

Trabalho Prático - Grupo 4

João Abreu, pg55895, Miguel Jacinto, pg57590, e Ricardo Pereira, pg56001

Index Terms

SDN, P4, Data Plane, Control Plane, MSLP, Firewall, Controller, Load Balance

I. INTRODUÇÃO

Este relatório foi desenvolvido no âmbito da unidade curricular de *Redes Definidas por Software*, integrada no Mestrado em Engenharia Informática da Universidade do Minho. No presente trabalho prático, recorreu-se à linguagem de programação **P4**, uma linguagem de alto nível concebida para definir o comportamento do *data plane* de dispositivos de rede de forma flexível.

O principal objetivo consistiu em configurar a rede de modo a implementar túneis com um protocolo próprio, uma *firewall* com capacidades *stateful* e uma ligação a um *controller* dinâmico desenvolvido em *Python*.

O **P4** surgiu como resposta à crescente necessidade de maior programabilidade nas redes, especialmente no contexto das *Software-Defined Networks* (SDNs). Com o P4, é possível descrever como os pacotes são processados nos *switches* e *routers*, especificando, por exemplo, como os cabeçalhos são analisados, modificados, reenviados ou descartados. A sua principal vantagem reside na capacidade de permitir aos programadores definir novos comportamentos de encaminhamento sem depender do *firmware* dos dispositivos tradicionais.

Para compreender o funcionamento das redes programáveis com P4, é essencial distinguir entre o **data plane** e o **control plane**:

- **Data Plane:** é responsável pelo encaminhamento efetivo dos pacotes de dados. Este plano executa operações como o *header matching*, a modificação de campos, a contagem de pacotes através de counters e a aplicação de regras de encaminhamento. No contexto do P4, o *data plane* é totalmente programável, permitindo definir com precisão o comportamento dos dispositivos de rede em função dos objetivos pretendidos.
- **Control Plane:** é responsável pela tomada de decisões relativas ao encaminhamento dos pacotes, nomeadamente sobre como e para onde devem ser enviados. Este plano gere a instalação de regras no *data plane*, podendo ser configurado de forma **estática**, por exemplo, através da ferramenta *switch_cli*, onde se inserem manualmente as regras nas tabelas dos switches. No entanto, o *control plane* pode também ser configurado **dinamicamente**, sendo representado por um controller inteligente que interage diretamente com os dispositivos da rede em tempo de execução, como é o caso do controller desenvolvido neste trabalho prático.

Para cumprir eficazmente o seu papel, o control plane deve implementar diversas funcionalidades essenciais, tais como:

- Injetar *pipelines* nos dispositivos de rede;
- Adicionar regras a tabelas;
- Alterar ou remover regras previamente configuradas;
- Realizar *queries* para fins de monitorização do tráfego e estado da rede;
- Registar e manter a informação atualizada sobre os dispositivos;
- Receber comunicações dos dispositivos.

Label Switching Protocols (LSP) são protocolos em que o envio de pacotes é baseado em labels, no lugar dos endereços tradicionais como MACs e IPs. A utilização de *labels* é fundamental neste tipo de redes, uma vez que reduz a complexidade dos pacotes e do encaminhamento dos mesmos, permitindo identificar os pacotes mais rapidamente do que através do encaminhamento tradicional por IP, que exige a análise de todo o cabeçalho do pacote.

Nos pontos de entrada e saída da rede, os *edge routers* são responsáveis por aplicar e remover as *labels*, respetivamente. Já os *routers* intermédios, situados no interior da rede, limitam-se a encaminhar os pacotes com base nas *labels*, sem necessidade de inspecionar o conteúdo completo do cabeçalho, o que resulta num encaminhamento mais eficiente e com menor latência. Este protocolo é especialmente vantajoso em redes de elevado desempenho e redes core, onde a escalabilidade e rapidez no processamento de pacotes são críticas. Além disso, o MSLP pode ser integrado com um *controller* dinâmico, que monitoriza o estado da rede e ajusta a colocação das etiquetas em tempo real, otimizando assim o uso dos recursos da rede e garantindo qualidade de serviço (*QoS*).

II. IMPLEMENTAÇÃO

A. Topologia

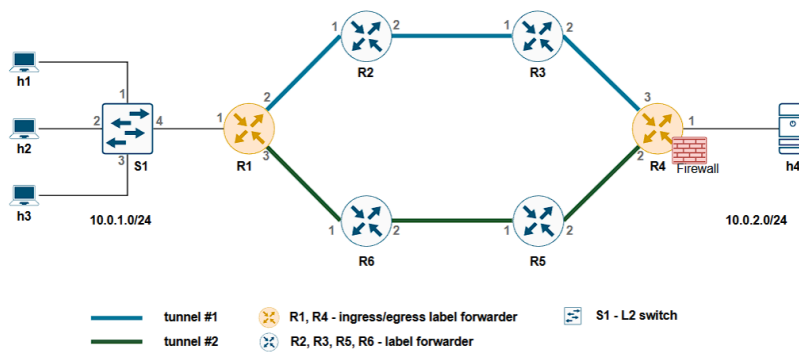


Fig. 1. Topologia da implementação: composta por um switch, dois routers MPLS (sendo o R4 também uma firewall) e quatro routers internos à rede.

A topologia utilizada na nossa implementação baseia-se na apresentada no enunciado, conforme ilustrado na Figura 1. A topologia inclui 2 LAN's distintas, uma delas com um switch de camada 2 a fazer a ligação à rede de 3 hosts, simulando uma rede interna de uma casa comum, e a outra apenas com um host ligado diretamente à rede, simulando um servidor.

A topologia conta ainda com vários switches, de diferentes tipos e que operam em diferentes camadas da rede. Para cada tipo de dispositivo foi implementado um programa P4 específico, posteriormente injetado pelo controller:

- **S1 – l2switch.p4:** Switch de camada 2 que aprende endereços MAC dinamicamente e encaminha pacotes analisando os cabeçalhos *Ethernet*. Para endereços que ainda não conheça adota o comportamento de um *Hub*, reencaminhando os pacotes para todas as portas definidas num grupo de *Multicast*. Uma vez que conheça o endereço passa a reencaminhar os pacotes apenas para a porta específica.
- **R1 – l3switch_mslp.p4:** Router de camada 3 com suporte a encaminhamento IPv4 e MSLP, este por meio de túneis. Processa cabeçalhos desde a camada de ligação até à camada de transporte, suportando os seguintes cabeçalhos: Ethernet; IPv4 e MSLP; TCP, UDP e ICMP.
- **R4 – l3switch_mslp_firewall.p4:** Inclui todas as funcionalidades do programa `l3switch_mslp.p4`, com a adição de funcionalidades de *stateful firewall*. Utiliza *Bloom Filters* para

controlar o tráfego TCP, UDP e ICMP, podendo impedir a passagem de fluxos desconhecidos com base na direção do mesmo. Permite ainda abrir portas para tráfego TCP e/ou UDP, permitindo a passagem de qualquer fluxo do protocolo pretendido que seja direcionado a essas portas.

- **R2, R3, R5, R6 – 13switch_tunnel.p4:** Routers especializados no encaminhamento de pacotes dentro de túneis MSLP. Não suportam encaminhamento por outros protocolos que não o MSLP. Processam a primeira *label* do protocolo para realizar o encaminhamento do pacote, removendo-a à saída.

Nota: Todos os dispositivos utilizam a arquitetura `V1Model`, com parser, controlo de entrada/saída, checksum e deparser.

TABLE I: Tabela dispositivos, portas, endereços IP, endereços MAC e labels da nossa Topologia

Dispositivo A	Porta A	IP A	MAC A	Label A	Dispositivo B	Porta B
h1	1	10.0.1.1/32	aa:00:00:00:00:01	NA	s1	1
h2	1	10.0.1.2/32	aa:00:00:00:00:02	NA	s1	2
h3	1	10.0.1.3/32	aa:00:00:00:00:03	NA	s1	3
s1	1	NA	cc:00:00:00:01:01	NA	h1	1
s1	2	NA	cc:00:00:00:01:02	NA	h2	1
s1	3	NA	cc:00:00:00:01:03	NA	h3	1
s1	4	NA	cc:00:00:00:01:04	NA	r1	1
r1	1	10.0.1.254/24	aa:00:00:00:01:01	0x1010	s1	4
r1	2	10.0.12.1/24	aa:00:00:00:01:02	0x1020	r2	1
r1	3	10.0.16.1/24	aa:00:00:00:01:03	0x1030	r6	1
r2	1	10.0.12.2/24	aa:00:00:00:02:01	0x2010	r1	2
r2	2	10.0.23.2/24	aa:00:00:00:02:02	0x2020	r3	1
r3	1	10.0.23.3/24	aa:00:00:00:03:01	0x3010	r2	2
r3	2	10.0.34.3/24	aa:00:00:00:03:02	0x3020	r4	3
r4	1	10.0.2.254/24	aa:00:00:00:04:01	0x4010	h4	1
r4	2	10.0.45.4/24	aa:00:00:00:04:02	0x4020	r5	2
r4	3	10.0.34.4/24	aa:00:00:00:04:03	0x4030	r3	2
r5	1	10.0.56.5/24	aa:00:00:00:05:01	0x5010	r6	2
r5	2	10.0.45.5/24	aa:00:00:00:05:02	0x5020	r4	2
r6	1	10.0.16.6/24	aa:00:00:00:06:01	0x6010	r1	3
r6	2	10.0.56.6/24	aa:00:00:00:06:02	0x6020	r5	1
h4	1	10.0.2.1/32	aa:00:00:00:00:04	NA	r4	1

A Tabela I apresenta as configurações utilizadas na implementação da topologia representada na Figura 1. Cada linha descreve uma ligação entre dois dispositivos, indicando o dispositivo e a porta de origem, os respetivos endereços IP, MAC e Label MSLP associados à porta, e o dispositivo e porta de destino.

B. My Sequence Label Protocol (MSLP)

```

1  const bit<16> TYPE_MSLP = 0x88B5;
2
3  header mslp_t {
4      bit<16> etherType; // L3 protocol
5  }
6  header label_t {
7      bit<16> label;
8      bit<8>  bos;
9  }
10
11 struct headers {
12     mslp_t    mslp;
13     label_t[4] labels;
14     ...
15 }

```

1) *Cabeçalhos*: Para a implementação de um protocolo único, criado por nós para introduzir o encaminhamento através de labels, optamos por introduzir 2 novos cabeçalhos:

- **mslp_t**: cabeçalho único em cada pacote, contém um só campo `etherType` de 16 bits, usado para guardar o identificador do protocolo de rede encapsulado pelo MSLP.
- **label_t**: definido como uma *header stack*, cada elemento contém um campo `label` de 16 bits, com a label em concreto (ver Tabela I) e um campo `bos` de 8 bits, que indica se é a última label da pilha ou não, necessário para o processo de *parsing* do cabeçalho.

Tivemos ainda de definir um novo `etherType` para identificar o nosso protocolo no cabeçalho Ethernet. Escolhemos o valor `0x88B5` pois é um valor reservado para protocolos personalizados, não havendo assim riscos de ofuscar algum protocolo existente, ainda que na nossa simulação apenas se use o protocolo IPv4 (`etherType`: `0x800`).

Analisando com mais detalhe o cabeçalho `label_t`, o campo `bos` ocupa 8 bits para que o tamanho total de cada cabeçalho seja múltiplo de 8, porque na prática apenas é necessário 1 bit para representar os 2 possíveis estados do campo (0 e 1). Já o campo `label` é constituído por 2 bytes, totalizando os 16 bits, em que o primeiro byte representa o dispositivo a que pertence a label e o segundo a porta a que corresponde. Deste modo, cada switch é capaz de atribuir uma porta de saída e reescrever os endereços MAC necessários, de acordo com a label recebida.

O tamanho total do cabeçalho poderia ser reduzido, para 8 bits, sendo 4 para o dispositivo, 3 para a porta e 1 para o `bos`, no entanto o ganho de performance seria mínimo, senão mesmo insignificante, e implicaria uma perda de representatividade enorme. Com labels de 16 bits podemos "atribuir" 12 bits (3 algarismos hexadecimais) para representar 4096 dispositivos diferentes, cada um com 16 portas (os 4 bits restantes). Com labels de 16 bits e um `bos` separado da label, temos ainda a vantagem de tornar os valores de cada campo mais legíveis numa captura de pacotes, permitindo identificar mais facilmente as labels num pacote e o percurso efetuado pelo mesmo.

2) *Parsing*: O processo de parsing difere um pouco do habitual devido à stack de labels.

```

1  state parse_ethernet {
2      packet.extract(hdr.ethernet);
3      transition select (hdr.ethernet.etherType) {
4          TYPE_MSLP: parse_mslp;
5          TYPE_IPV4: parse_ipv4;
6          default: accept;
7      }
8  }
9
10 state parse_mslp {
11     packet.extract(hdr.mslp);
12     transition parse_labels;
13 }
14
15 state parse_labels {
16     packet.extract(hdr.labels.next);
17     transition select (hdr.labels.last.bos) {
18         0x00: parse_labels; // Create a loop
19         0x01: guess_labels_payload;
20     }
21 }
22
23 state guess_labels_payload {
24     transition select (hdr.mslp.etherType) {
25         TYPE_IPV4: parse_ipv4;
26         default: accept;
27     }
28 }

```

O processo de parsing difere um pouco do habitual devido à stack de labels. O primeiro cabeçalho a

ser extraído é o `mslp_t`, seguido do primeiro `label_t`. Aqui, caso o campo `bos` da label extraída seja 0, volta a executar o mesmo estado, extraindo a label seguinte e criando assim um ciclo para extrair todas as labels, independentemente do tamanho da stack. Caso o `bos` seja 1, termina o loop e avança para um estado que verifica qual o cabeçalho (de nível 3) seguinte consoante o `mslp_t` extraído antes.

3) *Ingress e Egress*: O fluxograma da Figura 2 resume o processo lógico de um pacote num Router MSLP.

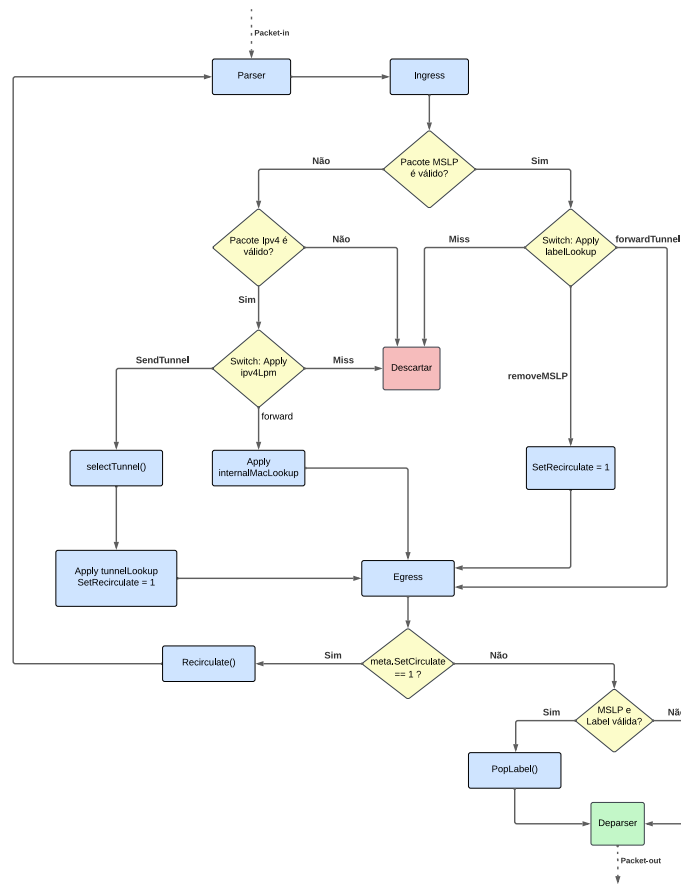


Fig. 2. Fluxograma representativo da lógica do MSLP. O diagrama evidencia o recirculamento de pacotes que ocorre nos routers R1 e R4 durante o processo de encapsulamento e desencapsulamento entre pacotes IPv4 e MSLP.

Cada pacote pode ser processado 1 ou 2 vezes num router MSLP. Na primeira iteração o pacote é preparado para entrar ou sair do túnel e na segunda é preparado para ser encaminhado para o destino.

No caso de um pacote MSLP (como se viesse de um túnel), verifica-se se o pacote deve continuar no túnel ou se deve sair, consoante a label que recebe. Isto permite que o router funcione tanto como entrada e saída de um túnel como ponto intermédio de outro túnel. Para que o pacote saia do túnel, os cabeçalhos do protocolo MSLP são marcados como inválidos para que não sejam emitidos, e o pacote é recirculado no Egress, voltando a ser processado como um novo pacote (voltando ao Parser). Na segunda iteração, o pacote já não é "visto" como um pacote MSLP, mas sim um pacote IPv4 normal, sendo processado como tal e encaminhado para o destino de acordo com a tabela `ipv4Lpm`.

No caso de um pacote IPv4 (como se viesse da LAN), verifica-se se o destino pertence à mesma LAN, a outra LAN ou se é desconhecido. No último caso, o pacote é apenas descartado. No caso de pertencer à mesma LAN, o pacote é reenviado por IPv4 para a porta correspondente. No caso de ser para outra LAN, é selecionado um túnel consoante uma hash gerada pelas informações de origem, destino e protocolos do pacote, são adicionados os cabeçalhos MSLP com as labels correspondentes ao túnel selecionado, e o pacote é recirculado. Na segunda iteração, o pacote é tratado como outro pacote qualquer que venha de um túnel.

Assumindo que as regras de fluxo estão todas bem escritas para a topologia utilizada, esta implementação permite a passagem de IPv4 para MSLP e vice-versa, sem correr riscos de gerar ciclos infinitos dentro de um router ou entre os routers de cada extremidade de um túnel. No entanto, a implementação é limitada ao uso de um túnel entre 2 LANs, sendo muito provavelmente necessárias adaptações à mesma para escalar para mais do que 2 LANs.

C. Stateful Firewall

A firewall implementada opera de forma *stateful*, com a capacidade de controlar e manter registos de fluxos de pacotes ativos, recorrendo a *Bloom filters* e a tabelas para indicar portas permitidas. O funcionamento completo da firewall está ilustrado na Figura 3, onde se pode acompanhar detalhadamente a lógica de operação da firewall.

```

1 struct metadata {
2     @field_list(0) // preserved on recirculate_preserving_field_list
3     bit<9> ingress_port;
4 }
5 #define BLOOM_FILTER_ENTRIES 4096
6
7 register<bit<1>>(4096) bloom_filter_1;
8 register<bit<1>>(4096) bloom_filter_2;
9 bit<32> reg_pos_1; bit<32> reg_pos_2;
10 bit<1> reg_val_1; bit<1> reg_val_2;
11 bit<1> direction; bit<1> activateFirewall;
12
13 table checkDirection {
14     key = { meta.ingress_port: exact;
15             standard_metadata.egress_spec: exact; }
16     actions = { setDirection;
17                 drop; }
18     size = 16;
19     default_action = drop;
20 }
21
22 // semelhante para UDP tambem
23 table allowedPortsTCP {
24     key = { hdr.tcp.dstPort: exact; }
25     actions = { NoAction; }
26     size = 16;
27     default_action = NoAction;
28 }

```

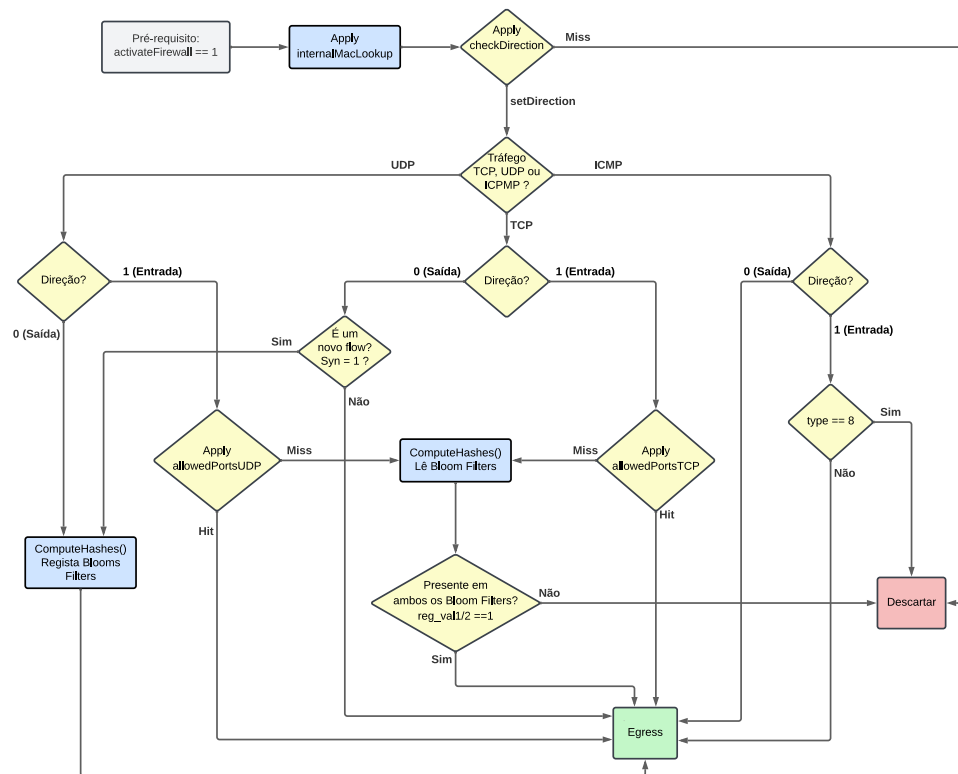


Fig. 3. Fluxograma representativo dos caminhos que os pacotes podem seguir na Firewall.

Para tráfego TCP e UDP, as tabelas `allowedPortsTCP` e `allowedPortsUDP` definem portas de destino autorizadas para o tráfego respetivo, permitindo definir dinamicamente exceções à *firewall*. Caso o tráfego não seja destinado a uma dessas portas, a *firewall* acompanha os fluxos utilizando dois *Bloom filters*, representados pelas duas estruturas `bloom_filter_1` e `bloom_filter_2` com 4096 entradas cada. Cada fluxo é mapeado para dois índices, um em cada filtro, calculados através de funções de hash com algoritmos diferentes (`crc16` e `crc32`).

A *firewall* é ativada quando a variável `activateFirewall` é definida como 1, algo que ocorre no máximo uma vez durante o processamento de cada pacote num router MSLP-Firewall. O processamento da *firewall* ocorre sempre em último lugar, imediatamente antes de reencaminhar o pacote, ou seja, caso o processamento MSLP necessite de recircular o pacote, a *firewall* é apenas executada no final da segunda iteração.

A tabela `checkDirection` fornece flexibilidade na configuração da *firewall*, permitindo que o controlador defina dinamicamente quando pretende que um pacote seja filtrado pela mesma (`direction=1`) e quando não pretende (`direction=0`), através das portas de entrada e de saída do pacote. Um detalhe importante é que um pacote recirculado deixa de ter uma porta de entrada associada, daí ter sido necessário, na primeira iteração, guardar o valor nos metadados na variável `meta.ingress_port` com a *tag* `@field_list(0)`, a indicar que o valor desse campo deve ser preservado durante a recirculação do pacote.

A *firewall* processa pacotes UDP, TCP e ICMP, conforme a direção definida e as suas características. Se não for selecionada uma direção, por defeito os pacotes são processados como se de entrada se

tratassem.

- **Pacotes de saída (`direction == 0`):**

- **UDP:** Regista o fluxo nos *Bloom filters* utilizando o 5-tuple (IP origem/destino, portas origem/destino, protocolo), marcando as posições calculadas pelos hashes para permitir o tráfego de resposta futuro.
- **TCP:** Apenas quando a flag SYN está ativa (início da conexão TCP) é que o fluxo é registado nos filtros. Pacotes sem SYN são permitidos sem registo, assumindo que o fluxo já está identificado.
- **ICMP:** Permite todos os pacotes, não sendo necessário qualquer registo.

- **Pacotes de entrada (`direction == 1`):**

- **UDP e TCP:** Primeiro são verificadas as portas de destino nas tabelas `allowedPortsUDP` e `allowedPortsTCP`, respetivamente. Se a porta estiver na lista, o pacote é permitido. Caso contrário, verifica-se nos *Bloom filters* se o fluxo reverso está registado — só permite o pacote se ambos os filtros indicarem que o fluxo existe.
- **ICMP:** São descartados pacotes do tipo Echo Request (tipo 8), protegendo a rede contra varreduras externas. Outros tipos são permitidos.

D. Controller

1) *Configuração Inicial dos Switches:* O *controller* começa por ler os ficheiros de configuração JSON com todas as regras para conexão e configuração de cada switch, incluindo a localização dos ficheiros que descrevem os programas P4 e os logs.

A função `create_connections_to_switches` estabelece conexões via gRPC com os switches, definindo o *controller* como mestre através de `MasterArbitrationUpdate`. Cada switch é configurado pela função `setup_switches`, que executa as seguintes tarefas:

- **Instalação do Programa P4:** Verifica se existe algum programa P4 instalado e, caso exista, se o programa instalado é o mesmo que está indicado no JSON, comparando os ficheiros `p4info`. Se necessário, instala o programa P4 correspondente usando `SetForwardingPipelineConfig`. Em reinicializações, força a reinstalação mesmo que o programa correto já esteja instalado.
- **Clone Engines e Grupos Multicast:** Define grupos de multicast para difusão de pacotes nos switches L2 e *clone engines* para envio de pacotes para o *controller*.
- **Default actions:** Define as ações padrão das várias tabelas (ex.: `ipv4Lpm`, `labelLookup`, `tunnelLookup`, ...).
- **Leitura de Regras:** Lê as regras atuais das tabelas, armazenando-as no dicionário `state` para manter uma visão informada do estado dos switches.
- **Regras Estáticas:** Instala regras pré-definidas, como encaminhamento IPv4 ou MSLP, re-escrita de endereços MAC, entre outras, caso ainda não estejam escritas nas tabelas ou estejam escritas com parâmetros diferentes.

Estas operações garantem que os switches estejam prontos para processar pacotes IPv4 e MSLP, com tabelas configuradas para permitir o correto funcionamento das funcionalidades descritas nos pontos anteriores.

2) *Túneis MSLP e Load Balancing:* O *controller* verifica o estado atual dos túneis e, se necessário, instala o estado inicial na tabela `tunnelLookup`. Aplica as regras em ambos os switches do túnel (ex.: R1 e R4), garantindo consistência nos rótulos de cada extremidade.

Para implementar o load balance, foi utilizada uma solução baseada em contadores de pacotes. Cada router MSLP possui um contador para monitorizar os pacotes recebidos em cada túnel. A verificação `instance_type==0` garante que o pacote só é contabilizado uma vez, não contabilizando pacotes

recirculados. O contador está definido com 4 entradas (índices de 0 a 3) para facilitar a leitura e contagem dos pacotes de acordo com a porta de entrada, uma vez que na nossa topologia as portas associadas aos túneis são sempre as 2 e 3. Os índices 0 e 1 ficam assim inutilizados, embora este último ainda receba atualizações do valor que nunca são lidas. Dado o propósito da simulação e a escala da topologia não consideramos o impacto relevante.

```
1 // Contador dos pacotes de cada tunel
2 counter(4, CounterType.packets) tunnel_counter;
3
4 // No apply do Ingress de R1 e R4
5 if (standard_metadata.instance_type == 0) {
6     tunnel_counter.count((bit<32>) standard_metadata.ingress_port);
7 }
```

Para cada túnel é criada no *controller* uma *thread* dedicada que lê os contadores de cada router em intervalos de tempo pré-definidos, calculando o tráfego recebido em ambos os switches do túnel para cada caminho (*up* e *down*).

Se a diferença entre o tráfego dos dois caminhos exceder um limiar, o *controller* alterna o estado do túnel, atualizando a tabela `tunnelLookup` com as labels MSLP invertidas face ao estado anterior. O estado atual é guardado na variável `tunnels`, garantindo que a troca é realmente efetuada e que não reescreve o mesmo estado em que o túnel já se encontrava. Quando é feita uma troca de estado, o intervalo de tempo até à próxima verificação é triplicado para dar uma margem maior para as mudanças fazerem efeito numa tentativa de voltar a equilibrar o tráfego entre os 2 caminhos do túnel.

3) *Interação e Gestão em Tempo Real*: O *controller* disponibiliza comandos interativos que permitem a gestão em tempo real da rede, sem precisar de o reiniciar:

- `show`: Consulta o estado atual do controlador e das tabelas de switches específicos.
- `reset`: Reinicializa túneis, contadores, switches específicos ou a topologia inteira;

Comandos como `exit`, `quit` ou `q` terminam a execução do *controller*. Antes de terminar, este espera que todas as *threads* que estejam adormecidas acordem e terminem, antes de terminar a *main thread*.

Os comandos `show` permitem inspecionar com detalhe algumas informações relevantes sobre o estado da simulação e aferir resultados de testes. Já os comandos `reset` são os responsáveis por permitir a alteração de regras de fluxo e outras configurações dos switches dinamicamente sem ter de reinicializar o controlador. Para alterar uma configuração basta alterar o ficheiro JSON e fazer o *reset* da estrutura adequada para que as mudanças se reflitam na simulação.

Durante o *reset* de um switch em específico são também reinicializados todos os túneis, voltando às regras definidas no JSON, ou a umas mais recentes caso o ficheiro `tunnels_config` tenha sido atualizado também. Isto é feito para evitar inconsistências entre as tabelas do switch, os túneis MSLP já existentes e o estado atual dos túneis, uma vez que este estado é gerido por *threads* separadas e cada túnel está associado a 2 switches diferentes. Como as regras de criação e manutenção de estado dos túneis estão centralizadas num único ficheiro e não espalhadas por cada ficheiro de configuração de um switch, e ao reiniciar um switch todas as suas regras são apagadas, incluindo as dos túneis, somos obrigados a reinicializar todos os túneis.

III. TESTES & RESULTADOS

A. Teste à Natureza Stateful do Controlador

Este teste teve como objetivo demonstrar que o controlador mantém estado interno, ou seja, não reinstala regras desnecessariamente se estas já estiverem configuradas nos switches.

Procedimento a ser cumprido:

- 1) Executar o controlador pela primeira vez. Deverão surgir várias mensagens como a da Figura 4, onde é indicado no terminal a instalação de regras nas tabelas dos switches.
- 2) Interromper a execução do controlador manualmente:

```
1 exit
```

- 3) Voltar a iniciar o controlador. Desta vez, não deverão aparecer mensagens relativas à instalação de novas regras, indicando que o controlador reconhece que as tabelas já se encontram corretamente configuradas.

- 4) Para inspecionar as tabelas, digite:

```
1 show r1
```

- 5) Como exibido na Figura 5, confirmar que todas as regras previamente instaladas continuam ativas nas tabelas dos switches, comprovando que o controlador preserva o estado da rede entre execuções.

```
netmin@minie-ws:~/RDS/rds$ python3 controller/tp-controller.py --config co
one_config.json"
----- Connecting to the devices... -----
----- Connection successful! -----

----- Installing P4 Programs... -----
s1: No P4 program found, installing...
s1: P4 program installed.
r1: No P4 program found, installing...
r1: P4 program installed.
r2: No P4 program found, installing...
r2: P4 program installed.
r3: No P4 program found, installing...
r3: P4 program installed.
r4: No P4 program found, installing...
r4: P4 program installed.
r5: No P4 program found, installing...
r5: P4 program installed.
r6: No P4 program found, installing...
r6: P4 program installed.
----- P4 Programs Installation done! -----

----- Installing MC Groups and Clone Sessions... -----
Installed Multicast Group 1 on s1
Installed clone session 300 on s1
Q Listening for packet-ins on s1
----- MC Groups and Clone Sessions done! -----

----- Writing Default Actions... -----
----- Write Default Actions done! -----

----- Reading Tables Rules... -----
----- Read Tables Rules done! -----

----- Writing Static Rules... -----
Adding rule to r1 for table MyIngress.ipv4lpm
Adding rule to r1 for table MyIngress.ipv4lpm
Adding rule to r1 for table MyIngress.ipv4lpm
Adding rule to r1 for table MyIngress.ipv4lpm
Adding rule to r1 for table MyIngress.labelLookup
Adding rule to r1 for table MyIngress.labelLookup
Adding rule to r1 for table MyIngress.labelLookup
Adding rule to r1 for table MyIngress.internalMacLookup
Adding rule to r1 for table MyIngress.internalMacLookup
Adding rule to r1 for table MyIngress.internalMacLookup
Adding rule to r2 for table MyIngress.labelLookup
Adding rule to r2 for table MyIngress.labelLookup
```

Fig. 4. Primeira Execução do Controller. As regras foram corretamente adicionadas.

```
netmin@minie-ws:~/RDS/rds$ python3 controller/tp-controller.py --config configs/switch
one_config.json"
----- Connecting to the devices... -----
----- Connection successful! -----

----- Installing P4 Programs... -----
s1: Correct P4 program already installed, skipping.
r1: Correct P4 program already installed, skipping.
r2: Correct P4 program already installed, skipping.
r3: Correct P4 program already installed, skipping.
r4: Correct P4 program already installed, skipping.
r5: Correct P4 program already installed, skipping.
r6: Correct P4 program already installed, skipping.
----- P4 Programs Installation done! -----

----- Installing MC Groups and Clone Sessions... -----
Q Listening for packet-ins on s1
----- MC Groups and Clone Sessions done! -----

----- Writing Default Actions... -----
----- Write Default Actions done! -----

----- Reading Tables Rules... -----
----- Read Tables Rules done! -----

----- Writing Static Rules... -----
----- Write Static Rules done! -----

[r1-to-r4] detected existing tunnel state 0
$ Starting tunnel monitor 'r1-to-r4' between r1 & r4

>>>

[r1-to-r4] up=0, down=0
[r1] up=0, down=0
[r4] up=0, down=0
[r1-to-r4] no switch needed
```

Fig. 5. Segunda Execução do Controller. Nenhuma nova regra foi adicionada, uma vez que estavam já guardadas.

B. Teste à Capacidade de Aprendizagem do Switch

Este teste tem como objetivo verificar se o switch s1 é capaz de aprender dinamicamente os endereços MAC dos hosts envolvidos nas comunicações, funcionando como um *learning switch*.

- 1) Confirmar que a tabela de encaminhamento do switch s1 está inicialmente vazia como a da Figura 6.
- 2) Para inspecionar a tabela do switch s1, digite:

```
1 show s1
```

- 3) Efetuar uma comunicação entre dois hosts. Por exemplo, enviar um ping do host h1 para o host h2:

```
1 h1 ping h2
```

- 4) Observar os prints no terminal do controlador (Figura 7), que deverão indicar a aprendizagem de novos endereços MAC e a instalação das respectivas regras de encaminhamento.
- 5) Reinspecionar a tabela do switch s1 para verificar se foram adicionadas entradas correspondentes aos dois hosts:

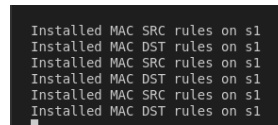
```
1 show s1
```

- 6) Confirmar que a tabela contém agora entradas para os fluxos de tráfego entre h1 e h2 como a da Figura 8. O número de entradas pode variar:
 - Se for a primeira comunicação entre os hosts h1 e h2 após o arranque da topologia (sem execuções anteriores no Mininet), serão adicionadas 4 regras à tabela do switch: duas para o fluxo de ida e duas para o fluxo de volta.
 - Caso já tenha existido anteriormente uma execução do Mininet onde houve comunicação entre os hosts, poderão ser instaladas até 6 regras, devido à existência de regras residuais associadas a múltiplos fluxos aprendidos.



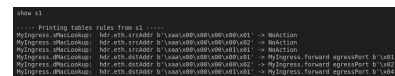
```
show s1
----- Printing tables rules from s1 -----
>>>
```

Fig. 6. Estado inicial do switch s1. Sem regras instaladas.



```
Installed MAC SRC rules on s1
Installed MAC DST rules on s1
Installed MAC SRC rules on s1
Installed MAC DST rules on s1
```

Fig. 7. Prints no terminal do controlador após o ping ser executado no Mininet.



```
show s1
----- Printing tables rules from s1 -----
h1egress: eth1:macaddr: h1:aa:aa:aa:aa:aa:aa:1 -> h1:aa:aa:aa:aa:aa:aa:1
h1egress: eth1:macaddr: h1:aa:aa:aa:aa:aa:aa:2 -> h1:aa:aa:aa:aa:aa:aa:2
h1egress: eth1:macaddr: h1:aa:aa:aa:aa:aa:aa:1 -> h1:aa:aa:aa:aa:aa:aa:1
h1egress: eth1:macaddr: h1:aa:aa:aa:aa:aa:aa:2 -> h1:aa:aa:aa:aa:aa:aa:2
h1egress: eth1:macaddr: h1:aa:aa:aa:aa:aa:aa:1 -> h1egress: forward: egress:port: h1:aa:aa:aa:aa:aa:aa:1
h1egress: eth1:macaddr: h1:aa:aa:aa:aa:aa:aa:2 -> h1egress: forward: egress:port: h1:aa:aa:aa:aa:aa:aa:2
```

Fig. 8. Estado final da tabela do switch s1. As regras foram corretamente instaladas.

C. Teste para fazer o reset do Switch

Este teste tem como objetivo verificar o comportamento do switch após a reinstalação das regras de clone, e como o reset permite aplicar alterações no ficheiro Json dinamicamente.

- 1) Alterar o ficheiro JSON de configuração do clone, removendo a entrada correspondente à porta 2 (exemplo: "egress_port": 2, "instance": 1). Esta remoção simula a ausência de uma réplica necessária para o correto funcionamento da comunicação. Essa alteração está representada na Figura 9.
- 2) Efetuar um teste de conectividade entre dois hosts. Como exibido na Figura 10, comunicação não funcionará:

```
1 h1 ping h2
```

- 3) Corrigir novamente o ficheiro JSON de configuração do clone, adicionando de volta a entrada com a porta 2.
- 4) Como exibido na Figura 11, enviar o comando de reset ao switch s1 através do controller, para que este volte a instalar as regras, agora com a configuração atualizada:

```
1 reset s1
```

- 5) Repetir o teste de conectividade entre os hosts. Desta vez, como demonstrado na Figura 12 a comunicação deverá funcionar corretamente:

```
1 h1 ping h2
```

```

1 {
2   "s1": {
3     "mcSessionId": 1,
4     "broadcastReplicas": [
5       { "egress_port": 1, "instance": 1 },
6       { "egress_port": 3, "instance": 1 },
7       { "egress_port": 4, "instance": 1 }
8     ],
9     "cpuSessionId": 100,
10    "cpuReplicas": [
11      { "egress_port": 510, "instance": 1 }
12    ]
13  }
14 }

```

Fig. 9. Remoção da entrada correspondente ao à porta 2.

```

reset s1
✚ Resetting switch s1...
❏ Cleaning ALL tunnel rules from all switches...
❏ Stopped tunnel monitor for r1-to-r4
❏ Removed rule with match {'meta.tunnel': 0} from r1
❏ Removed rule with match {'meta.tunnel': 1} from r1
❏ Removed rule with match {'meta.tunnel': 0} from r4
❏ Removed rule with match {'meta.tunnel': 1} from r4
[s1] packet-in stream cancelled, exiting listener.
[s1] listener thread terminating.
----- Connecting to the device... -----
----- Connection successful! -----

```

Fig. 11. Limpar as regras associadas ao Switch s1.

```

Ready !
*** Starting CLI:
mininet> h1 ping h2 -c 10
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data:
From 10.0.1.1 icmp_seq=1 Destination Host Unreachable
From 10.0.1.1 icmp_seq=2 Destination Host Unreachable
From 10.0.1.1 icmp_seq=3 Destination Host Unreachable
From 10.0.1.1 icmp_seq=4 Destination Host Unreachable
From 10.0.1.1 icmp_seq=5 Destination Host Unreachable
From 10.0.1.1 icmp_seq=6 Destination Host Unreachable
From 10.0.1.1 icmp_seq=7 Destination Host Unreachable
From 10.0.1.1 icmp_seq=8 Destination Host Unreachable
From 10.0.1.1 icmp_seq=9 Destination Host Unreachable
From 10.0.1.1 icmp_seq=10 Destination Host Unreachable

--- 10.0.1.2 ping statistics ---
10 packets transmitted, 0 received, +10 errors, 100% packet loss, time 9201ms
pipe 4
mininet>

```

Fig. 10. Ping de h1 para h2. Não funciona pela ausência da regra.

```

--- 10.0.1.2 ping statistics ---
10 packets transmitted, 0 received, +10 errors, 100% packet loss, time 9201ms
pipe 4
mininet> h1 ping h2 -c 10
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data:
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=4.39 ms
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=8.91 ms (DUP!)
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.694 ms (DUP!)
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.631 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=2.09 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=64 time=0.566 ms
64 bytes from 10.0.1.2: icmp_seq=6 ttl=64 time=1.74 ms
64 bytes from 10.0.1.2: icmp_seq=7 ttl=64 time=0.922 ms
64 bytes from 10.0.1.2: icmp_seq=8 ttl=64 time=0.949 ms
64 bytes from 10.0.1.2: icmp_seq=9 ttl=64 time=0.429 ms
64 bytes from 10.0.1.2: icmp_seq=10 ttl=64 time=0.861 ms

--- 10.0.1.2 ping statistics ---
10 packets transmitted, 10 received, +2 duplicates, 0% packet loss, time 9142ms
rtt min/avg/max/mdev = 0.429/2.596/8.987/3.024 ms

```

Fig. 12. Ping de h1 para h2. Funciona pelo acrescento da nova regra.

D. Teste de Firewall para Proteção de Porta Específica

Este teste tem como objetivo verificar a funcionalidade da firewall ao proteger uma porta específica, permitindo comunicação apenas em uma direção.

- 1) Confirmar que o host h4 consegue enviar um ping para o host h1:

```
h4 ping h1
```

A comunicação deverá funcionar corretamente como a demonstrada na Figura 13.

- 2) Testar a comunicação inversa, tentando enviar um ping de h1 para h4:

```
h1 ping h4
```

A comunicação deverá falhar como exibido na Figura 14, porque a Firewall protege a porta e impede o fluxo nesta direção.

```

mininet> h4 ping h1 -c 10
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data:
64 bytes from 10.0.1.1: icmp_seq=1 ttl=63 time=5.49 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=63 time=5.81 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=63 time=4.45 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=63 time=5.43 ms
64 bytes from 10.0.1.1: icmp_seq=5 ttl=63 time=6.40 ms
64 bytes from 10.0.1.1: icmp_seq=6 ttl=63 time=3.95 ms
64 bytes from 10.0.1.1: icmp_seq=7 ttl=63 time=4.81 ms
64 bytes from 10.0.1.1: icmp_seq=8 ttl=63 time=4.41 ms
64 bytes from 10.0.1.1: icmp_seq=9 ttl=63 time=4.20 ms
64 bytes from 10.0.1.1: icmp_seq=10 ttl=63 time=4.64 ms

--- 10.0.1.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9015ms
rtt min/avg/max/mdev = 3.948/4.959/6.401/0.748 ms

```

Fig. 13. Ping entre o H4 para o H1. Como é um ping interno à rede da Firewall, é considerado uma saída (0) e por isso o fluxo é permitido.

```

mininet> h1 ping h4 -c 10
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.

--- 10.0.2.1 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9238ms

```

Fig. 14. Ping entre o H1 para o H4. Como é um ping externo da Rede Firewall, é bloqueado.

E. Teste para Desativar a Firewall Dinamicamente

Este teste tem como objetivo verificar a capacidade de desativar a firewall dinamicamente através da alteração das regras no ficheiro JSON, seguido de um reset do switch.

- 1) Como exibido na Figura 14, o ping de h1 para h4 não funciona.
- 2) Na Figura 15 foi alterado o ficheiro JSON das regras do switch r4 para definir todos os parâmetros das regras da firewall para zero, indicando que a firewall está desativada.
- 3) Na Figura 16, foi realizado o reset do switch r4 para aplicar as novas regras.
- 4) Verificar que o ping de h1 para h4 funciona, como representado na Figura 17, o que significa que a firewall está desativada.
- 5) **Nota:** Podemos também verificar que o ping de h4 para h1 também funciona, confirmando que a firewall está completamente desativada e não apenas invertida.

Fig. 15. Alterar as portas para zero no Json do r4. O objetivo, é fazer com que todas as portas se comportem como uma saída (0).

Fig. 16. Fazer reset do R4, para renovar o router 4

Fig. 17. Ping H1 para H4, na firewall normalmente configurada, é bloqueado. Perante as alterações funciona.

F. Teste de conectividade TCP e Portas permitidas na Firewall

- 1) Abrir os terminais para os hosts h1 e h4:

```
1 mininet> xterm h1 h4
```

- 2) No h4, iniciar um servidor HTTP na porta 8080:

```
1 python3 -m http.server 8080
```

- 3) No h1, testar a conectividade TCP ao servidor executado em h4:

```
1 wget 10.0.2.1:8080
```

Também é necessário testar a comunicação em ambos os sentidos, com o h1 a funcionar como servidor (IP: 10.0.1.1, conforme indicado na Tabela I). O fluxo de h1 para h4 deve ser bloqueado pela firewall (Figura 18), enquanto o fluxo de h4 para h1 deve ser permitido (Figura 19). O comando utilizado deverá conseguir aceder ao servidor e realizar o download da página via HTTP, validando assim a comunicação TCP entre os dois hosts.

- 4) Para testar as portas permitidas pela firewall, é necessário configurá-las no ficheiro JSON do r4. Na Figura 20 é possível observar a configuração para permitir a porta 81 no HTTP e a porta 53 no protocolo UDP.
- 5) Testar o envio para a porta 81 do HTTP, com o h1 como cliente e o h4 como servidor. Na Figura 21 apresenta-se o teste realizado. Para qualquer outra porta HTTP, o fluxo deverá ser bloqueado.

```

root@netsin-vm:/home/netsin/RDS/rds# wget 10.0.2.1:8080
--2025-05-31 23:05:51-- http://10.0.2.1:8080/
Connecting to 10.0.2.1:8080... ^C
root@netsin-vm:/home/netsin/RDS/rds#

```

Fig. 18. Pedido HTTP na porta 8080 do H1 para o H4. Este pedido falha porque é bloqueado pela Firewall

```

{
  "MyIngress.allowedPortsTCP": {
    "{ \"hdr.tcp.dstPort\": 81 }": {
      "action": "NoAction",
      "params": {}
    }
  },
  "MyIngress.allowedPortsUDP": {
    "{ \"hdr.udp.dstPort\": 53 }": {
      "action": "NoAction",
      "params": {}
    }
  }
}

```

Fig. 20. Configurações no Json `r4.json`, para permitir a Porta 81 no TCP e a Porta 53 no UDP

G. Teste para verificar o comportamento dos Counters

Este teste tem como objetivo verificar o funcionamento dos *counters* associados aos switches, que contabilizam o número de pacotes que correspondem a cada regra instalada.

- 1) Enviar 5 pacotes do host `h4` para o host `h1`, utilizando o seguinte comando:

```
h4 ping h1 -c 5
```

- 2) Observar os prints no terminal do controller, que devem indicar que os pacotes foram contabilizados pelos *counters*.
- 3) Efetuar o reset dos *counters* para limpar os valores acumulados:

```
reset counters
```

```

[r1-to-r4] up=0, down=0
[r1] up=0, down=0
[r4] up=0, down=0
[r1-to-r4] no switch needed

[r1-to-r4] up=5, down=5
[r1] up=5, down=5
[r4] up=2, down=3
[r1-to-r4] no switch needed

#reset counters
# Reset counter 'tunnel_counter' idx#2 on r1
# Reset counter 'tunnel_counter' idx#3 on r4
# Reset counter 'tunnel_counter' idx#3 on r4
# Reset counter 'tunnel_counter' idx#2 on r4

>>>
[r1-to-r4] up=0, down=0
[r1] up=0, down=0
[r4] up=0, down=0
[r1-to-r4] no switch needed

```

Fig. 22. Contabilização dos *counters* após envio de pacotes.

```

root@netsin-vm:/home/netsin/RDS/rds# wget 10.0.2.1:8080
--2025-05-31 23:05:51-- http://10.0.2.1:8080/
Connecting to 10.0.2.1:8080... ^C
root@netsin-vm:/home/netsin/RDS/rds# python3 -m http.server http.server 8080
usage: server.py [-h] [-c] [-bind ADDRESS] [-directory DIRECTORY] [port]
server.py: error: argument port: invalid int value: http.server
root@netsin-vm:/home/netsin/RDS/rds# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
10.0.2.1 - [21/May/2025:23:09:55] "GET / HTTP/1.1" 200 -

root@netsin-vm:/home/netsin/RDS/rds#

```

Fig. 19. Pedido HTTP na porta 8080 do H4 para o H1. Este pedido passa pela Firewall

```

root@netsin-vm:/home/netsin/RDS/rds# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
^C
Keyboard interrupt received, exiting.
root@netsin-vm:/home/netsin/RDS/rds# wget 10.0.1.1:8080
--2025-05-31 23:09:55-- http://10.0.1.1:8080/
Connecting to 10.0.1.1:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 783 [text/html]
Saving to: 'index.html'

index.html 100%[=====] 783 --KB/s in 0s

2025-05-31 23:09:55 (24.1 MB/s) - 'index.html' saved [783/783]

root@netsin-vm:/home/netsin/RDS/rds#

```

Fig. 21. Ligação HTTP de `h1` para `h4` na porta 81. A comunicação externa foi permitida, confirmando que a porta está aberta.

Na Figura 22, os campos `up` e `down` indicam, respetivamente, o número de pacotes que atravessaram o túnel superior e o túnel inferior entre os dispositivos de rede (routers ou switches). Inicialmente, ambos os contadores estavam a zero. Após o envio de cinco pacotes, foram registados incrementos que confirmam a passagem dos pacotes nos dois sentidos. Os contadores nos túneis superiores representam o total de pacotes transmitidos em cada direção ao longo do túnel.

H. Teste para Verificar o Comportamento dos Túneis com Seleção por Hash

Este teste tem como objetivo verificar o comportamento da seleção de túneis baseada no hash. Quando apenas um dos túneis é bloqueado (por exemplo, através da firewall), espera-se que aproximadamente metade dos pacotes seja descartada. Isto prova o bom funcionamento do load balancing, porque os pacotes são bem distribuídos.

- 1) Alterar a configuração da firewall (no ficheiro `r4.json` como exemplificado na Figura 23) para bloquear o tráfego apenas numa das portas utilizadas pelos túneis.
 - 2) Efetuar um teste de conectividade prolongado entre os hosts:
- ```
1 h1 ping h4 -c 20
```
- 3) Observar o resultado, algo como o `ping` da Figura 24. Deverá ser visível uma perda de pacotes a rondar os 50%, pois apenas um dos dois túneis permanece operacional.

```
{
 "mygress_checkDirection": {
 "meta_ingress_port": 1, "standard_metadata_egress_spec": 23: {
 "action": "mygress_setDirection",
 "params": {
 "dir": 0
 }
 }
 },
 "meta_ingress_port": 1, "standard_metadata_egress_spec": 33: {
 "action": "mygress_setDirection",
 "params": {
 "dir": 0
 }
 },
 "meta_ingress_port": 2, "standard_metadata_egress_spec": 13: {
 "action": "mygress_setDirection",
 "params": {
 "dir": 0
 }
 },
 "meta_ingress_port": 2, "standard_metadata_egress_spec": 33: {
 "action": "mygress_setDirection",
 "params": {
 "dir": 0
 }
 },
 "meta_ingress_port": 3, "standard_metadata_egress_spec": 23: {
 "action": "mygress_setDirection",
 "params": {
 "dir": 0
 }
 },
 "meta_ingress_port": 3, "standard_metadata_egress_spec": 33: {
 "action": "mygress_setDirection",
 "params": {
 "dir": 0
 }
 }
}
```

Fig. 23. Regra alterada na firewall. Inicialmente, a porta 2 do túnel estava bloqueada; com a alteração para `dir="0"`, passou a ser considerada uma porta de saída, permitindo o tráfego.

```
mininet> h1 ping h4
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data:
64 bytes from 10.0.2.1: icmp_seq=1 ttl=63 time=241 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=63 time=14.3 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=63 time=12.9 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=63 time=13.0 ms
64 bytes from 10.0.2.1: icmp_seq=5 ttl=63 time=10.1 ms
64 bytes from 10.0.2.1: icmp_seq=6 ttl=63 time=16.2 ms
64 bytes from 10.0.2.1: icmp_seq=7 ttl=63 time=12.1 ms
64 bytes from 10.0.2.1: icmp_seq=8 ttl=63 time=26.2 ms
64 bytes from 10.0.2.1: icmp_seq=9 ttl=63 time=21.5 ms
64 bytes from 10.0.2.1: icmp_seq=10 ttl=63 time=14.8 ms
64 bytes from 10.0.2.1: icmp_seq=11 ttl=63 time=16.9 ms
64 bytes from 10.0.2.1: icmp_seq=12 ttl=63 time=17.9 ms
64 bytes from 10.0.2.1: icmp_seq=13 ttl=63 time=21.5 ms
64 bytes from 10.0.2.1: icmp_seq=14 ttl=63 time=18.5 ms
64 bytes from 10.0.2.1: icmp_seq=15 ttl=63 time=27.5 ms
64 bytes from 10.0.2.1: icmp_seq=16 ttl=63 time=17.5 ms
64 bytes from 10.0.2.1: icmp_seq=17 ttl=63 time=19.8 ms
64 bytes from 10.0.2.1: icmp_seq=18 ttl=63 time=15.0 ms
64 bytes from 10.0.2.1: icmp_seq=19 ttl=63 time=12.8 ms
^C
- - - 10.0.2.1 ping statistics - - -
37 packets transmitted, 19 received, 48.6486% packet loss, time 3683ms
rtt min/avg/max/mdev = 10.084/29.123/240.951/50.279 ms
mininet>
```

Fig. 24. Ping de `h1` para `h4`. Com a firewall configurada para bloquear apenas uma das portas, cerca de metade dos pacotes é encaminhada com sucesso pelas restantes portas disponíveis. Numa configuração padrão, todos os pacotes seriam bloqueados.

#### I. Load Balancing

Este teste tem como objetivo avaliar o comportamento do mecanismo de load balancing entre túneis, por meio da modificação das regras de hash definidas no programa `P4`.

- 1) Modificar a configuração do algoritmo de hash no `P4 mslp`, alterando os túneis utilizados para os labels 0 e 1, em vez de 0 e 2, conforme ilustrado na Figura 25.
- 2) Recompilar o programa `P4` para aplicar as alterações na configuração.
- 3) Executar o controlador, garantindo o carregamento das novas regras no sistema.
- 4) Como na Figura 26, verificar as regras ativas no router `r1`, confirmando a atualização para os túneis com labels 0 e 1, com o seguinte comando:

```
1 show r1
```

- 5) Realizar um ping contínuo de h4 para h1, mantendo a comunicação ativa por alguns segundos.
- 6) Consultar os counters após um período de execução, observando a contagem de pacotes para cada túnel.
- 7) Confirmar que os pacotes trafegam exclusivamente por um dos túneis, indicado pelo aumento do contador correspondente, enquanto o outro permanece inalterado. Este comportamento reflete a distribuição dos pacotes baseada no algoritmo de hash.
- 8) Nas Figuras 27 e 28, os logs do controlador evidenciam que o estado do sistema foi atualizado e que o túnel preferencial foi corretamente selecionado.
- 9) Executar novamente o comando `show r1` para verificar as novas regras aplicadas no router r1, conforme ilustrado na Figura 27.

```

action selectTunnel(bit<16> srcPort, bit<16> dstPort) {
 hash{
 meta.tunnel,
 HashAlgorithm.crc32,
 (bit<2>)0,
 {
 hdr.ipv4.protocol,
 hdr.ipv4.srcAddr,
 hdr.ipv4.dstAddr,
 srcPort,
 dstPort
 },
 (bit<2>)1
 },
 {
 (bit<2>)0
 }
}

```

Fig. 25. Alteração do algoritmo de hash no programa P4, configurando os túneis para utilizar os labels 0 e 1.

```

[1] r1-r4: up=29, down=31
[2] r1: up=15, down=15
[3] r4: up=14, down=16
[4] r1-to-r4: no switch needed

```

Fig. 27. Regras atualizadas no r1, refletindo a aplicação do novo algoritmo de hash e a utilização dos túneis corretos.

```

[1] r1-r4: up=29, down=31
[2] r1: up=15, down=15
[3] r4: up=14, down=16
[4] r1-to-r4: no switch needed

```

Fig. 26. Estado inicial das regras no r1, exibindo os túneis configurados antes da modificação das labels.

```

[r1-to-r4] up=29, down=31
[r1] up=15, down=15
[r4] up=14, down=16
[r1-to-r4] no switch needed

```

Fig. 28. Logs do controller após a alteração dos estados. Quando o tráfego está desequilibrado, o controller tenta redistribuir os pacotes entre os túneis. Nesta situação, foi alcançado o equilíbrio; se voltar a ocorrer um desequilíbrio, o controller ajusta novamente, redirecionando o tráfego conforme necessário.



#### IV. TRABALHO FUTURO

Apesar de bastante completo, ainda há muitas melhorias que podem ser feitas neste projeto para o tornar mais completo ainda e mais robusto.

Uma delas seria a implementação de um mecanismo para reinicializar todos os *bloom filters* periodicamente, mediante alcance de um limiar estabelecido ou manualmente através de um comando como os restantes comandos. É recomendado fazer limpezas periódicas aos *bloom filters* para evitar acumulação de *flows* obsoletos que possam provocar coincidências indesejadas e permitir a passagem pela *firewall* de fluxos não autorizados. No entanto, dado que a topologia que desenvolvemos é pequena e não tem muito tráfego, consideramos que o impacto é insignificante. Ainda assim, para um escala maior seria recomendado implementar um método de limpeza dos filtros.

#### V. CONCLUSÃO

Este trabalho prático demonstrou com sucesso a implementação de uma rede definida por software utilizando P4 para configurar o *data plane* e um controlador dinâmico em Python para gerir o *control plane*. Através da criação de um protocolo próprio, foi possível implementar túneis para encaminhamento eficiente de pacotes baseado em labels, reduzindo a complexidade e a latência em comparação com o encaminhamento tradicional por IP. A definição de uma *firewall* e de um controlador permitiram relembrar, consolidar e até adquirir novos conhecimentos sobre o funcionamento da rede que usamos todos os dias.

Os testes realizados validaram a robustez e a flexibilidade da implementação, destacando-se:

- A capacidade do controlador manter uma visão do estado da rede entre execuções, evitando reinstalações desnecessárias;
- A aprendizagem dinâmica de endereços MAC pelo switch L2;
- A configuração dinâmica da *firewall* para proteger interfaces específicas e permitir exceções de tráfego para portas estabelecidas;
- O correto funcionamento dos counters para monitorização de tráfego;
- A eficácia do *load balancing*, tanto baseado na hash de seleção do túnel como na troca de labels dos túneis pelo controlador para equilibrar o tráfego.

Apesar de limitações, como a restrição a túneis entre duas LANs e a ausência de um mecanismo automático de limpeza dos *bloom filters*, os resultados confirmam a viabilidade da solução para cenários de redes programáveis.

Este projeto reforçou a importância da programabilidade nas redes modernas, evidenciando as vantagens do P4 na personalização do comportamento dos dispositivos de rede e da integração com um controlador dinâmico para gestão em tempo real. Em suma, este trabalho prático proporcionou uma compreensão prática e aprofundada dos conceitos de Redes Definidas por Software e da sua aplicação em redes de alto desempenho.