

# **Redes Definidas por Software**

## **2023/2024**

### **Trabalho Prático 1**

#### **Grupo 7**

**Implementing a L3 L4 Stateful Firewall with P4**



**Universidade do Minho**  
Escola de Engenharia

André Alves<sup>[pg53651]</sup>, Renato Gomes<sup>[pg54174]</sup>, and Afonso Marques<sup>[pg53601]</sup>



<sup>1</sup> Universidade do Minho, Braga, Portugal  
<sup>2</sup> [www.uminho.pt](http://www.uminho.pt)/PT

**Abstract.** Redes Definidas por Software (SDN) revolucionaram a forma como as redes são projetadas, implementadas e geridas. Em vez de depender de hardware proprietário e estático para encaminhar o tráfego, SDN separa o plano de controlo do plano de dados, permitindo uma abordagem mais programável e dinâmica para a gestão de redes. O P4 (Programming Protocol-independent Packet Processors) é uma linguagem de programação projetada para definir a forma como os dispositivos de rede, como switches e routers, processam pacotes de dados. Com P4, é possível especificar o comportamento de encaminhamento de pacotes em um nível granular, permitindo a implementação de funcionalidades personalizadas e adaptáveis em dispositivos de rede. O Mininet é uma plataforma de emulação de rede de código aberto que permite criar redes virtuais de teste em um único computador, o que simplifica o processo de simulação de topologias de rede complexas, permitindo que se experimente e teste diversos cenários de rede de forma rápida e eficiente.

**Keywords:** P4 · OVSSwitch · Router · Host · LAN.

## 1 Introdução

Neste primeiro trabalho, o objetivo é implementar uma firewall com estado (stateful) de Camada 3 (L3) e Camada 4 (L4) utilizando a linguagem P4 (Programming Protocol-independent Packet Processors). O cenário envolve uma rede corporativa com três LANs: Sales (LAN1), Research And Development (R&D) (LAN2) e Management (LAN3). Cada LAN possui o seu próprio conjunto de servidores e hosts conectados a switches Open vSwitch (OVS), e os switches OVS estão conectados a routers P4 BMv2 (modelo comportamental v2). A seguinte topologia foi usada no decorrer do trabalho:

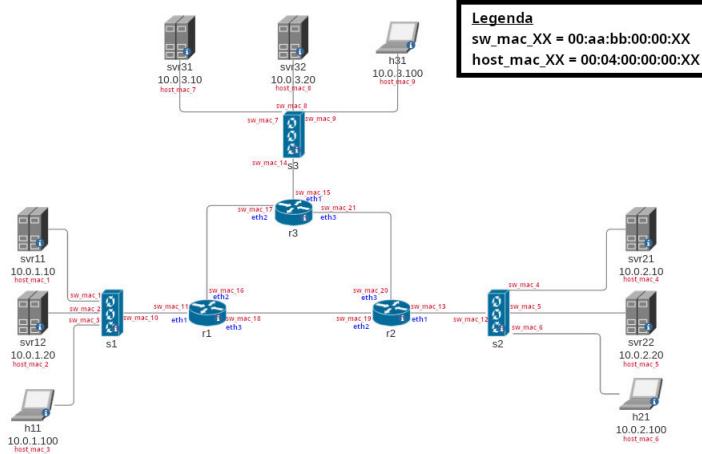


Fig. 1: Topologia utilizada

## 2 Desenvolvimento

### 2.1 O cenário em estudo

O cenário em estudo, tal como já foi referido, consiste numa rede corporativa dividida em três redes de área local (LANs) que consistem em Sales (LAN1), Research And Development (R&D) (LAN2) e Management (LAN3). Cada LAN possui um conjunto de dois servidores e um computador pessoal (host) conectados a switches Open vSwitch (OVS), sendo que os switches estão conectados a routers P4 BMv2 (modelo comportamental v2).

De forma a limitar certos contactos indesejados ou desnecessários, foi especificado que esta rede deve obedecer a um conjunto de regras cujo cumprimento deve ser garantido pela implementação de uma firewall. As regras são as seguintes:

- Sales (LAN 1):

1. Necessita de acesso ao sistema de CRM (10.0.1.10) na porta TCP 443;
2. Necessita de acesso ao servidor de E-Mail (10.0.1.20) na porta TCP 25 e 587;
3. Deve ter acesso limitado à LAN2 e à LAN3 para serviços específicos:
  - Acesso ao Servidor de Pesquisa 1 (10.0.2.10) na porta TCP 80;
  - Acesso ao Servidor de Dados Financeiros 1 (10.0.3.10) na porta TCP 8080;

- R&D (LAN 2):

1. Requer acesso aos seus servidores de pesquisa dedicados (intervalo de IP: 10.0.2.0/24);
2. Deve ter acesso limitado à LAN1 e LAN3 para serviços específicos:
  - Acesso ao Servidor de E-mail (10.0.1.20) na porta TCP 25;
  - Acesso ao Servidor de Dados Financeiros 2 (10.0.3.20) na porta TCP 443;

- Management (LAN 3):

1. Precisa de acesso seguro aos servidores de dados financeiros (10.0.3.10, 10.0.3.20) na porta TCP 8080;
2. Acesso seguro aos sistemas administrativos (10.0.3.100) na porta TCP 22 para SSH.
3. Deve ter acesso limitado à LAN1 e LAN2 para serviços específicos:
  - Acesso ao sistema de CRM (10.0.1.10) na porta TCP 443;
  - Acesso ao Servidor de Pesquisa 2 (10.0.2.20) na porta TCP 22.

## 2.2 Implementação da topologia

A criação do cenário virtual consistiu na concessão de um ficheiro escrito na linguagem Python, TP1-topo.py, cujo código é usado para criar uma topologia de rede em Mininet, usando as seguintes importações essenciais, como Mininet, Topo, setLogLevel, info, CLI, OVSSwitch, P4Switch e P4Host.

Dentro deste ficheiro, há uma classe denominada *SingleSwitchTopo*, que serve para criar uma topologia de rede simples com switchs P4, switchs OVS e hosts. Os hosts criados consistem nos computadores e servidores da topologia (ver figura 1). A sua configuração é muito simples, sendo apenas definido qual o seu endereço IP, no formato 10.0.XX.XX, e o seu endereço MAC com o formato 00:04:00:00:00:XX. De seguida cria-se a ligação entre o host e o switch mais próximo, ou seja aquele que fica apenas a um salto de distância, usando sempre um endereço MAC específico para cada porta onde um host está ligado.

Por exemplo, o host svr11 tem o IP 10.0.1.10 com MAC 00:04:00:00:00:01 (host\_mac\_1) e está ligado ao switch s1 na interface 00:aa:bb:00:00:01 (sw\_mac\_1).

```
( ... )

sw_mac_base = "00:aa:bb:00:00:%02x"
host_mac_base = "00:04:00:00:00:%02x"

host1 = self.addHost('svr11',
                     ip = "10.0.1.10/24",
                     mac = host_mac_base % (1))

sw_mac1 = sw_mac_base % (1)
self.addLink(host1, s1, addr2=sw_mac1)

( ... )
```

Nesta classe também se definem os routers e switchs que existem na topologia, sendo que cada um tem uma especificação diferente. Para cada router foi especificado que se deve usar a classe P4Switch, fornecida pela equipa docente, e qual o *thrift port* que se deve usar para se poder injetar as regras. Cada router possui um *thrift port* distinto, sendo que r1 fica com 9090, r2 com 9091 e r3 com 9092. A injeção das regras nos routers é feita através de um script bash, routers.sh, que permite automatizar esse processo. Este script deve ser executado dentro da diretoria commands do projeto.

Os switchs são a entidade que possui a menor quantidade de configuração, sendo apenas dito que devem usar a classe importada OVSSwitch, e que devem permitir a passagem de tráfego em todas as suas portas, independentemente da

origem. O script switchs.sh permite automatizar este processo de configuração dos flows, devendo ser corrido dentro da diretoria commands do projeto.

Finalmente a função main inicia a topologia definida e configura as ARP tables e rotas padrão para os hosts, bem como as gateways de cada um.

### 2.3 Implementação da firewall

De forma a implementar a firewall, começamos por proceder à criação de um ficheiro em P4, router\_with\_firewall.p4, que especifica o que acontece a cada pacote que é transmitido na rede Mininet. Aqui temos definidas estruturas que permitem armazenar o conteúdo extraído de cada pacote, como por exemplo a estrutura *ipv4\_t* que permite saber qual é o endereço IPv4 de origem e de destino bem como o protocolo usado.

( ... )

```
header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

( ... )

Extraindo o protocolo conseguimos também saber qual a porta de destino usada através da estrutura *tcp\_t* e do campo *dstPort* com 16 bits.

( ... )

```
header tcp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
```

```

    bit<4> dataOffset; // how long the TCP header is
    bit<3> res;
    bit<3> ecn;           // Explicit congestion notification
    bit<6> ctrl;          // URG,ACK,PSH,RST,SYN,FIN
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}

( ... )

```

Uma das principais funções deste ficheiro P4 é a definição de tabelas que permitem aplicar ações aos pacotes quando determinadas regras são cumpridas. As tabelas definidas para o propósito deste trabalho consistem em:

- **ipv4\_lpm**

Esta tabela tem como propósito auxiliar o encaminhamento dos pacotes na rede. Tem como ação padrão o drop, e as regras aplicadas especificam que a ação ipv4\_fwd deve ser aplicada sempre que ocorre uma correspondência na tabela. Por exemplo no router r1, que faz fronteira com a LAN 1, a regra

```
table_add ipv4_lpm ipv4_fwd 10.0.2.20/32 => 10.0.2.254 3
```

especifica que qualquer pacote que entre em r1 com destino ao endereço 10.0.2.20 deve ter como próximo salto o endereço 10.0.2.254 (router r2) e deve sair pela porta 3 do router r1. Esta manipulação é realizada pela ação ipv4\_fwd.

- **src\_mac**

Nesta tabela atribuímos um endereço MAC a todas as portas de um router. Por exemplo no router r1, a seguinte regra

```
table_add src_mac rewrite_src_mac 3 => 00:aa:bb:00:00:18
```

indica que a interface eth3 na porta 3 do router r1 tem o endereço MAC 00:aa:bb:00:00:18. Este endereço MAC irá ser a origem do pacote para o próximo salto.

- **dst\_mac**

Finalmente terminamos o processo de encaminhamento com a aplicação desta tabela. Aqui vamos indicar para qual interface o pacote se deve dirigir tendo em conta o IP de destino que possui e o MAC de origem obtido na tabela anterior. A seguinte regra aplicada em r1

```
table_add dst_mac rewrite_dst_mac 10.0.2.254
=> 00:aa:bb:00:00:19
```

diz que se o destino for o IP 10.0.2.254 (ou seja a LAN 2), deve então deslocar-se para a interface com endereço MAC igual a 00:aa:bb:00:00:19.

- **firewall**

Após o encaminhamento estar feito focamos-nos agora na aplicação das regras de exclusão da firewall. Esta tabela recebe quatro chaves, sendo elas o endereço de origem, o endereço de destino, o protocolo usado para o transporte e a porta TCP usada para a comunicação que são obtidos através do parse do conteúdo do pacote. Assim conseguimos ter a informação suficiente para construir as regras que vão ser injetadas na nossa firewall. As regras específicas da tabela da firewall seguem o formato abaixo

```
table_add firewall drop <ip LAN>/24 <ip destino>
0x06 <lim inf port> -> <lim sup port> => 1
```

onde o "ip LAN" pode ser 10.0.1.0, 10.0.2.0 ou 10.0.3.0; o "ip destino" é qualquer endereço IP que exista na rede, idealmente será o IP de um dos hosts; e os "lim inf port" e "lim sup port" definem o intervalo de portas onde se aplica a regra. Todas as regras da firewall servem para negar a passagem de tráfego em casos específicos, pelo que a ação usada nas regras injetadas é drop e a ação padrão é NoAction.

O exemplo em baixo é uma das regras da firewall onde o que se pretende fazer é indicar ao router r1 que este deve impedir qualquer tráfego em TCP com origem na LAN2, destino 10.0.1.20 (svr12) e que esteja no intervalo de portas 0 a 24 e 26 a 65535 (valor máximo permitido em 16 bits). Assim a porta 25 é a única permitida para tráfego TCP entre estes dois hosts.

```
table_add firewall drop 10.0.2.0/24 10.0.1.20
0x06 0->24 => 1
table_add firewall drop 10.0.2.0/24 10.0.1.20
0x06 26->65535 => 1
```

- **firewall\_2**

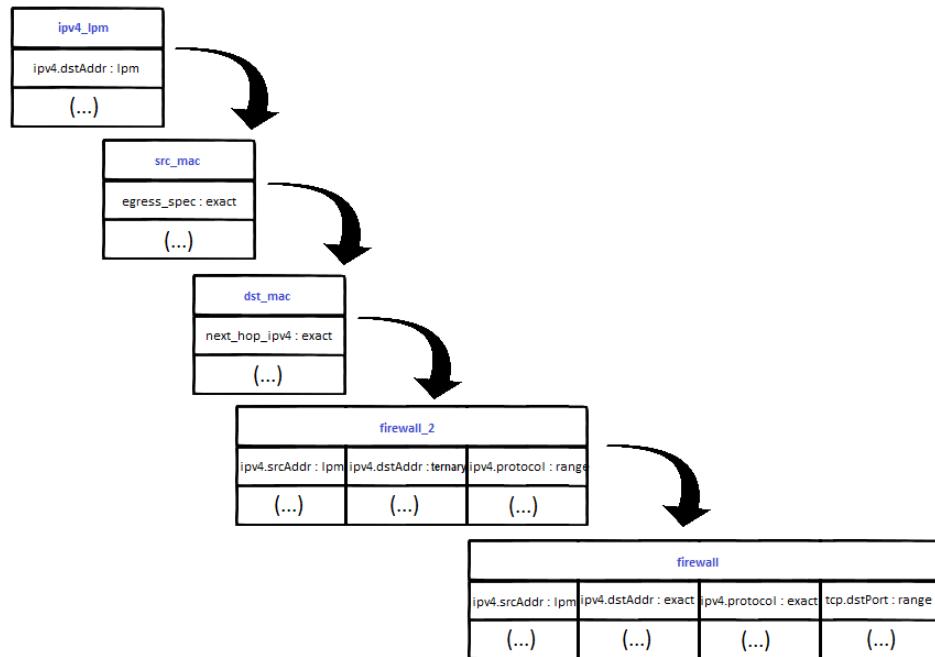
Esta tabela é uma extensão da firewall. Enquanto que a primeira apenas aplica exclusões ao tráfego em TCP, esta segunda tabela serve para impedir todo o tráfego entre LANs que esteja no range dos protocolos especificados nela, onde para o nosso caso apenas permitimos passar apenas o que seja TCP. Um exemplo de uma regra segue em baixo:

```
table_add firewall_2 drop 10.0.2.0/24 10.0.1.0&&255.255.255.0
0x00->0x05 => 1
```

```
table_add firewall_2 drop 10.0.2.0/24 10.0.1.0&&&255.255.255.0
0x07->0xFF => 1
```

Neste exemplo de firewall\_2, basicamente estamos a dizer ao router r1 que qualquer tráfego proveniente de LAN2, com destino a qualquer endereço compreendido entre 10.0.1.0 a 10.0.1.255 (ou seja a LAN1) e cujo valor em bits para o protocolo esteja entre 0x00 a 0x05 ou 0x07 a 0xFF deve ser impedido. Assim, o único valor permitido para o campo do protocolo na trama recebida é 0x06, que corresponde ao valor de TCP.

A seguinte imagem traduz a ordem pela qual aplicamos as tabelas:



Ou seja primeiro aplicamos as regras de encaminhamento para os pacotes e só depois aplicamos as regras de exclusão de tráfego da firewall.

## 2.4 Testes Realizados E Discussão

De forma a garantir que a implementação da firewall proposta funciona, realizamos um conjunto de testes.

## 2.5 Teste de Comunicação Intra-LAN

Primeiramente queremos garantir que todos os dispositivos dentro de uma LAN têm comunicação assegurada entre si, independentemente do protocolo usado ou porta usada. Sendo assim, este teste tem a finalidade de comprovar que a configuração nos switches foi feita corretamente.

Iremos fazer o teste na LAN 2. A seguinte configuração foi usada no switch s2, onde é possível ver que para qualquer uma das quatro portas de entrada, existe a possibilidade de sair por uma das três portas restantes.

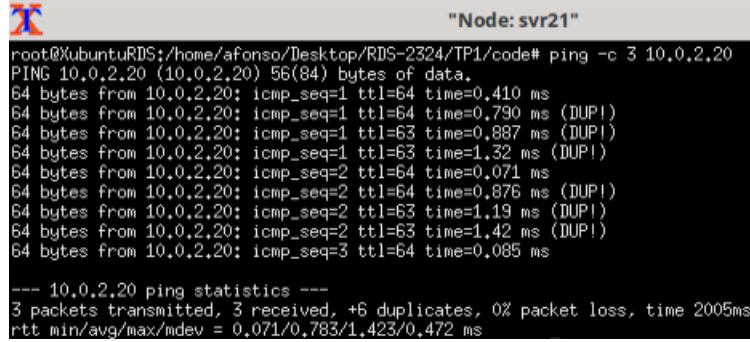
```
sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2,3,4
sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1,3,4
sudo ovs-ofctl add-flow s1 in_port=3,actions=output:1,2,4
sudo ovs-ofctl add-flow s1 in_port=4,actions=output:1,2,3
```

Os resultados foram os seguintes, onde se obteve sempre sucesso na tentativa de comunicação.

```
"Node: h21"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# ping -c 3 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=64 time=0.473 ms
64 bytes from 10.0.2.10: icmp_seq=1 ttl=64 time=1.77 ms (DUP!)
64 bytes from 10.0.2.10: icmp_seq=1 ttl=63 time=1.87 ms (DUP!)
64 bytes from 10.0.2.10: icmp_seq=1 ttl=63 time=2.60 ms (DUP!)
64 bytes from 10.0.2.10: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=64 time=1.57 ms (DUP!)
64 bytes from 10.0.2.10: icmp_seq=2 ttl=63 time=2.11 ms (DUP!)
64 bytes from 10.0.2.10: icmp_seq=2 ttl=63 time=2.88 ms (DUP!)
64 bytes from 10.0.2.10: icmp_seq=3 ttl=64 time=0.138 ms

--- 10.0.2.10 ping statistics ---
3 packets transmitted, 3 received, +6 duplicates, 0% packet loss, time 2011ms
rtt min/avg/max/mdev = 0.138/1.508/2.882/0.964 ms
```

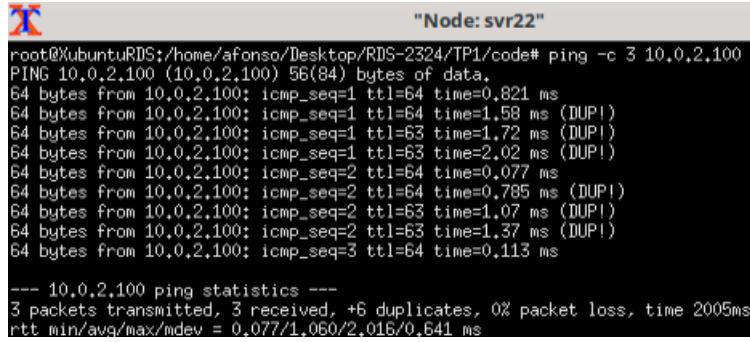
Fig. 2: h21 ping svr21



```
"Node: svr21"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# ping -c 3 10.0.2.20
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
64 bytes from 10.0.2.20: icmp_seq=1 ttl=64 time=0.410 ms
64 bytes from 10.0.2.20: icmp_seq=1 ttl=64 time=0.790 ms (DUP!)
64 bytes from 10.0.2.20: icmp_seq=1 ttl=63 time=0.887 ms (DUP!)
64 bytes from 10.0.2.20: icmp_seq=1 ttl=63 time=1.32 ms (DUP!)
64 bytes from 10.0.2.20: icmp_seq=2 ttl=64 time=0.071 ms
64 bytes from 10.0.2.20: icmp_seq=2 ttl=64 time=0.876 ms (DUP!)
64 bytes from 10.0.2.20: icmp_seq=2 ttl=63 time=1.19 ms (DUP!)
64 bytes from 10.0.2.20: icmp_seq=2 ttl=63 time=1.42 ms (DUP!)
64 bytes from 10.0.2.20: icmp_seq=3 ttl=64 time=0.085 ms

--- 10.0.2.20 ping statistics ---
3 packets transmitted, 3 received, +6 duplicates, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.071/0.783/1.423/0.472 ms
```

Fig. 3: svr21 ping svr22



```
"Node: svr22"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# ping -c 3 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
64 bytes from 10.0.2.100: icmp_seq=1 ttl=64 time=0.821 ms
64 bytes from 10.0.2.100: icmp_seq=1 ttl=64 time=1.58 ms (DUP!)
64 bytes from 10.0.2.100: icmp_seq=1 ttl=63 time=1.72 ms (DUP!)
64 bytes from 10.0.2.100: icmp_seq=1 ttl=63 time=2.02 ms (DUP!)
64 bytes from 10.0.2.100: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 10.0.2.100: icmp_seq=2 ttl=64 time=0.785 ms (DUP!)
64 bytes from 10.0.2.100: icmp_seq=2 ttl=63 time=1.07 ms (DUP!)
64 bytes from 10.0.2.100: icmp_seq=2 ttl=63 time=1.37 ms (DUP!)
64 bytes from 10.0.2.100: icmp_seq=3 ttl=64 time=0.113 ms

--- 10.0.2.100 ping statistics ---
3 packets transmitted, 3 received, +6 duplicates, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.077/1.060/2.016/0.641 ms
```

Fig. 4: svr22 ping h21

## 2.6 Teste de Comunicação Inter-LAN

Agora partimos para teste da firewall. Aqui o objetivo é garantir que as regras da firewall são devidamente cumpridas. Comecemos por mostrar que as regras na LAN 1 são asseguradas. 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 ao 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.

```
X "Node: h11"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# nping -tcp -c 3 -p 80 10.0.2.10

Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-05-17 18:46 BST
SENT (0.0263s) TCP 10.0.1.100:35581 > 10.0.2.10:80 S ttl=64 id=17603 iplen=40 seq=2742341179 win=1480
RCVD (0.0296s) TCP 10.0.2.10:80 > 10.0.1.100:35581 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (1.0267s) TCP 10.0.1.100:35581 > 10.0.2.10:80 S ttl=64 id=17603 iplen=40 seq=2742341179 win=1480
RCVD (1.0288s) TCP 10.0.2.10:80 > 10.0.1.100:35581 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (2.0299s) TCP 10.0.1.100:35581 > 10.0.2.10:80 S ttl=64 id=17603 iplen=40 seq=2742341179 win=1480
RCVD (2.0316s) TCP 10.0.2.10:80 > 10.0.1.100:35581 RA ttl=62 id=0 iplen=40 seq=0 win=0

Max rtt: 3.147ms | Min rtt: 1.643ms | Avg rtt: 2.262ms
Raw packets sent: 3 (120B) | Rcvd: 3 (120B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 2.07 seconds
```

Fig. 5: h11, TCP, porta 80, svr21

```
X "Node: h11"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# nping -tcp -c 3 -p 8080 10.0.3.10

Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-05-17 18:47 BST
SENT (0.0340s) TCP 10.0.1.100:49668 > 10.0.3.10:8080 S ttl=64 id=32131 iplen=40 seq=468798741 win=1480
RCVD (0.0387s) TCP 10.0.3.10:8080 > 10.0.1.100:49668 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (1.0356s) TCP 10.0.1.100:49668 > 10.0.3.10:8080 S ttl=64 id=32131 iplen=40 seq=468798741 win=1480
RCVD (1.0375s) TCP 10.0.3.10:8080 > 10.0.1.100:49668 RA ttl=62 id=0 iplen=40 seq=0 win=0
SENT (2.0378s) TCP 10.0.1.100:49668 > 10.0.3.10:8080 S ttl=64 id=32131 iplen=40 seq=468798741 win=1480
RCVD (2.0406s) TCP 10.0.3.10:8080 > 10.0.1.100:49668 RA ttl=62 id=0 iplen=40 seq=0 win=0

Max rtt: 4.662ms | Min rtt: 1.758ms | Avg rtt: 3.012ms
Raw packets sent: 3 (120B) | Rcvd: 3 (120B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 2.08 seconds
```

Fig. 6: h11, TCP, porta 8080, svr31

```
X "Node: h11"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# nping -tcp -c 3 -p 8080 10.0.3.20

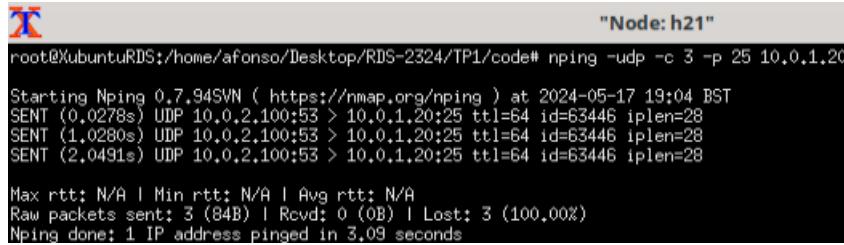
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-05-17 18:47 BST
SENT (0.0213s) TCP 10.0.1.100:24134 > 10.0.3.20:8080 S ttl=64 id=57473 iplen=40 seq=3253507614 win=1480
SENT (1.0216s) TCP 10.0.1.100:24134 > 10.0.3.20:8080 S ttl=64 id=57473 iplen=40 seq=3253507614 win=1480
SENT (2.0330s) TCP 10.0.1.100:24134 > 10.0.3.20:8080 S ttl=64 id=57473 iplen=40 seq=3253507614 win=1480

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 3 (120B) | Rcvd: 0 (0B) | Lost: 3 (100.00%)
Nping done: 1 IP address pinged in 3.08 seconds
```

Fig. 7: h11, TCP, porta 8080, svr32

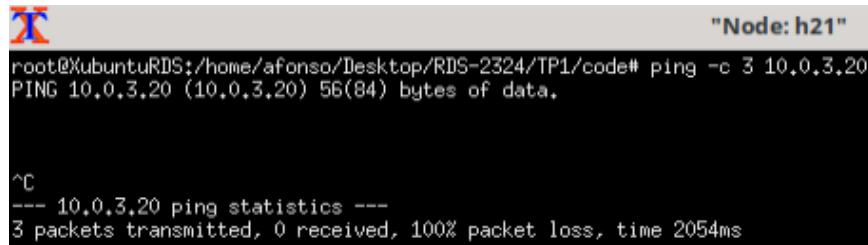
Na figura 5 e 6 comprova-se que as regras da LAN 1 onde h11 pode comunicar em TCP nas devidas portas para svr21 e svr31 estão a funcionar pois existe resposta. Na figura 7 tentamos comunicar com um IP proibido para a LAN 1 (o svr32) e, como esperado, não se recebeu resposta do outro lado.

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, pegamos nas regras da LAN 2 que ditam que é permitido comunicar com os endereços 10.0.1.20 (svr12) e 10.0.3.20 (svr32) em TCP, trocando o protocolo para UDP e ICMP.



```
"Node: h21"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# nping -udp -c 3 -p 25 10.0.1.20
Starting Nping 0.7.94SWN ( https://nmap.org/nping ) at 2024-05-17 19:04 BST
SENT (0.0278s) UDP 10.0.2.100:53 > 10.0.1.20:25 ttl=64 id=63446 iplen=28
SENT (1.0280s) UDP 10.0.2.100:53 > 10.0.1.20:25 ttl=64 id=63446 iplen=28
SENT (2.0491s) UDP 10.0.2.100:53 > 10.0.1.20:25 ttl=64 id=63446 iplen=28
Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 3 (84B) | Rcvd: 0 (0B) | Lost: 3 (100.00%)
Nping done: 1 IP address pinged in 3.09 seconds
```

Fig. 8: h21, UDP, porta 25, svr12



```
"Node: h21"
root@XubuntuRDS:/home/afonso/Desktop/RDS-2324/TP1/code# ping -c 3 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data.

^C
--- 10.0.3.20 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2054ms
```

Fig. 9: h21, ICMP, svr32

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

## 2.7 Dificuldades

Durante a resolução deste trabalho deparamos-nos com algumas dificuldades, nomeadamente no que diz respeito ao funcionamento da topologia no Mininet, proveniente da falta de conhecimento sobre Mininet, P4 e alguns conceitos fundamentais de redes.

Após conseguirmos ter a rede a funcionar, partimos para o processo de acrescentar as regras da firewall, recorrendo ao método de tentativa e erro, até que tudo se encaixasse e fizesse minimamente sentido.

Admitimos que a área das redes não é o nosso forte pelo que certamente poderá haver algumas falhas que da nossa parte poderão não ter sido tão óbvias.

### 3 Conclusão

Neste trabalho prático, implementamos uma firewall utilizando a linguagem P4 para uma rede corporativa composta por três LANs. Utilizamos o Mininet para simular a topologia de rede e testar a implementação.

Inicialmente, configuramos a topologia de rede em Python, definindo as conexões entre os hosts, switches e routers. Em seguida, implementamos as regras de encaminhamento e exclusão de tráfego na firewall através de tabelas P4, permitindo um controlo granular dos pacotes com base nos endereços IP, portas e protocolos.

Os testes realizados confirmaram que a comunicação inter-LAN e intra-LAN foi estabelecida com sucesso, respeitando as regras definidas para limitar acessos indesejados. Apesar das dificuldades encontradas, especialmente devido ao desconhecimento prévio de Mininet e P4, conseguimos superar os desafios e alcançar os objetivos propostos.

Esta experiência proporcionou um aprendizado significativo sobre redes definidas por software, a linguagem P4 e a implementação de firewalls, destacando a importância da programação e da automação na gestão eficiente de redes modernas.

### 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.paloaltonetworks.com/cyberpedia/what-is-a-stateful-firewall>
4. <https://netbeez.net/blog/how-to-use-nping/>