

Relatório do EP2

MAC5910 – Programação para Redes de Computadores – 2/2021

Giovanni Aparecido da Silva Oliveira (13071092)

01/11/2021

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?

Resposta:

Quando um pacote sem fluxo definido chega no *switch*, ele é encaminhado ao controlador [1].

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?

Resposta:

```
mininet@mininet-vm:~$ sudo traceroute -I www.inria.fr
traceroute to www.inria.fr (128.93.162.83), 30 hops max, 60 byte
packets
 1  10.0.2.2 (10.0.2.2)  0.375 ms  0.304 ms  0.283 ms
 2  192.168.0.1 (192.168.0.1)  1.659 ms  3.981 ms  3.973 ms
 3  10.16.0.1 (10.16.0.1)  13.391 ms  13.780 ms  13.720 ms
 4  bb025009.virtua.com.br (187.2.80.9)  13.200 ms  13.829 ms  13.646 ms
 5  embratel-T0-2-1-0-1-agg01.bgp0ao.embratel.net.br (189.23.245.1)
    13.402 ms  13.389 ms  13.371 ms
 6  200.230.235.55 (200.230.235.55)  133.603 ms  136.983 ms
    136.800 ms
 7  ebt-B11911-intl01.nyk.embratel.net.br (200.230.220.154)
    142.822 ms  147.304 ms  147.268 ms
 8  ae11.cr1-nyc2.ip4.gtt.net (209.120.132.221)  133.283 ms
    129.174 ms  130.737 ms
 9  et-3-3-0.cr2-par7.ip4.gtt.net (213.200.119.214)  217.572 ms
    217.519 ms  217.497 ms
10  renater-gw-ix1.gtt.net (77.67.123.206)  224.720 ms
    224.014 ms  224.000 ms
11  tel-1-inria-rtr-021.noc.renater.fr (193.51.177.107)  224.189 ms
    223.679 ms  227.615 ms
12  inria-rocquencourt-tel-4-inria-rtr-021.noc.renater.fr (193.51.184.177)  22
    223.079 ms  223.431 ms
13  unit240-reth1-vfw-ext-dc1.inria.fr (192.93.122.19)  224.023 ms
    307.342 ms  307.218 ms
14  prod-inriafr-cms.inria.fr (128.93.162.83)  307.305 ms
    308.803 ms  308.111 ms
```

Os pacotes trafegaram na LAN doméstica de origem, nos saltos de número 1 e 2 (IPs 10.0.2.2 e 192.168.0.1). No salto 3 (IP 10.16.0.1), os pacotes alcançaram uma rede local privada, provisionada pelo ISP.

A partir do salto 3, até o salto de número 7 (IP 200.230.220.154), os pacotes permaneceram no Brasil [2, 3, 4], embora não seja possível determinar com certeza a região, em decorrência da divergência dos dados de localidade que, exemplificando com o salto 7, variam entre Rio de Janeiro [5], São Paulo [6] e Minas Gerais [7], ao utilizar serviços distintos de informação de localização geográfica de endereços IP.

No salto de número 8, os pacotes alcançaram o primeiro roteador na Europa, embora novamente haja divergência entre a localidade (Alemanha [8] e França [9]).

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).

Resposta:

Seja \bar{X} o valor médio para a amostra populacional obtida no experimento e CI_k o intervalo de confiança, em função do parâmetro de confiança k . Então:

$$\bar{X} = \overline{Rate}_{h1 \rightarrow h3}^{ovsk} = \frac{69.6 + 57.3 + 57.2 + 68.5 + 67.1}{5} = \mathbf{63.94 \text{ [Gb/s]}} \quad (1)$$

$$CI_k = \bar{X} \pm \frac{k * \sigma_X}{\sqrt{N-1}} = 63.94 \pm k * \frac{5.52}{\sqrt{4}} = \mathbf{63.94 \pm k * 2.76 \text{ [Gb/s]}} \quad (2)$$

CI%	k	CI _k
68.3%	1	63.94 ± 2.76; (± 4.32%)
90%	1.645	63.94 ± 4.54; (± 7.10%)
95%	1.960	63.94 ± 5.409 [Gb/s]; (± 8.46%)
99%	2.576	63.94 ± 7.109; (± 11.12%)
99.9999%	4.892	63.94 ± 13.501; (± 21.12%)

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?

Resposta:

Seja \bar{X} o valor médio para a amostra populacional obtida no experimento e CI_k o intervalo de confiança, em função do parâmetro de confiança k . Então:

$$\bar{X} = \overline{Rate}_{h1 \rightarrow h3}^{user} = \frac{357 + 403 + 288 + 309 + 351 + 419}{5} = 354.5 \text{ [Mb/s]} \quad (3)$$

$$CI_k = \bar{X} \pm \frac{k * \sigma_X}{\sqrt{N-1}} = 354.5 \pm k * \frac{21.67}{\sqrt{4}} = 354.5 \pm k * 20.835 \text{ [Mb/s]} \quad (4)$$

CI%	k	CI _k
68.3%	1	354.5 ± 20.835 ; (± 5.88%)
90%	1.645	354.5 ± 34.274; (± 9.67%)
95%	1.960	354.5 ± 40.838 [Mb/s]; (±11.52%)
99%	2.576	354.5 ± 53.672 ; (± 15.14%)
99.9999%	4.892	354.5 ± 101.927; (± 28.75%)

O valor médio dos testes da presente Seção é aproximadamente 184.7 vezes menor que o valor médio obtido na Seção 3.

A diferença entre os valores, decorre do fato de que, no *switch* da Seção anterior, com a opção `ovsk`, os pacotes trafegam dentro do mesmo ambiente virtual de rede, caracterizado como *kernel-space*.

O *switch* da presente seção executa em um ambiente separado, chamado *user-space*. Quando executando nesse ambiente, todos os pacotes que trafegam pelo *switch* devem passar do *kernel-space* para o *user-space*. Essa passagem dos pacotes de e para o *user-space* é reponsável pela queda significativa no desempenho.

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.

Resposta:

Seja \bar{X} o valor médio para a amostra populacional obtida no experimento e CI_k o intervalo de confiança, em função do parâmetro de confiança k . Então:

$$\bar{X} = \overline{Rate}_{h1 \rightarrow h3}^{user} = \frac{11.5 + 7.7 + 6.03 + 14 + 10.6}{5} = \mathbf{9.966 \text{ [Mb/s]}} \quad (5)$$

$$CI_k = \bar{X} \pm \frac{k * \sigma_X}{\sqrt{N-1}} = 9.966 \pm k * \frac{3.149}{\sqrt{4}} = \mathbf{9.966 \pm k * 1.408 \text{ [Mb/s]}} \quad (6)$$

CI%	k	CI _k
68.3%	1	9.966 ± 1.408; (± 14.13%)
90%	1.645	9.966 ± 2.316; (± 23.24%)
95%	1.960	9.966 ± 2.76 [Mb/s]; (± 27.69%)
99%	2.576	9.966 ± 3.627; (± 36.40%)
99.9999%	4.892	9.966 ± 6.888; (± 69.12%)

O valor médio dos testes da presente Seção é aproximadamente 6569.7 vezes menor que o valor médio obtido na Seção 3.

A diferença entre os valores, decorre do fato de que, no *switch* da Seção anterior, com a opção `ovsk`, os pacotes trafegam dentro do mesmo ambiente virtual de rede, caracterizado como *kernel-space*.

O *hub* da presente seção executa em um ambiente separado, chamado *user-space*. Quando executando nesse ambiente, todos os pacotes que trafegam pelo *switch* devem passar do *kernel-space* para o *user-space*. Essa passagem dos pacotes de e para o *user-space* é reponsável pela queda significativa no desempenho.

Na Figura 1, a saída do comando `tcpdump` permite identificar o comportamento de *hub* do controlador executado nessa Seção. A causa é simples: a partir de **h3** (ou qualquer outro host que se conecte ao *switch*), pode-se captar todo o fluxo de pacotes entre **h1** e **h2**.

```

11:28:31.825379 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  .....E.
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .Tl.@.@.....
    0x0020:  0000 0000 0000 0a00 0002  .....d.P.....}a..
11:28:31.826128 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....E.
    0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .T....@.....
    0x0020:  0000 0000 0001 0a00 0001  .....d.P.....}a..
11:28:31.826660 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 20621, seq 1, length 64
    0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .....E.
    0x0010:  0054 6c8a 4000 4001 ba1c 0a00 0001 0a00  .Tl.@.@.....
    0x0020:  0002 0800 5cbe 508d 0001 cf8e 7d61 0000  ....\P.....}a..
    0x0030:  0000 33f0 0b00 0000 0000 1011 1213 1415  ..3.....
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....! "$%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637 67
11:28:31.827195 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 20621, seq 1, length 64
    0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .....E.
    0x0010:  0054 a3ce 0000 4001 c2d8 0a00 0002 0a00  .T....@.....
    0x0020:  0001 0000 64be 508d 0001 cf8e 7d61 0000  ....d.P.....}a..
    0x0030:  0000 33f0 0b00 0000 0000 1011 1213 1415  ..3.....
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....! "$%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637 67

```

Figura 1: Saída do comando 'tcpdump' sendo executado em h3 (MAC :::::3, IP 10.0.0.3) em simultaneidade com o controlador da presente Seção. É possível observar o tráfego de mensagens entre h1 (MAC :::::1, IP 10.0.0.1) e h2 (MAC :::::2, IP 10.0.0.2) em h3.

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.

Resposta:

Seja \bar{X} o valor médio para a amostra populacional obtida no experimento e CI_k o intervalo de confiança, em função do parâmetro de confiança k . Então:

$$\bar{X} = \overline{Rate}_{h1 \rightarrow h3}^{user} = \frac{6.38 + 10.7 + 13.3 + 8.04 + 9.72}{5} = \mathbf{9.682 \text{ [Mb/s]}} \quad (7)$$

$$CI_k = \bar{X} \pm \frac{k * \sigma_X}{\sqrt{N-1}} = 9.682 \pm k * \frac{2.632}{\sqrt{4}} = \mathbf{9.682 \pm k * 1.177 \text{ [Mb/s]}} \quad (8)$$

CI%	k	CI _k
68.3%	1	9.682 ± 1.177; (± 12.23%)
90%	1.645	9.682 ± 1.936; (± 20.11%)
95%	1.960	9.682 ± 2.307 [Mb/s]; (± 23.96%)
99%	2.576	9.682 ± 3.032; (± 31.49%)
99.9999%	4.892	9.682 ± 5.758; (± 59.81%)

O valor médio dos testes da presente Seção é aproximadamente 0.97 vezes o valor médio obtido na Seção 5.

A proximidade entre os valores, decorre do fato de que ambos controladores rodam no ambiente conhecido como *user-space*. Embora a amostra seja pequena para realizar uma afirmação mais sólida, conforme o esperado, há uma ligeira melhora no desempenho, dado que o número de mensagens trocadas reduz significativamente ao compararmos o *hub* com o *switch*.

Na Figura 2, a saída do comando `tcpdump` permite identificar o comportamento de *switch* do controlador executado nessa Seção. É possível observar o tráfego de mensagens de aplicação de h1 para h2, apenas a partir de h2, e não em h3, o que demonstra tal comportamento do controlador, que não faz mais *broadcast* de pacotes com destinatários *unicast*, mas envia apenas ao devido destinatário (o envio de requisições ARP e outras mensagens *broadcast* continua ocorrendo normalmente).

```

"Node: h2"
root@mininet-vm:~/pox# tcpdump -XX -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
15:18:38.217713 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0000 0a00 0002  .....
15:18:38.217780 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0001 0a00 0001  .....
15:18:38.225708 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 29640, seq 1, length 64
    0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .....E.
    0x0010:  0054 5e74 4000 4001 c832 0a00 0001 0a00  .T^t@.@..2.....
    0x0020:  0002 0800 e1f8 73c8 0001 bec4 7d61 0000  .....s.....}a..
    0x0030:  0000 a444 0300 0000 0000 1011 1213 1415  ...D.....
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....! "$%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637 67
15:18:38.225767 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 29640, seq 1, length 64
    0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .....E.
    0x0010:  0054 0bd6 0000 4001 5ad1 0a00 0002 0a00  .T....@.Z.....

"Node: h3"
root@mininet-vm:~/pox# tcpdump -XX -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
15:18:38.217708 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0000 0a00 0002  .....

```

Figura 2: Saída do comando 'tcpdump' sendo executado em h2 (MAC :::::2, IP 10.0.0.2) e h3 (MAC :::::3, IP 10.0.0.3), respectivamente, em simultaneidade com o controlador da presente Seção. É possível observar o tráfego de mensagens entre h1 (MAC :::::1, IP 10.0.0.1) e h2 (MAC :::::2, IP 10.0.0.2) é visível apenas em h2, e não mais em h3 (ou outros eventuais *hosts* que se conectem ao *switch*).

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.

Resposta:

Seja \bar{X} o valor médio para a amostra populacional obtida no experimento e CI_k o intervalo de confiança, em função do parâmetro de confiança k . Então:

$$\bar{X} = \overline{Rate}_{h1 \rightarrow h3}^{user} = \frac{69.9 + 64.5 + 50.8 + 57.6 + 67.7}{5} = \mathbf{62.1 \text{ [Gb/s]}} \quad (9)$$

$$CI_k = \bar{X} \pm \frac{k * \sigma_X}{\sqrt{N-1}} = 62.1 \pm k * \frac{7.84}{\sqrt{4}} = \mathbf{62.1 \pm k * 3.506 \text{ [Gb/s]}} \quad (10)$$

CI%	k	CI _k
68.3%	1	62.1 ± 3.5068; (± 5.65%)
90%	1.645	62.1 ± 5.768; (± 9.29%)
95%	1.960	62.1 ± 6.873 [Gb/s]; (±11.07%)
99%	2.576	62.1 ± 9.033; (± 14.55%)
99.9999%	4.892	62.1 ± 17.153; (± 27.62%)

O valor médio dos testes da presente Seção é aproximadamente 6567.8 vezes maior que o valor médio obtido na Seção anterior.

O ganho obtido nesse *switch*, decorre do uso do *kernel-space* para realizar o roteamento de pacotes, ativado por uma mensagem de controle, gerada pela chamada `of.ofp_flow_mod()`.

Na Figura 3, a saída do comando `tcpdump` permite identificar o comportamento de *switch* do controlador executado nessa Seção. É possível observar o tráfego de mensagens de aplicação de h1 para h2, apenas a partir de h2, e não mais em h3, o que demonstra um comportamento funcional análogo ao do controlador desenvolvido na Seção anterior.

Na Figura 4, é possível notar o comportamento de alteração da tabela de fluxos do *switch* observado. Inicialmente, a tabela de rotas encontra-se vazia. Após requisições serem realizadas entre os *hosts* h1 e h2, nota-se a criação de duas regras, uma em cada sentido, para suportar o fluxo de dados entre os dois *hosts*.

```

"Node: h2"
root@mininet-vm:~/pox# tcpdump -XX -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
14:19:18.677983 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0000 0a00 0002                .....
14:19:18.678002 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0001 0a00 0001                .....
14:19:18.686716 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 26532, seq 1, length 64
    0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .....E.
    0x0010:  0054 cc30 4000 4001 5a76 0a00 0001 0a00  .T.0@.0.Zv.....
    0x0020:  0002 0800 215f 67a4 0001 d6b6 7d61 0000  ....!_g.....}a..
    0x0030:  0000 5210 0a00 0000 0000 1011 1213 1415  ..R.....
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!"#$%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
14:19:18.686730 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 26532, seq 1, length 64
    0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .....E.
    0x0010:  0054 442f 0000 4001 2278 0a00 0002 0a00  .TD/..@."x.....

"Node: h3"
root@mininet-vm:~/pox# tcpdump -XX -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
14:19:18.677981 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0000 0a00 0002                .....

```

Figura 3: Saída do comando 'tcpdump' sendo executado em h2 (MAC :::::2, IP 10.0.0.2) e h3 (MAC :::::3, IP 10.0.0.3), em simultaneidade com o controlador da presente Seção. É possível observar o tráfego de mensagens entre h1 (MAC :::::1, IP 10.0.0.1) e h2 (MAC :::::2, IP 10.0.0.2) é visível apenas em h2, e não mais em h3.

```

root@mininet-vm:~/pox# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
root@mininet-vm:~/pox# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=3.799s, table=0, n_packets=1, n_bytes=98, idle_timeout=100
, idle_age=3, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:0
0:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0 actions=ou
tput:2
  cookie=0x0, duration=3.795s, table=0, n_packets=1, n_bytes=98, idle_timeout=100
, idle_age=3, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:0
0:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=ou
tput:1

```

Figura 4: Resultado de duas execuções do comando 'ovs-ofctl dump-flows s1'. A primeira execução, antes de haver troca de mensagens entre h1 (MAC :::::1, IP 10.0.0.1) e h2 (MAC :::::2, IP 10.0.0.2). A segunda execução, após h1 e h2 terem trocado mensagens.

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)

Resposta:

`firewall.py` contém a implementação de um *firewall*, com regras na forma $(IP_{src}, IP_{dst}, Port, Prot)$, onde: IP_{src} é o endereço de IP de origem, IP_{dst} é o endereço de destino, $Port$ é o número da porta do sistema operacional e $Prot$ é o protocolo que pode ser definido em `tcp`, `udp` ou `None`. Cada um dos campos pode ter seu valor definido como `None`, a fim de que qualquer valor seja aceito para o referido campo. As regras do *firewall* são lidas a partir de uma variável do *script* com nome `rules`. O trecho que deve ser alterado está bem sinalizado no código (ver linha 7). Para os testes apresentados nessa seção, foram utilizadas as seguintes configurações:

```
(None, None, None, 'tcp'),  
(IPAddr('10.0.0.3'), None, None, None),  
(None, IPAddr('10.0.0.4'), None, None),  
(IPAddr('10.0.0.1'), IPAddr('10.0.0.3'), None, None),  
(None, None, 3001, None),  
(IPAddr('10.0.0.2'), None, 3000, None),  
(IPAddr('10.0.0.1'), IPAddr('10.0.0.2'), 3000, None),
```

Por conveniência, o controlador possui embutido consigo o *switch* desenvolvido na questão anterior. Para desabilitá-lo basta comentar a linha:

```
core.openflow.addListenerByName("ConnectionUp", start_switch)
```

Nas Figuras 5 e 6, é possível observar o ambiente utilizado para realizar testes de verificação de correção do funcionamento do *firewall*, para as regras mencionadas acima. A seguir, uma listagem com a descrição de cada teste realizado e o resultado esperado:

Teste 1¹

Regra	<code>(None, None, None, 'tcp')</code>
Casos de Teste	<code>10.0.0.*:*/tcp → 10.0.0.*:*/tcp</code>
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 2

Regra	<code>(IPAddr('10.0.0.3'), None, None, None)</code>
Casos de Teste	<code>10.0.0.3:*/udp → 10.0.0.*:*/udp</code>
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 3

¹O caractere `''` possui a semântica de qualquer valor ou *wildcard*. No caso de uso, significa que testes foram realizados com diversas variações do respectivo campo.

Regra	(None, IPAddr('10.0.0.4'), None, None)
Casos de Teste	10.0.0.*:*/udp → 10.0.0.4:*/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 4

Regra	(IPAddr('10.0.0.1'), IPAddr('10.0.0.3'), None, None)
Casos de Teste	10.0.0.1:*/udp → 10.0.0.3:*/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 5

Regra	(None, None, 3001, None)
Casos de Teste	10.0.0.*:*/udp → 10.0.0.*:3001/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 5b

Regra	(None, None, 3001, None)
Casos de Teste	10.0.0.*:3001/udp → 10.0.0.*:*/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 6

Regra	(IPAddr('10.0.0.2'), None, 3000, None)
Casos de Teste	10.0.0.2:*/udp → 10.0.0.*:3000/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 6b

Regra	(IPAddr('10.0.0.2'), None, 3000, None)
Casos de Teste	10.0.0.2:3000/udp → 10.0.0.*:*/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 6c

Regra	(IPAddr('10.0.0.2'), None, 3000, None)
Casos de Teste	10.0.0.2:3002/udp → 10.0.0.1:1234/udp
Resultado	Os pacotes chegam no destino, pois não são bloqueado pelo <i>firewall</i> , uma vez que o fluxo se dá em uma porta diferente da 3000.

Teste 6d

Regra	(IPAddr('10.0.0.2'), None, 3000, None)
Casos de Teste	10.0.0.2:1234/udp → 10.0.0.1:3002/udp
Resultado	Os pacotes chegam no destino, pois não são bloqueado pelo <i>firewall</i> , uma vez que o fluxo se dá em uma porta diferente da 3000.

Teste 7

Regra	(IPAddr('10.0.0.1'), IPAddr('10.0.0.2'), 3000, None)
Casos de Teste	10.0.0.1:*/udp → 10.0.0.2:3000/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 7b

Regra	(IPAddr('10.0.0.1'), IPAddr('10.0.0.2'), 3000, None)
Casos de Teste	10.0.0.1:3000/udp → 10.0.0.*:*/udp
Resultado	Nenhum pacote chega no destino pois é bloqueado pelo <i>firewall</i>

Teste 7c

Regra	(IPAddr('10.0.0.1'), IPAddr('10.0.0.2'), 3000, None)
Casos de Teste	10.0.0.4:3000/udp → 10.0.0.3:1234/udp
Resultado	Os pacotes chegam no destino, pois não são bloqueado pelo <i>firewall</i> , uma vez que o fluxo se dá em IPs diferentes dos especificados na regra.

Teste de Desempenho: Os testes de desempenho apresentaram resultados semelhantes às do *switch* funcionando isoladamente. Desse modo, conclui-se que o *firewall* não impôs atraso significativo à conexão de dados.

The figure displays eight terminal windows arranged in a 4x2 grid, showing network traffic and netcat commands across four hosts (h1, h2, h3, h4). The top row shows the results of the `tcpdump` command on the principal interface of each host, displaying ARP and ICMP traffic. The bottom row shows the results of the `netcat` (nc) command, which is used to send and receive packets. The terminal windows are titled "Node: h1", "Node: h2", "Node: h3", and "Node: h4".

Figura 5: Ambiente *Mininet* com 4 *hosts* comunicando-se através do *firewall* via TCP. As imagens dos terminais estão organizadas em pares verticais: o terminal superior mostra o resultado do comando `tcpdump` na interface principal de cada *host*; o comando `netcat` (abreviado por `nc`) é utilizado no terminal inferior para fazer emitir pacotes TCP para os *hosts* da rede. Variações entre os parâmetros foram realizadas, a fim de que se pudesse verificar cada uma das regras definidas. O título do topo da janela dos terminais indica o *host* proprietário do respectivo terminal.

The figure displays eight terminal windows arranged in a 4x2 grid, showing network traffic and netcat commands across four hosts (h1, h2, h3, h4). The top row shows the initial state of the hosts, with h1 and h2 displaying network traffic and h3 and h4 displaying netcat commands. The bottom row shows the results of the netcat commands, with h1 and h2 displaying network traffic and h3 and h4 displaying netcat commands.

Host h1: Shows network traffic from 13:22:22 to 13:22:26. The traffic includes ICMP and UDP packets. The netcat command is used to connect to 10.0.0.2 on port 3002.

Host h2: Shows network traffic from 13:22:22 to 13:22:22. The traffic includes ICMP and UDP packets. The netcat command is used to connect to 10.0.0.1 on port 3002.

Host h3: Shows network traffic from 13:20:45 to 13:20:50. The traffic includes ICMP and UDP packets. The netcat command is used to connect to 10.0.0.3 on port 3002.

Host h4: Shows network traffic from 13:20:46 to 13:20:53. The traffic includes ARP and UDP packets. The netcat command is used to connect to 10.0.0.4 on port 3002.

Figura 6: Ambiente *Mininet* com 4 *hosts* comunicando-se através do *firewall* via UDP. As imagens dos terminais estão organizadas em pares verticais: o terminal superior mostra o resultado do comando `tcpdump` na interface principal de cada *host*; o comando `netcat` (abreviado por `nc`) é utilizado no terminal inferior para fazer pacotes UDP para os *hosts* da rede. Variações entre os parâmetros foram realizadas, a fim de que se pudesse verificar cada uma das regras definidas. O título do topo da janela dos terminais indica o *host* proprietário do respectivo terminal.

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

Resposta:

- **Máquina Host:** Ubuntu Desktop 20.04.3 LTS; Intel® Core™ i7-10510U CPU @ 1.80GHz × 8; 15,4 GiB de RAM.
- **Máquina Virtual (Guest):** Ubuntu 14.04.4 LTS; 4 VCPU; 8 GiB de RAM. *Hypervisor:* Oracle VM Virtual Box v. 6.1.26 r145957².

Todos experimentos e medições foram realizadas no ambiente virtualizado da **máquina guest**.

Seguindo uma sugestão presente em um comentário do código original do tutorial, a implementação do *switch* da Seção 7 foi configurado com `idle_timeout=100`, o que significa que um fluxo é removido da tabela do *switch*, caso permaneça ocioso por mais de 100 segundos. A influência desse parâmetro nos testes, consiste de um pequeno tempo adicional no início da simulação, requerido para recriar as tabelas de fluxo do *switch*.

Referências

- [1] Wikipedia, “OpenFlow”. , <https://en.wikipedia.org/wiki/OpenFlow#Description>
- [2] ipgeolocation, “IP Geolocation Information of 187.2.80.9”. , <https://ipgeolocation.io/ip-location/187.2.80.9>
- [3] ipgeolocation, “IP Geolocation Information of 189.23.245.1”. , <https://ipgeolocation.io/ip-location/189.23.245.1>
- [4] ipgeolocation, “IP Geolocation Information of 200.230.235.55”. , <https://ipgeolocation.io/ip-location/200.230.235.55>
- [5] ipgeolocation, “IP Geolocation Information of 200.230.220.154”. , <https://ipgeolocation.io/ip-location/200.230.220.154>
- [6] dbip, “200.230.220.154 - ebt-B11911-intl01.nyk.embratel.net.br”. , <https://db-ip.com/200.230.220.154>
- [7] IP2Location, “IP Address 200.230.220.154 Demo”. , <https://www.ip2location.com/demo/200.230.220.154>
- [8] ipgeolocation, “IP Geolocation Information of 209.120.132.221”. , <https://ipgeolocation.io/ip-location/209.120.132.221>
- [9] dbip, “209.120.132.221 - ebt-B11911-intl01.nyk.embratel.net.br”. , <https://db-ip.com/209.120.132.221>

²Imagem disponível em <https://github.com/mininet/openflow-tutorial/wiki/Installing-Required-Software>