Relatório do EP3
MAC0352 – Redes de Computadores e Sistemas Distribuídos – 2s2017

Rodrigo Ribeiro Santos de Carvalho (9298037)
Marcos Vinicius do Carmo Sousa (9298274)

1 Passo 1

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

O switch manda o pacote para o controlador. O controlador, por sua vez, pode descartar o pacote ou enviar para o switch uma nova regra de como tratar pacotes similares a este.

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  10.0.4.2 (10.0.4.2)
    5.885 ms  8.273 ms  5.964 ms
 2  gateway (172.20.10.1)
    11.670 ms  8.566 ms  10.109 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  et2-0-0.sanpaolo8.spa.seabone.net (149.3.181.17)
    571.064 ms  574.975 ms  570.802 ms
10  et9-3-0.miami15.mia.seabone.net (195.22.199.179)
    659.050 ms  658.977 ms  658.384 ms
11  gtt.miami15.mia.seabone.net (89.221.41.197)
    658.649 ms  658.579 ms  662.962 ms
12  xe-5-0-4.cr0-par7.ip4.gtt.net (141.136.105.218)
    772.865 ms  775.566 ms  775.295 ms
```

```
13  renater-gw-ix1.gtt.net (77.67.123.206)
    764.726 ms  764.635 ms  259.889 ms
14  193.51.177.107 (193.51.177.107)
    268.979 ms  268.021 ms  292.091 ms
15  inria-rocquencourt-te1-4-inria-rtr-021.noc.renater.fr (193.51.184.177)
    279.718 ms  284.169 ms  284.170 ms
16  unit240-reth1-vfw-ext-dc1.inria.fr (192.93.122.19)
    286.783 ms  285.851 ms  324.569 ms
17  ezp3.inria.fr (128.93.162.84)
    323.651 ms  318.527 ms  315.286 ms>
```

A partir do ponto 12 da rota o caminho é feito na europa.
Utilizando o comando whois no ponto 11, temos a seguinte saida:

```
<...
inetnum:        89.221.41.0 - 89.221.41.255
netname:        SEA-BONE
descr:          TI Sparkle Seabone Miami (US) POP
remarks:        INFRA-AW
country:        US
admin-c:        SEA6-RIPE
tech-c:         SEA6-RIPE
status:         ASSIGNED PA
remarks:        ================================================
remarks:        For Internet Abuse & Spam reports : abuse@seabone.net
remarks:        ================================================
mnt-by:         AS6762-MNT
created:        2007-07-12T07:21:02Z
last-modified:  2007-07-12T07:21:02Z
source:         RIPE
...>
```

Em country é possível ver US, indicando ser um roteador nos Estados Unidos.

Utilizando novamente o comando whois no ponto 12 (o próximo roteador).

```
<...
inetnum:        141.136.105.0 - 141.136.105.255
netname:        NACNET-EU
descr:          Tinet International Network
descr:          Hugenottenalle 167
descr:          63263 Neu-Isenburg
country:        DE
remarks:        INFRA-AW
admin-c:        SE33-RIPE
tech-c:         TIS333-RIPE
status:         ASSIGNED PA
mnt-by:         AS3257-NET-MNT
```

```
created:        2014-03-28T13:35:52Z
last-modified:  2014-03-28T13:35:52Z
source:         RIPE

role:           Tiscali International
address:        Tinet SpA
address:        Hugenottenallee 167
address:        63263 Neu-Isenburg
address:        Germany
...>
```

Em country temos DE, que indica Deutschland (Alemanha), além do endereço de registro ser "Germany" (Alemanha).

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).

```
<
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.67 Gbits/sec', '1.68 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.38 Gbits/sec', '1.38 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.64 Gbits/sec', '1.65 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.56 Gbits/sec', '1.57 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['1.44 Gbits/sec', '1.44 Gbits/sec']
>
------------------------------------------------
Média: 1.538 Gbits/sec
IC: 95% 1.428 - 1.648 Gbits/sec
```

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?

```
<
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['966 Kbits/sec', '1.03 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['972 Kbits/sec', '1.04 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['966 Kbits/sec', '1.03 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['971 Kbits/sec', '1.04 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['967 Kbits/sec', '1.03 Mbits/sec']
>
------------------------------------------------
Média: 968.4 Kbits/sec
IC: 95% 965.875 - 970.925 Kbits/sec

Corresponde a 1588.2 vezes menor que o da seção anterior

Utilizando o user-space switch, os pacotes devem cruzar do user-space
para o kernel-space e voltar a cada hop, ao invés de ficar no kernel como
no modo anterior, isso torna o user-space switch mais facil de modificar,
pois não precisa lidar com kernel, porem fica bem lento a taxa para a
simulação.
```

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.

<

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['8.89 Mbits/sec', '9.65 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.27 Mbits/sec', '9.97 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.44 Mbits/sec', '9.98 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['10.3 Mbits/sec', '10.9 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['10.8 Mbits/sec', '11.5 Mbits/sec']
>
```

Média: 9.94 Mbits / sec
Intervalo de confiança (95%): 9.16 - 10.72

Corresponde a 154.7 vezes menor que a seção 3.

A justificativa da diferença é que agora todo pacote que o switch recebe é mandado para o controlador. Na seção 3, o pacote era enviado apenas uma vez e o controlador estabelecia uma regra para o switch saber o que fazer quando um pacote similar chegasse até ele. Dessa forma, a transmissão entre os dois hosts ficava bem mais rápida.

Outro motivo da diferença é que como estamos trabalhando com um hub, todo pacote que o hub recebe é enviado para todo mundo. A execução do iperf, na verdade, apenas testa a velocidade entre dois hosts (no nosso caso h1 e h3). Assim, o h2 acaba recebendo os pacotes de h1 , usados para testar a velocidade e, dessa forma, a conexão se torna ligeiramente mais lenta.

Número de acotes captados durante a execução dos 5 iperf, de acordo com o tcpdump (para os hots) e wireshark (para o controlador nos hosts e no controlador:
h1: 16620
h2: 26333
h3: 26333
controlador: 13464

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.

```
<
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['10.4 Mbits/sec', '11.3 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['9.17 Mbits/sec', '9.72 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['10.9 Mbits/sec', '11.9 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['10.7 Mbits/sec', '11.4 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['11.2 Mbits/sec', '12.2 Mbits/sec']
>
```

```
  Média:  10.47 Mbits / sec
Intervalo de confiança (95%):  9.49 - 11.44
```

```
  Corresponde a 1.05 mais rápido que a seção anterior.
```

```
Agora o h2 não recebe mais os pacotes (na verdade o h2 recebe apenas um
pacote ARP broadcast), tornando a conexão ligeiramente mais rápida.
```

```
Número de pacotes captados durante a execução dos 5 iperf, de acordo com
o tcpdump (para os hosts) e wireshark (para o controlador, excluindo os
pacotes echo-request e echo-reply):
h1:  15747
h2:  1
h3:  24382
controlador:  13005
```

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.

```
<
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['302 Mbits/sec', '304 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['315 Mbits/sec', '315 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['309 Mbits/sec', '311 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['307 Mbits/sec', '308 Mbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['312 Mbits/sec', '314 Mbits/sec']
>
```

Média: 309 Mbits / sec
Intervalo de confiança (95%): 302.9 - 315.1

Obs: Como fizemos os testes usando o tcpdump ativo nos host e com o wireshark no controlador, a velocidade acaba diminuindo muito. Sem esses programas, a velocidade fica em torno de 2 Gbits / sec

Número de pacotes captados durante a execução dos 5 iperf, de acordo com o tcpdump (para os hots) e wireshark (para o controlador e excluindo os pacotes echo-request e echo-reply:
h1: 44758
h2: 2
h3: 44758
controlador: 48

Saída do sudo ovs-ofctl dump-flows s1:

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=7.754s, table=0, n_packets=0, n_bytes=0, idle_timeout=60
 cookie=0x0, duration=65.932s, table=0, n_packets=2, n_bytes=84, idle_timeout=
```

```
 cookie=0x0, duration=8.746s, table=0, n_packets=1, n_bytes=42, idle_timeout=6
 cookie=0x0, duration=8.748s, table=0, n_packets=1, n_bytes=42, idle_timeout=6
 cookie=0x0, duration=55.904s, table=0, n_packets=1, n_bytes=42, idle_timeout=
 cookie=0x0, duration=46.734s, table=0, n_packets=3, n_bytes=206, idle_timeout
 cookie=0x0, duration=47.72s, table=0, n_packets=4, n_bytes=272, idle_timeout=
 cookie=0x0, duration=64.947s, table=0, n_packets=3, n_bytes=294, idle_timeout
 cookie=0x0, duration=54.926s, table=0, n_packets=0, n_bytes=0, idle_timeout=6
 cookie=0x0, duration=13.743s, table=0, n_packets=2, n_bytes=196, idle_timeout
 cookie=0x0, duration=44.215s, table=0, n_packets=10690, n_bytes=705548, idle_
 cookie=0x0, duration=45.24s, table=0, n_packets=28153, n_bytes=1384929874, id
```

Percebemos que pouquissímos pacotes são enviados ao controlador. A diferença agora é que se um pacote é enviado ao controlador, ele irá enviar ao switch uma regra de como tratar pacotes similares a esse, evitando enviar muitos pacotes ao controlador e, portanto, elevando a velocidade de conexão.

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)

Primeiramente, o firewall lê as regras passadas por um arquivo e armazena essas regras numa lista de tuplas. Quando um pacote chega do switch, o firewall irá extrair as informações do pacote (IP's, portas, protocolo) e então começará a iterar sobre todas as regras para verificar se emparelha com alguma delas.

Se caso encontrar uma regra compatível, o pacote é ignorado e descartado. Caso contrário, o pacote segue adiante.

```
<
mininet> s1 cat pox/rules
10.0.0.1 10.0.0.3 -1 -1 tcp
10.0.0.3 10.0.0.2 -1 -1 -1
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 X
h3 -> h1 X
*** Results: 33% dropped (4/6 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
Waiting for iperf to start up...^C
Interrupt
mininet>
```

>

Acima, temos uma regra para bloquear todos os pacotes de origem 10.0.0.3 com destino 10.0.0.2, não importando se é TCP, UDP, ou ICMP. Com o teste do ping, percebemos que não foi trocado pacotes entre o host 1 e 2.

Outra regra bloqueia pacotes TCP entre com origem 10.0.0.1 e destino 10.0.0.3. Vemos seu funcionamento através do comando iperf (que utiliza TCP): não houve troca de pacotes por vários minutos (tivemos que dar um ctrl-C para matar o processo).

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

Configuração do computador virtual:
SO: Ubuntu 64-bits
Memória RAM: 1024 MB
Memória de Vídeo: 12 MB
Placa de rede (virtualizada): Intel PRO/1000 T Server

Configuração do computador real:
SO: Ubuntu 64-bits
Processador: Core i5
Memória RAM: 4 GB

10 Referências

- https://openflow.stanford.edu/display/ONL/POX+Wiki
-