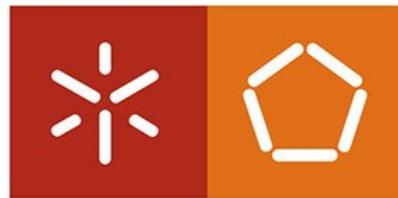


Redes Definidas por Software

2023/2024

Trabalho Prático 3
Grupo 7

Control Plane - P4Runtime



Universidade do Minho
Escola de Engenharia

André Alves^[pg53651], Renato Gomes^[pg54174], and Afonso Marques^[pg53601]



¹ Universidade do Minho, Braga, Portugal
² www.uminho.pt/PT

Abstract. No campo da programação de redes, o plano de Controlo serve como o cérebro que orquestra o comportamento e a funcionalidade do plano de dados subjacente. Tecnologias pioneras como o P4Runtime revolucionam a forma como interagimos e gerimos dispositivos de rede, oferecendo uma flexibilidade e programabilidade sem precedentes. Este trabalho descreve os passos envolvidos na atualização do ambiente de trabalho para suportar o P4, garantindo compatibilidade e funcionalidade para o desenvolvimento com P4Runtime.

Keywords: P4 · OVSKernelSwitch · Router · P4Runtime · Controller.

1 Introdução

Neste terceiro e ultimo trabalho, o objetivo é expandir o conhecimento e habilidades adquiridos nos trabalhos anteriores de forma a desenvolver um controlador P4Runtime. O objetivo principal é fazer a transição da injeção manual de regras via *simple_switch_CLI* para o controlo dinâmico e programável de dispositivos P4 usando P4Runtime. O controlador criado terá de ser capaz de gerir a topologia de rede mais complexa desenvolvida ao longo do semestre, que se segue na imagem em baixo:

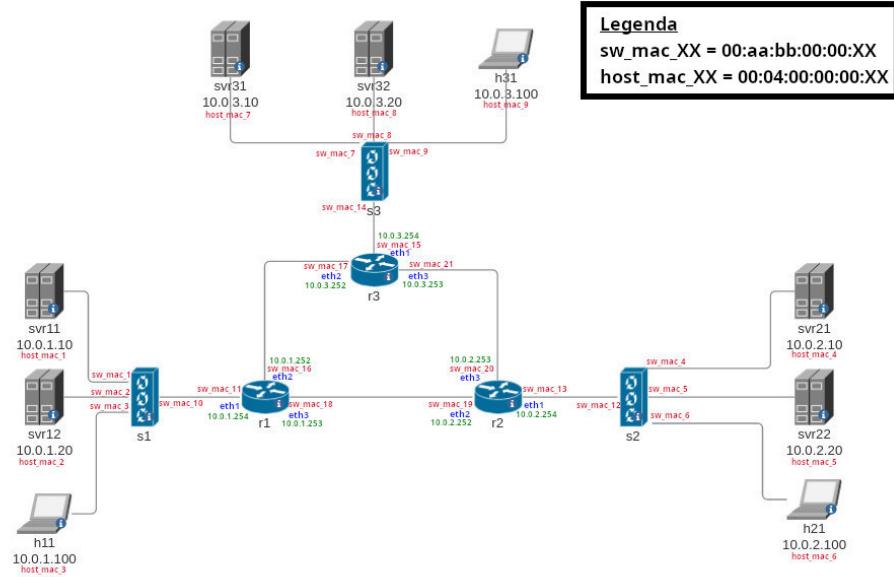


Fig. 1: Topologia em uso

2 Desenvolvimento

2.1 A topologia

O cenário em estudo mantém-se o mesmo dos trabalhos anteriores. Consiste numa rede corporativa dividida em três LANs: Sales (LAN1), Research and Development (LAN2) e Management (LAN3). Cada LAN possui dois servidores e um computador pessoal conectados a switches Open vSwitch (OVS), que por sua vez estão conectados a routers P4 BMv2. Os hosts têm endereços IP no formato 10.0.XX.XX e endereços MAC no formato 00:04:00:00:00, conectados ao switch mais próximo. Cada router possui um porto thrift distinto: r1 com 9090, r2 com 9091 e r3 com 9092.

A única alteração relevante, mas sem impacto notável foi a mudança de classe utilizada nos switchs s1, s2 e s3, que passaram de OVSSwitch para OVSKernel-Switch. Os routers r1, r2 e r3 também foram alterados, passando de P4Switch para P4RuntimeSwitch, sendo-lhes também adicionado um identificador de dispositivo único, uma *grpc-port* para comunicação com o controlador e um *cpu-port*.

(...)

```
r1 = self.addSwitch('r1',
                     cls = P4RuntimeSwitch,
                     sw_path = sw_path,
                     #json_path = json_path,
                     thrift_port = thrift_port,
                     grpc_port = grpc_port,
                     device_id = 1,
                     cpu_port = 510)
```

(...)

2.2 Implementação do controlador P4Runtime

A implementação do controlador P4Runtime consistiu na introdução de um ficheiro novo, o TP3-controller.py, sendo que o ficheiro router_tp3.p4 se manteve com o código herdado do TP2. O ficheiro do controlador contem informação relativa a todas as regras que vão ser injetadas nos routers. Contem informação relativa ao mapeamento de endereços MAC para cada porta de cada router através do seguinte conjunto de instruções:

```
port_mac_mapping_r1 = 1: "00:aa:bb:00:00:11",
2: "00:aa:bb:00:00:16", 3: "00:aa:bb:00:00:18"
```

```
port_mac_mapping_r2 = 1: "00:aa:bb:00:00:13",
2: "00:aa:bb:00:00:19", 3: "00:aa:bb:00:00:20"
```

```
port_mac_mapping_r3 = 1: "00:aa:bb:00:00:15",
2: "00:aa:bb:00:00:17", 3: "00:aa:bb:00:00:21"
```

Para determinadas tabelas definidas no router_tp3.p4, existe uma função no controlador que trata de criar e escrever uma entrada nessa mesma tabela. Por exemplo, as regras de forwarding, aplicadas nas tabelas *ipv4_lpm* e *dst_mac*, são tratadas pela função *writeFwdRules* do controlador, que recebe como parâmetros o dispositivo, o endereço de destino, a máscara aplicada, o próximo salto, a porta de saída e o MAC de destino.

Com esta nova forma de aplicar restrições, foi necessário que os três conjuntos de regras que inicialmente existiam nos ficheiros de commands fossem convertidos para funcionar dentro do controlador.

Regras Forwarding e Firewall

Relativamente às regras para ser realizado o forwarding de mensagens e para a criação da nossa firewall, foi feita basicamente uma conversão direta entre o exemplo que existia no código providenciado com base na topologia adjacente e com as regras que nós tínhamos anteriormente, pois a lógica mantinha-se, apenas sendo alterada a sintaxe delas.

Segue abaixo um exemplo da conversão das regras:

Regra de forwarding:

- **antes** : table_add ipv4_lpm ipv4_fwd 10.0.1.10/32 => 10.0.1.10 1
table_add dst_mac rewrite_dst_mac 10.0.1.10 => 00:04:00:00:00:01
- **depois** : writeFwdRules(p4info_helper, r1, "10.0.1.10", 32, "10.0.1.10", 1, "00:04:00:00:00:01")

Regra de firewall:

- **antes** : table_add firewall drop 10.0.2.0/24 10.0.1.10 0x06 0->65535 =>1
- **depois** : writeFirewallRules(p4info_helper, r1, "10.0.2.0", 24, "10.0.1.10", 6, [0,50])
writeFirewallRules(p4info_helper, r1, "10.0.2.0", 24, "10.0.1.10", 6, [51,65535])

É de notar que na conversão que fizemos nas regras de firewall tivemos um problema relativamente ao intervalo de portas que queríamos testar. Por uma razão desconhecida, não é aceite um intervalo de portas que as abranja na totalidade, ou seja, não é possível colocarmos um intervalo de [0, 65535]. Consoante isto decidimos partir a regra em duas num número arbitrário, fazendo com que desta forma sejam abrangidas a gama de portas na sua totalidade, como é possível ver acima.

Regras de Firewall_2

Já em relação à nossa Firewall_2, tal que tem o trabalho de permitir passar os pacotes consoante o seu protocolo, tivemos de realizar algumas modificações para estas funcionarem. Como nós originalmente detínhamos tanto regras para não deixar passar e para não tomar qualquer ação, isto demonstrou-se ser um problema para a conversão delas. Com isto decidimos alterar as regras que nós tínhamos onde não tomávamos ação para regras para não permitir tráfego. A baixo é possível ver como foi feita esta conversão:

Regra original de bloqueio:

- ***antes*** : table_add firewall_2 drop 10.0.2.0/24 10.0.1.0->10.0.1.251 0x00->0x05
=> 2
table_add firewall_2 drop 10.0.2.0/24 10.0.1.0->10.0.1.251 0x07->0xFF =>
2
- ***depois*** : writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.0", 24, ["10.0.1.0",
"10.0.1.99"], [0, 5])
writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.0", 24, ["10.0.1.0", "10.0.1.99"],
[7, 255])
writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.0", 24, ["10.0.1.101", "10.0.1.251"],
[0, 5])
writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.0", 24, ["10.0.1.101", "10.0.1.251"],
[7, 255])

Regra de não atuação:

- ***antes*** : table_add firewall_2 NoAction 10.0.2.254/32 10.0.1.100->10.0.1.100
0x01->0x01 => 1
- ***depois*** : writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.254", 32, ["10.0.1.100",
"10.0.1.100"], [0, 0])
writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.254", 32, ["10.0.1.100", "10.0.1.100"],
[2, 5])
writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.254", 32, ["10.0.1.100", "10.0.1.100"],
[7, 255])

```

writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.100", 32, ["10.0.1.100", "10.0.1.100"],
[0, 5])
writeFirewall_2_Rules(p4info_helper, r1, "10.0.2.100", 32, ["10.0.1.100", "10.0.1.100"],
[7, 255])

```

Como é possível ver acima, a conversão da regra de não atuação foi particularmente verbosa. Basicamente foi feito que ao invés de permitir que um pacote com um certo protocolo, neste caso 1, pudesse passar, bloqueámos todos os que não sejam o 1. De notar que também não bloqueamos para o protocolo 6, isto acontece, pois com a lógica que tínhamos anteriormente se bloqueássemos para o protocolo 6 iríamos andar a bloquear todo o TCP de poder chegar a estes dispositivos. Para isto voltar a funcionar tivemos então que dividir a de bloqueio de tráfego para poder manipular os dispositivos fora da gama que poderíamos afetar, tornando esta também um pouco mais verbosa.

Regras de ICMP

Já para as regras de ICMP estas foram também relativamente fáceis de se converterem, sendo a conversão relativamente direta, como é possível ver abaixo:

- *antes* : table.add icmp_for_router respond_icmp 10.0.1.254 0x01 =>
- *depois* : writeICMP_Rules(p4info_helper, r1, "10.0.1.254", 1)

2.3 Testes Realizados E Discussão

O seguinte conjunto de testes serve para comprovar que a transição de injeção manual para um controlador não implica que a firewall e respostas ICMP deixem de funcionar. No fundo, quer-se mostrar que o comportamento da rede se manteve dentro do esperado.

Teste de Firewall

Primeiramente queremos garantir que a firewall continua a funcionar. Para isso vai-se testar algumas das regras que foram definidas no TP1. Para diferentes casos no host h11, os resultados foram os seguintes. Foi utilizado o comando nping, que só necessita de um terminal, evitando assim o uso de iperf, que necessita de dois.

No h11 tentamos aceder aos locais onde lhe é permitido, obtendo sucesso em todos os casos. Adicionalmente, ao aceder a locais proibidos, como, por exemplo, destinos com um IP, protocolo ou porta proibida, o resultado é o esperado, não havendo comunicação.

```
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3# nping -tcp -p 80 10.0.2.10
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-06-11 19:28 BST
SENT (0.0178s) TCP 10.0.1.100:37412 > 10.0.2.10:80 S ttl=64 id=65516 iplen=40 seq=2747173296 win=1480
RCVD (0.0222s) TCP 10.0.2.10:80 > 10.0.1.100:37412 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (1.0197s) TCP 10.0.1.100:37412 > 10.0.2.10:80 S ttl=64 id=65516 iplen=40 seq=2747173296 win=1480
RCVD (1.0237s) TCP 10.0.2.10:80 > 10.0.1.100:37412 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (2.0222s) TCP 10.0.1.100:37412 > 10.0.2.10:80 S ttl=64 id=65516 iplen=40 seq=2747173296 win=1480
RCVD (2.0245s) TCP 10.0.2.10:80 > 10.0.1.100:37412 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (3.0238s) TCP 10.0.1.100:37412 > 10.0.2.10:80 S ttl=64 id=65516 iplen=40 seq=2747173296 win=1480
RCVD (3.0253s) TCP 10.0.2.10:80 > 10.0.1.100:37412 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (4.0259s) TCP 10.0.1.100:37412 > 10.0.2.10:80 S ttl=64 id=65516 iplen=40 seq=2747173296 win=1480
RCVD (4.0299s) TCP 10.0.2.10:80 > 10.0.1.100:37412 RA ttl=62 id=0 iplen=40 seq=0 win=0

Max rtt: 4.389ms | Min rtt: 1.331ms | Avg rtt: 3.061ms
Raw packets sent: 5 (200B) | Rcvd: 5 (200B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 4.08 seconds
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3#
```

Fig. 2: h11, TCP, porta 80, svr21

```
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3# nping -tcp -p 8080 10.0.3.10
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-06-11 19:29 BST
SENT (0.0275s) TCP 10.0.1.100:10538 > 10.0.3.10:8080 S ttl=64 id=4130 iplen=40 seq=3037743646 win=1480
RCVD (0.0323s) TCP 10.0.3.10:8080 > 10.0.1.100:10538 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (1.0340s) TCP 10.0.1.100:10538 > 10.0.3.10:8080 S ttl=64 id=4130 iplen=40 seq=3037743646 win=1480
RCVD (1.0355s) TCP 10.0.3.10:8080 > 10.0.1.100:10538 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (2.0432s) TCP 10.0.1.100:10538 > 10.0.3.10:8080 S ttl=64 id=4130 iplen=40 seq=3037743646 win=1480
RCVD (2.0473s) TCP 10.0.3.10:8080 > 10.0.1.100:10538 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (3.0506s) TCP 10.0.1.100:10538 > 10.0.3.10:8080 S ttl=64 id=4130 iplen=40 seq=3037743646 win=1480
RCVD (3.0534s) TCP 10.0.3.10:8080 > 10.0.1.100:10538 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (4.0521s) TCP 10.0.1.100:10538 > 10.0.3.10:8080 S ttl=64 id=4130 iplen=40 seq=3037743646 win=1480
RCVD (4.0552s) TCP 10.0.3.10:8080 > 10.0.1.100:10538 RA ttl=62 id=0 iplen=40 seq=0 win=0

Max rtt: 4.706ms | Min rtt: 1.384ms | Avg rtt: 3.020ms
Raw packets sent: 5 (200B) | Rcvd: 5 (200B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 4.09 seconds
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3#
```

Fig. 3: h11, TCP, porta 8080, svr31

```
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3# nping -tcp -p 8080 10.0.3.20
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-06-11 19:30 BST
SENT (0.0432s) TCP 10.0.1.100:13647 > 10.0.3.20:8080 S ttl=64 id=6660 iplen=40 seq=3685080924 win=1480
SENT (1.0448s) TCP 10.0.1.100:13647 > 10.0.3.20:8080 S ttl=64 id=6660 iplen=40 seq=3685080924 win=1480
SENT (2.0454s) TCP 10.0.1.100:13647 > 10.0.3.20:8080 S ttl=64 id=6660 iplen=40 seq=3685080924 win=1480
SENT (3.0507s) TCP 10.0.1.100:13647 > 10.0.3.20:8080 S ttl=64 id=6660 iplen=40 seq=3685080924 win=1480
SENT (4.0551s) TCP 10.0.1.100:13647 > 10.0.3.20:8080 S ttl=64 id=6660 iplen=40 seq=3685080924 win=1480

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 5 (200B) | Rcvd: 0 (0B) | Lost: 5 (100.00%)
Nping done: 1 IP address pinged in 5.10 seconds
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3#
```

Fig. 4: h11, TCP, porta 8080, svr32

Na figura 2 e 3 comprova-se que as regras da LAN 1 onde h11 pode comunicar em TCP nas devidas portas para svr21 e svr31 estão a funcionar, pois existe

resposta. Na figura 4 tentamos comunicar com um IP proibido para a LAN 1 (o svr32) e, como esperado, não se recebeu resposta do outro lado.

Já provamos que as restrições para a comunicação em TCP funcionam. Falta agora provar que qualquer outro tipo de comunicação que não seja TCP é proibido entre as três LANs. Para esse efeito, fizemos os seguintes comandos:

```
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3# nping -udp -p 8080 10.0.3.10
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-06-11 19:32 BST
SENT (0.0271s) UDP 10.0.1.100:53 > 10.0.3.10:8080 ttl=64 id=61791 iplen=28
SENT (1.0349s) UDP 10.0.1.100:53 > 10.0.3.10:8080 ttl=64 id=61791 iplen=28
SENT (2.0360s) UDP 10.0.1.100:53 > 10.0.3.10:8080 ttl=64 id=61791 iplen=28
SENT (3.0541s) UDP 10.0.1.100:53 > 10.0.3.10:8080 ttl=64 id=61791 iplen=28
SENT (4.0545s) UDP 10.0.1.100:53 > 10.0.3.10:8080 ttl=64 id=61791 iplen=28

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 5 (140B) | Rcvd: 0 (0B) | Lost: 5 (100.00%)
Nping done: 1 IP address pinged in 5.11 seconds
```

Fig. 5: h11, UDP, porta 8080, svr31

```
root@XubuntuRDS:/home/afonso/Desktop/RDS-G07-TP3/code_TP3# ping 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
```

Fig. 6: h11, ICMP, h21

Novamente obteve-se sucesso. A perda dos pacotes indica que as regras da firewall estão a ser respeitadas, sendo unicamente possível o tráfego em TCP e este tem de ocorrer em determinadas portas e endereços permitidos.

Teste de ICMP

Falta agora testar se os routers continuam a ser capazes de responder a pedidos ICMP aos endereços atribuídos a cada uma das suas interfaces. No teste anterior já vimos que não é possível fazer ping a um host. Vamos agora tentar comunicar em ICMP com a interface de um dos routers e um servidor de uma das LANs.

```
mininet> h21 ping svr11
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
```

Fig. 7: h21, ICMP, svr11

```
mininet> h21 ping 10.0.3.252
PING 10.0.3.252 (10.0.3.252) 56(84) bytes of data.
64 bytes from 10.0.3.252: icmp_seq=1 ttl=61 time=1.86 ms
64 bytes from 10.0.3.252: icmp_seq=2 ttl=61 time=1.43 ms
64 bytes from 10.0.3.252: icmp_seq=3 ttl=61 time=2.31 ms
64 bytes from 10.0.3.252: icmp_seq=4 ttl=61 time=3.25 ms
64 bytes from 10.0.3.252: icmp_seq=5 ttl=61 time=2.09 ms
^C
--- 10.0.3.252 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4025ms
rtt min/avg/max/mdev = 1.434/2.186/3.246/0.603 ms
mininet>
```

Fig. 8: h21, ICMP, r3-eth2

Os resultados demonstram que a manipulação dos pacotes ICMP continua a ser aplicada apenas em casos específicos, tal como é o pretendido.

2.4 Dificuldades

Neste trabalho as dificuldades foram as mesmas encontradas no trabalho anterior. Com isto referimo-nos à forma como as regras de firewall foram especificadas bem como a maneira como o pacote ICMP é manipulado no que diz respeito ao checksum.

No que diz respeito apenas ao TP3, a conversão das regras da tabela auxiliar *firewall_2* foram certamente o maior desafio. Enquanto que todas as outras tabelas apenas lidam com uma única prioridade e uma única ação (*ipv4.lpm*, *dst_mac*, *firewall* e *icmp_for_router*), a tabela *firewall_2*, que lida com a exlusão de todos os protocolos que não sejam TCP entre LANs, tem ações diferentes e prioridades diferentes para diferentes conjuntos de argumentos.

Inicialmente tentou-se fazer a conversão direta, através da aplicação de condições que, dependendo da máscara aplicada ao endereço IP de origem, permitiria fazer

a distinção entre uma regra cuja ação deveria ser NoAction ou drop, onde a prioridade das regras com NoAction deveria ser maior. No entanto, esta estratégia resultou em insucesso, devido a erros de compilação do controlador.

Decidiu-se então partir novamente para uma estratégia mais verbosa, onde apenas se manteve a regra drop e todas as entradas da tabela *firewall_2* que faziam uso de NoAction tiveram de ser reescritas de forma a suportar a ação drop. Isto resultou num aumento de dez regras de *firewall_2* no TP2 para trinta regras de *firewall_2* no TP3.

3 Conclusão

Neste relatório, focamo-nos na implementação de um controlador capaz de programar routers com a capacidade de responder a pedidos ICMP e de impedir certos pacotes de chegarem a certos destinos (firewall). Este exercício proporcionou uma compreensão prática e aprofundada sobre a construção de planos de controlo e a sua integração com o plano de dados em ambientes SDN.

Através deste exercício, adquirimos competências essenciais em P4 e SDN, compreendendo não apenas os aspectos técnicos, mas também a importância de soluções escaláveis e seguras em redes modernas. Em suma, este terceiro trabalho prático (TP3) demonstrou como a programação de redes com P4 e P4Runtime pode ser uma ferramenta poderosa para o desenvolvimento de infraestruturas de rede adaptáveis e eficientes, preparando-nos para futuros desafios no campo das redes definidas por software.

References

1. [https://en.wikipedia.org/wiki/P4_\(programming_language\)](https://en.wikipedia.org/wiki/P4_(programming_language))
2. <https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html>
3. <https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml>
4. <https://pt.wikipedia.org/wiki/Ping>
5. <https://nmap.org/nping/>