:46:01.010569 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 5817:8713, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 2896
09:46:01.010576 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 8713:11609, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 2896
09:46:01.010582 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 11609:13057, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.012396 IP 10.0.0.3.5001 > 10.0.0.1.45992: Flags [F.], seq 1, ack 2, win 57, options [nop,nop,TS val 875259 ecr 875254], length 0
09:46:01.012409 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [.], ack 2, win 58, options [nop,nop,TS val 875260 ecr 875259], length 0
09:46:01.038680 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 13057:14505, ack 1, win 58, options [nop,nop,TS val 875267 ecr 875257], length 1448
09:46:01.051993 IP 10.0.0.3.5001 > 10.0.0.1.45994: Flags [.], ack 25, win 57, options [nop,nop,TS val 875267 ecr 875259], length 0
```

`tcpdump no host h2 (vazio)`

```
root@mininet-vm:~# tcpdump -n -i h2-eth0 > dump_h2.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

`tcpdump no host h3 (20 primeiros pacotes)`

```
root@mininet-vm:~# tcpdump -n -i h3-eth0 > dump_h3.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

09:46:00.990693 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
09:46:00.990713 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
09:46:00.991981 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [S], seq 1032372313, win 29200,
    options [mss 1460,sackOK,TS val 875252 ecr 0,nop,wscale 9], length 0
09:46:00.991994 IP 10.0.0.3.5001 > 10.0.0.1.45992: Flags [S.], seq 798501729, ack 1032372314, win 28960,
    options [mss 1460,sackOK,TS val 875254 ecr 875252,nop,wscale 9], length 0
09:46:00.992956 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 875254 ecr 875254], length 0
09:46:00.996804 IP 10.0.0.1.45992 > 10.0.0.3.5001: Flags [F.], seq 1, ack 1, win 58, options [nop,nop,TS val 875254 ecr 875254], length 0
09:46:00.999825 IP 10.0.0.3.5001 > 10.0.0.1.45992: Flags [.], ack 2, win 57, options [nop,nop,TS val 875257 ecr 875254], length 0
09:46:01.000238 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [S], seq 2225526446, win 29200,
    options [mss 1460,sackOK,TS val 875254 ecr 0,nop,wscale 9], length 0
09:46:01.000265 IP 10.0.0.3.5001 > 10.0.0.1.45994: Flags [S.], seq 4120615762, ack 2225526447, win 28960,
    options [mss 1460,sackOK,TS val 875257 ecr 875254,nop,wscale 9], length 0
09:46:01.009917 IP 10.0.0.3.5001 > 10.0.0.1.45992: Flags [F.], seq 1, ack 2, win 57, options [nop,nop,TS val 875259 ecr 875254], length 0
09:46:01.013167 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 0
09:46:01.039002 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [P.], seq 1:25, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 24
09:46:01.039039 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 25:1473, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039058 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 1473:2921, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039074 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 2921:4369, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039091 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 4369:5817, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039114 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 5817:7265, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039131 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 7265:8713, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039146 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 8713:10161, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
09:46:01.039160 IP 10.0.0.1.45994 > 10.0.0.3.5001: Flags [.], seq 10161:11609, ack 1, win 58, options [nop,nop,TS val 875259 ecr 875257], length 1448
```

7 Passo 4 - Parte 3

Execute o comando `iperf`, conforme descrito no tutorial, usando o seu controlador `switch.py` melhorado, 5 vezes. Escreva abaixo o valor médio e o intervalo de confiança da taxa retornada (considere sempre o primeiro valor do vetor retornado). O resultado agora corresponde a quantas vezes mais o da Seção anterior? Qual o motivo dessa diferença? Use a saída do comando `tcpdump`, deixando claro em quais computadores virtuais ele foi executado, e saídas do comando `sudo ovs-ofctl`, com os devidos parâmetros, para justificar a sua resposta.

Teste	Gbits/sec
1	23.0
2	24.4
3	23.1
4	24.5
5	24.0

Média (GBits/sec)	Intervalo de Confiança (GBits/sec)
23.8	(22.92, 24.68)

Aproximadamente 835 vezes maior, e esta diferença é devido ao envio dos fluxos conhecidos pelo controlador ao `switch`. Agora, os pacotes recebidos por uma porta com destino conhecido são enviados à porta correspondente sem passar pelo controlador, implicando em ganho de desempenho.

O `fluxo do switch s1` mostra todos os fluxos conhecidos e enviados pelo controlador ao `switch`. É possível perceber que há apenas fluxos entres os `hosts 10.0.0.1` e `10.0.0.3` (`h1` e `h3`). Também é possível perceber que o `host h2` não viu nenhum pacote pela saída do `tcpdump`:

`fluxo do switch s1`

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1

NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=409.631s, table=0, n_packets=43761, n_bytes=2490634810, idle_age=404, tcp,vlan_tci=0x0000,
 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,tp_src=46016,tp_dst=5001 actions=output:3
 cookie=0x0, duration=409.715s, table=0, n_packets=3, n_bytes=198, idle_age=409, tcp,vlan_tci=0x0000,
 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=16,tp_src=46014,tp_dst=5001 actions=output:3
 cookie=0x0, duration=409.559s, table=0, n_packets=10112, n_bytes=667392, idle_age=404, tcp,vlan_tci=0x0000,
 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=46016 actions=output:1
 cookie=0x0, duration=409.669s, table=0, n_packets=1, n_bytes=66, idle_age=409, tcp,vlan_tci=0x0000,
 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=46014 actions=output:1
 cookie=0x0, duration=409.752s, table=0, n_packets=0, n_bytes=0, idle_age=409, arp,vlan_tci=0x0000,
 dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,arp_spa=10.0.0.3,arp_tpa=10.0.0.1,arp_op=2 actions=output:1
```

`tcpdump do host h1 (20 primeiros pacotes)`

```
root@mininet-vm:~# tcpdump -n -i h1-eth0 > dump_h1.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

09:56:20.435536 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
09:56:20.437085 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
09:56:20.437090 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [S], seq 1455017156, win 29200,
    options [mss 1460,sackOK,TS val 1030115 ecr 0,nop,wscale 9], length 0
09:56:20.505075 IP 10.0.0.3.5001 > 10.0.0.1.46014: Flags [S.], seq 2948560447, ack 1455017157, win 28960,
    options [mss 1460,sackOK,TS val 1030126 ecr 1030115,nop,wscale 9], length 0
09:56:20.505112 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 1030133 ecr 1030126], length 0
09:56:20.506125 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [F.], seq 1, ack 1, win 58, options [nop,nop,TS val 1030133 ecr 1030126], length 0
09:56:20.520749 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [S], seq 1231945580, win 29200,
    options [mss 1460,sackOK,TS val 1030137 ecr 0,nop,wscale 9], length 0
09:56:20.520922 IP 10.0.0.3.5001 > 10.0.0.1.46014: Flags [.], ack 2, win 57, options [nop,nop,TS val 1030134 ecr 1030133], length 0
09:56:20.521224 IP 10.0.0.3.5001 > 10.0.0.1.46014: Flags [F.], seq 1, ack 2, win 57, options [nop,nop,TS val 1030137 ecr 1030133], length 0
09:56:20.521236 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [.], ack 2, win 58, options [nop,nop,TS val 1030137 ecr 1030137], length 0
09:56:20.588779 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [S.], seq 2795696301, ack 1231945581, win 28960,
    options [mss 1460,sackOK,TS val 1030147 ecr 1030137,nop,wscale 9], length 0
09:56:20.588815 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 1030154 ecr 1030147], length 0
09:56:20.590129 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [P.], seq 1:25, ack 1, win 58, options [nop,nop,TS val 1030154 ecr 1030147], length 24
09:56:20.591467 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 25:2921, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591524 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 2921:5817, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591550 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 5817:8713, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591571 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 8713:11609, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591589 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 11609:13057, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 1448
09:56:20.630758 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [.], ack 25, win 57, options [nop,nop,TS val 1030154 ecr 1030154], length 0
09:56:20.630776 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 13057:15953, ack 1, win 58, options [nop,nop,TS val 1030165 ecr 1030154], length 2896
```

`tcpdump no host h2 (vazio)`

```
root@mininet-vm:~# tcpdump -n -i h2-eth0 > dump_h2.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

`tcpdump do host h3 (20 primeiros pacotes)`

```
root@mininet-vm:~# tcpdump -n -i h3-eth0 > dump_h3.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

09:56:20.436575 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
09:56:20.436588 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
09:56:20.474823 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [S], seq 1455017156, win 29200,
    options [mss 1460,sackOK,TS val 1030115 ecr 0,nop,wscale 9], length 0
09:56:20.474853 IP 10.0.0.3.5001 > 10.0.0.1.46014: Flags [S.], seq 2948560447, ack 1455017157, win 28960,
    options [mss 1460,sackOK,TS val 1030126 ecr 1030115,nop,wscale 9], length 0
09:56:20.505381 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 1030133 ecr 1030126], length 0
09:56:20.506136 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [F.], seq 1, ack 1, win 58, options [nop,nop,TS val 1030133 ecr 1030126], length 0
09:56:20.508192 IP 10.0.0.3.5001 > 10.0.0.1.46014: Flags [.], ack 2, win 57, options [nop,nop,TS val 1030134 ecr 1030133], length 0
09:56:20.521109 IP 10.0.0.3.5001 > 10.0.0.1.46014: Flags [F.], seq 1, ack 2, win 57, options [nop,nop,TS val 1030137 ecr 1030133], length 0
09:56:20.521240 IP 10.0.0.1.46014 > 10.0.0.3.5001: Flags [.], ack 2, win 58, options [nop,nop,TS val 1030137 ecr 1030137], length 0
09:56:20.558838 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [S], seq 1231945580, win 29200,
    options [mss 1460,sackOK,TS val 1030137 ecr 0,nop,wscale 9], length 0
09:56:20.558865 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [S.], seq 2795696301, ack 1231945581, win 28960,
    options [mss 1460,sackOK,TS val 1030147 ecr 1030137,nop,wscale 9], length 0
09:56:20.589093 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], ack 1, win 58, options [nop,nop,TS val 1030154 ecr 1030147], length 0
09:56:20.590142 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [P.], seq 1:25, ack 1, win 58, options [nop,nop,TS val 1030154 ecr 1030147], length 24
09:56:20.590154 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [.], ack 25, win 57, options [nop,nop,TS val 1030154 ecr 1030154], length 0
09:56:20.591482 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 25:2921, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591494 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [.], ack 2921, win 68, options [nop,nop,TS val 1030155 ecr 1030155], length 0
09:56:20.591529 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 2921:5817, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591537 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [.], ack 5817, win 80, options [nop,nop,TS val 1030155 ecr 1030155], length 0
09:56:20.591553 IP 10.0.0.1.46016 > 10.0.0.3.5001: Flags [.], seq 5817:8713, ack 1, win 58, options [nop,nop,TS val 1030155 ecr 1030147], length 2896
09:56:20.591560 IP 10.0.0.3.5001 > 10.0.0.1.46016: Flags [.], ack 8713, win 91, options [nop,nop,TS val 1030155 ecr 1030155], length 0
```

8 Passo 5

Explique a lógica implementada no seu controlador `firewall.py` e mostre saídas de comandos que comprovem que ele está de fato funcionando (saídas dos comandos `tcpdump`, `sudo ovs-ofctl`, `nc`, `iperf` e `telnet` são recomendadas)

O Firewall lê as regras a partir de um arquivo externo (a definição das regras no arquivo externo estão no arquivo `LEIAME.pdf`), e todas as regras são enviadas ao `switch`. Uma regra é composta por 4 parâmetros (endereço de origem, destino, porta e protocolo) e o Firewall é composto por várias regras. Um pacote é bloqueado se casa com os parâmetros definidos por qualquer regra.

Regra de bloqueio por endereço: bloquear pacotes entre o endereço de origem 10.0.0.1 e destino 10.0.03

Devemos mostrar que não há fluxo de pacotes direcionado entre `h1` e `h3`:

`iperf do h1 como cliente tentando acessar h3:`

```
root@mininet-vm:~# iperf -c 10.0.0.3
connect failed: Connection timed out
root@mininet-vm:~#
```

`iperf do h2 como cliente tentando acessar h3:`

```
root@mininet-vm:~# iperf -c 10.0.0.3
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.2 port 44088 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  27.7 GBytes  23.8 Gbits/sec
root@mininet-vm:~#
```

`iperf do h3 como servidor:`

```
root@mininet-vm:~# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 16] local 10.0.0.3 port 5001 connected with 10.0.0.2 port 44088
[ ID] Interval       Transfer     Bandwidth
[ 16]  0.0-10.0 sec  27.7 GBytes  23.7 Gbits/sec
```

Criando um ambiente cliente/servidor com o `iperf`, o cliente `h1` não consegue realizar a conexão com o servidor `h3`, pois todos os pacotes são bloqueados. O cliente `h2` consegue realizar a conexão sem problemas.

`pingall:`

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> X h2
*** Results: 33% dropped (4/6 received)
mininet>
```

Realizando o `pingall` entre todos os `hosts`, é possível verificar pela saída `tcpdump no host h1` que:

- `h1` envia um `echo request` para `h2` e recebe um `echo reply` do mesmo.
- `h1` envia um `echo request` para `h3`, mas não recebe o `echo reply`.
- `h1` recebe `echo request` de `h2` e `h3` e envia `echo reply` para ambos.

`tcpdump no host h1`

```
root@mininet-vm:~# tcpdump -n -i h1-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:28:36.869134 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8144, seq 1, length 64
12:28:36.869903 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8144, seq 1, length 64
12:28:36.877940 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 8145, seq 1, length 64
12:28:41.874790 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
12:28:41.875105 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
12:28:41.875114 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
12:28:41.875532 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
12:28:41.890470 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
12:28:41.890882 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
12:28:46.884129 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 8148, seq 1, length 64
12:28:46.884155 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 8148, seq 1, length 64
12:28:46.897372 IP 10.0.0.3 > 10.0.0.1: ICMP echo request, id 8150, seq 1, length 64
12:28:46.897383 IP 10.0.0.1 > 10.0.0.3: ICMP echo reply, id 8150, seq 1, length 64
12:28:51.906856 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
12:28:51.906875 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
```

É possível perceber pela saída `tcpdump no host h2` que:

- `h2` recebe `echo request` de `h1` e `h3` e envia `echo reply` para ambos.
- `h2` envia `echo request` para `h1` e `h3` e recebe `echo reply` de ambos.

`tcpdump no host h2`

```
root@mininet-vm:~# tcpdump -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:28:36.869367 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8144, seq 1, length 64
12:28:36.869403 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8144, seq 1, length 64
12:28:41.874819 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
12:28:41.875034 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
12:28:41.875052 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
12:28:41.875516 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
12:28:46.883952 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 8148, seq 1, length 64
12:28:46.884540 IP 10.0.0.1 > 10.0.0.2: ICMP echo reply, id 8148, seq 1, length 64
12:28:46.893729 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 8149, seq 1, length 64
12:28:46.893975 IP 10.0.0.3 > 10.0.0.2: ICMP echo reply, id 8149, seq 1, length 64
12:28:51.906608 ARP, Request who-has 10.0.0.3 tell 10.0.0.2, length 28
12:28:51.906929 ARP, Request who-has 10.0.0.2 tell 10.0.0.3, length 28
12:28:51.906940 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
12:28:51.907436 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
12:28:56.902982 IP 10.0.0.3 > 10.0.0.2: ICMP echo request, id 8153, seq 1, length 64
12:28:56.903009 IP 10.0.0.2 > 10.0.0.3: ICMP echo reply, id 8153, seq 1, length 64
```

É possível perceber pela saída `tcpdump no host h3` que:

- h3 recebe echo request de h2 e envia o echo reply para o mesmo.
- h3 envia echo request para h2 e recebe echo reply do mesmo.
- h3 não recebe o echo request de h1.
- h3 envia o echo request para h1, mas não recebe o echo reply do mesmo.

tcpdump no host h3

```
root@mininet-vm:~# tcpdump -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:28:41.890631 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
12:28:41.890645 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
12:28:46.893824 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 8149, seq 1, length 64
12:28:46.893836 IP 10.0.0.3 > 10.0.0.2: ICMP echo reply, id 8149, seq 1, length 64
12:28:46.897290 IP 10.0.0.3 > 10.0.0.1: ICMP echo request, id 8150, seq 1, length 64
12:28:51.906633 ARP, Request who-has 10.0.0.2 tell 10.0.0.3, length 28
12:28:51.906643 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
12:28:51.906980 ARP, Request who-has 10.0.0.3 tell 10.0.0.2, length 28
12:28:51.906989 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
12:28:51.907419 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
12:28:51.907444 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
12:28:56.902809 IP 10.0.0.3 > 10.0.0.2: ICMP echo request, id 8153, seq 1, length 64
12:28:56.903404 IP 10.0.0.2 > 10.0.0.3: ICMP echo reply, id 8153, seq 1, length 64
```

Portanto, os pacotes de origem h1 e destino h3 estão bloqueados.

Regra de bloqueio por protocolo: bloquear todos os pacotes com protocolo UDP
Devemos mostrar que qualquer pacote UDP independentemente do host ou da porta é bloqueado:

iperf do h1 como servidor daemon TCP e UDP:

```
root@mininet-vm:~# iperf -s -D
root@mininet-vm:~# Running Iperf Server as a daemon
The Iperf daemon process ID : 12235

root@mininet-vm:~# iperf -u -s -D
root@mininet-vm:~# Running Iperf Server as a daemon
The Iperf daemon process ID : 12242

root@mininet-vm:~#
```

É possível perceber pelas saídas do iperf de h2 e h3 que ambos os clientes conseguem conectar-se ao servidor através do protocolo TCP, mas não com UDP (a última linha exibe um *WARNING* sobre o não recebimento do pacote) utilizando a mesma porta (5001).

iperf do h2 como cliente TCP e depois UDP tentando acessar h1:

```
root@mininet-vm:~# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.2 port 37838 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  12.1 GBytes  10.4 Gbits/sec
root@mininet-vm:~# iperf -u -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.2 port 33262 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 15] Sent 893 datagrams
[ 15] WARNING: did not receive ack of last datagram after 10 tries.
root@mininet-vm:~#
```

iperf do h3 como cliente TCP e depois UDP tentando acessar h1:

```
root@mininet-vm:~# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.3 port 36012 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  8.99 GBytes  7.72 Gbits/sec
root@mininet-vm:~# iperf -u -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.3 port 34679 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 15] Sent 893 datagrams
[ 15] WARNING: did not receive ack of last datagram after 10 tries.
root@mininet-vm:~#
```

Regra de bloqueio de porta: bloquear pacotes com porta de destino 5000
Devemos mostrar que apenas a porta 5000 está bloqueada:

`iperf do h1 como servidor daemon na porta 5000 e 5001:`

```
root@mininet-vm:~# iperf -s -p 5000 -D
root@mininet-vm:~# Running Iperf Server as a daemon
The Iperf daemon process ID : 15874

root@mininet-vm:~# iperf -s -p 5001 -D
root@mininet-vm:~# Running Iperf Server as a daemon
The Iperf daemon process ID : 15879

root@mininet-vm:~#
```

É possível perceber pelas saídas do `iperf` de `h2` e `h3` que ambos os clientes não conseguem conectar-se ao servidor através da porta 5000, apenas pela 5001.

`iperf do h2 como cliente tentando acessar o h1 na porta 5000 e 5001:`

```
root@mininet-vm:~# iperf -c 10.0.0.1 -p 5000
connect failed: Connection timed out
root@mininet-vm:~# iperf -c 10.0.0.1 -p 5001
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.2 port 37974 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  3.21 GBytes  2.75 Gbits/sec
root@mininet-vm:~#
```

`iperf do h3 como cliente tentando acessar o h1 na porta 5000 e 5001:`

```
root@mininet-vm:~# iperf -c 10.0.0.1 -p 5000
connect failed: Connection timed out
root@mininet-vm:~# iperf -c 10.0.0.1 -p 5001
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 15] local 10.0.0.3 port 36148 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 15]  0.0-10.0 sec  2.76 GBytes  2.37 Gbits/sec
root@mininet-vm:~#
```

9 Configuração dos computadores virtual e real usados nas medições (se foi usado mais de um, especifique qual passo foi feito com cada um)

- Computador virtual: Ubuntu (64-bit), 1GB de Memoria RAM e 8GB de armazenamento.
- Computador real: Linux Mint (64-bit), Processador Intel Core-i5@2.2GHz, 8GB de Memória RAM e 128GB de armazenamento.

10 Referências

- Kumar, Pradeep. ”How to capture and analyze packets with tcpdump command on Linux”. *Linux Techi*, 26 Ago. 2018, www.linuxtechi.com/capture-analyze-packets-tcpdump-command-linux/