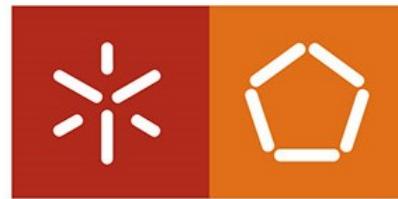


Redes Definidas por Software

2023/2024

Trabalho Prático 2
Grupo 7

L3 & L4 Packet Manipulation



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. Redes Definidas por Software (SDN) revolucionaram o design, implementação e gestão de redes ao separar o plano de controlo do plano de dados, permitindo uma gestão mais programável e dinâmica. O P4 (Programming Protocol-independent Packet Processors) é uma linguagem que define como dispositivos de rede processam pacotes, permitindo especificar o comportamento de encaminhamento de forma granular e implementar funcionalidades personalizadas. O Mininet é uma plataforma de emulação de rede de código aberto que facilita a criação de redes virtuais de teste num único computador, simplificando a simulação de topologias complexas e o teste de diversos cenários de rede.

Keywords: P4 · OVSSwitch · Router · Host · LAN · ICMP · Ping.

1 Introdução

Neste segundo trabalho, o objetivo é implementar novas funcionalidades nos routers da topologia original do TP1, mais especificamente a capacidade de os routers conseguirem responder a pedidos em ICMP (pings), em qualquer uma das suas interfaces. O exercício é baseado no conhecimento adquirido no trabalho anterior e introduz novos conceitos e desafios. Com a sua resolução, esperamos ter uma compreensão abrangente sobre a construção de planos de dados com P4 e a integração deles em ambientes SDN. A topologia sofreu ligeiras alterações, nomeadamente a atribuição de novos endereços IPv4 em todas as portas dos routers r1, r2 e r3.

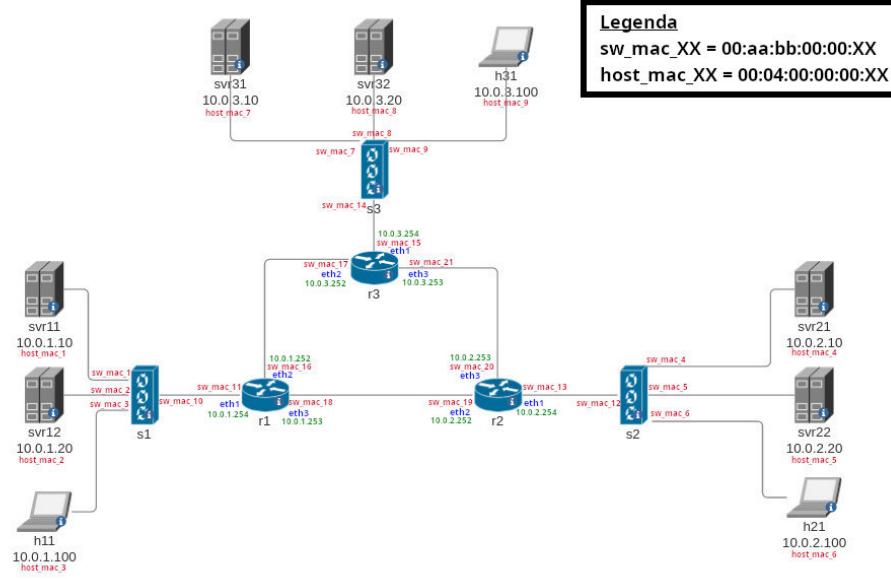


Fig. 1: Nova topologia em uso

2 Desenvolvimento

2.1 Alterações na topologia

O cenário em estudo é precisamente o mesmo que o do primeiro trabalho prático, ou seja, 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). As novas funcionalidades devem ser implementadas mantendo as regras da firewall herdadas do cenário original.

Relembrando o trabalho anterior, 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. A seguir 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. Cada router possui um *thrift port* distinto, sendo que r1 fica com 9090, r2 com 9091 e r3 com 9092.

Para a resolução do primeiro exercício, foram introduzidos novos endereços IPv4 que serão considerados os endereços das interfaces *eth2* (10.0.X.252) e *eth3* (10.0.X.253) em cada um dos routers. Vale a pena relembrar que em cada router (r1, r2 e r3), *eth1* já tem um endereço atribuído (no formato 10.0.X.254), que foi herdado do primeiro trabalho, onde *rX-eth1* atua como gateway dos hosts na LAN_X.

2.2 Implementação das respostas ICMP

Como o exercício necessita de manipular e analisar cabeçalhos ICMP, foi necessário adicionar uma nova estrutura que permitisse guardar a informação retirada do pacote recebido após o processo de parsing.

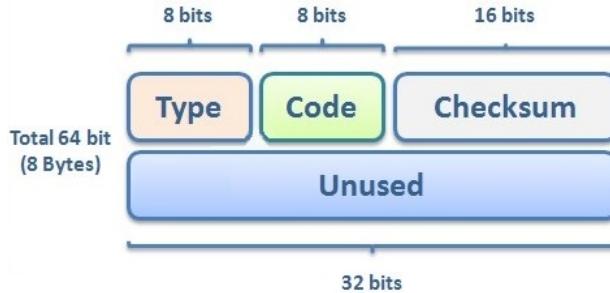


Fig. 2: Cabeçalho ICMP

A estrutura em questão, denominada *header icmp_t*, contem três campos:

- ***type*** - 8 bits para definir o tipo de mensagem ICMP, indicando a finalidade e a natureza do pacote. Existem diversos tipos de mensagens, como Echo Request (tipo 8) e Echo Reply (tipo 0), usados no comando "ping", Destination Unreachable (tipo 3), Time Exceeded (tipo 11), entre outros.
- ***code*** - 8 bits para proporcionar informações adicionais sobre o tipo de mensagem ICMP. Por exemplo, para o tipo "Destination Unreachable" (tipo 3), o código pode especificar se o destino é inacessível devido a uma rede inacessível (código 0), um host inacessível (código 1), ou outros motivos. Apesar de ser definido na estrutura, nunca chega a ser alterado na resolução deste trabalho.
- ***checksum*** - 16 bits utilizados para verificar a integridade do pacote ICMP. O checksum é um valor de verificação de erro calculado a partir dos dados do pacote. O receptor recalculará o checksum para garantir que o pacote não foi corrompido durante a transmissão.

Omitimos propositadamente os 32 bits do campo de dados, *data*, por não terem qualquer tipo de relevância para a resolução deste trabalho. O código em baixo ilustra a estrutura aplicada no parsing.

(...)

```
header icmp_t {
    bit<8> type;
    bit<8> code;
    bit<16> checksum;
```

```

}

(...)

state parse_ipv4 {
    packet.extract(hdr.ipv4); // --> populates the ipv4 header
    transition select(hdr.ipv4.protocol){
        TYPE_TCP: parse_tcp;
        TYPE_ICMP: parse_icmp;
        default: accept;
    }
}

state parse_tcp {
    packet.extract(hdr.tcp); // --> populates the tcp header
    transition accept;
}

state parse_icmp {
    packet.extract(hdr.icmp); // --> populates the icmp header
    transition accept;
}

(...)

```

Para permitir que os routers sejam capazes de responder a pedidos de ICMP, foi necessário adicionar uma nova tabela e uma nova ação ao ficheiro P4, renomeado router_tp2.p4, que expande o P4 original do TP1. A tabela *icmp_for_router* permite definir regras que especificam quais são os endereços IPv4 onde podem ocorrer pedidos com o protocolo ICMP.

```

(...)

table icmp_for_router {
    key = {hdr.ipv4.dstAddr : exact; hdr.ipv4.protocol : exact;}
    actions = {
        respond_icmp;
        NoAction;
    }
    default_action = NoAction;
}

```

(...)

Segue-se agora um exemplo de uma regra desta tabela, onde é especificado que qualquer pacote com destino à interface eth2 do router r1 deve ser tratado na ação *respond_icmp*. Esta regra deve ser aplicado no router r1.

```
table_add icmp_for_router respond_icmp 10.0.1.252 0x01 =>
```

A ação *respond_icmp* faz a manipulação do pacote recebido para que possa ser devolvido no nodo original. A alteração consiste em trocar os endereços IP e MAC, trocar o valor do campo *type* de 8 (Echo) para 0 (Echo Reply) e alterar o campo do checksum, adicionando o valor 2048 ao valor original. O valor 2048 foi escolhido porque representa a correção necessária no cálculo do checksum devido à alteração no campo *type* do ICMP. Esta alteração no campo *type* de 8 para 0 modifica a soma em complemento de um, e o ajuste de 2048 compensa essa mudança para manter o valor correto do checksum. No entanto, uma prática mais precisa seria recalcular o checksum completamente após quaisquer mudanças nos campos do cabeçalho ICMP para garantir a integridade dos dados.

```
(...)
```

```
action respond_icmp () {
    // Swap source and destination IP addresses
    bit<32> tmp_ip;
    tmp_ip = hdr.ipv4.srcAddr;
    hdr.ipv4.srcAddr = hdr.ipv4.dstAddr;
    hdr.ipv4.dstAddr = tmp_ip;

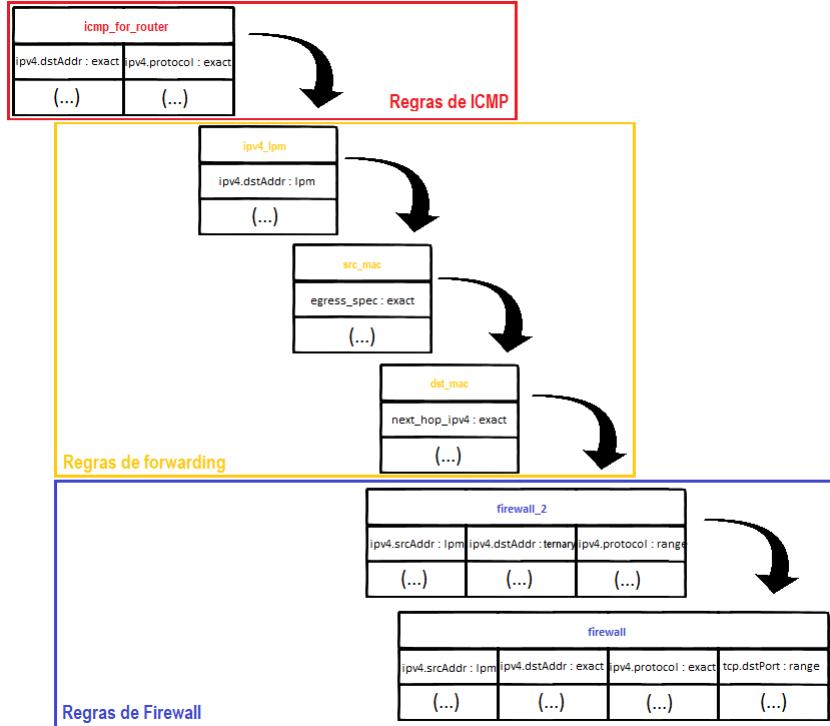
    // Swap source and destination MAC addresses
    bit<48> tmp_mac;
    tmp_mac = hdr.ethernet.srcAddr;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = tmp_mac;

    // Set ICMP type to Echo Reply (0)
    hdr.icmp.type = 0;

    // Calculate checksum
    hdr.icmp.checksum = hdr.icmp.checksum + 2048;
}
```

```
(...)
```

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



Ou seja, primeiro são aplicadas as regras relativas à capacidade de resposta a pedidos ICMP, depois realiza-se o encaminhamento para os pacotes e só no final é que se aplica as regras de exclusão de tráfego da firewall (nas tabelas `firewall_2` e `firewall`).

2.3 Testes Realizados E Discussão

Para garantir que a implementação das respostas a pedidos ICMP por parte dos routers funciona e não resulta no desrespeito das regras previamente definidas da firewall, foi elaborado um conjunto de testes.

2.4 Teste de Comunicação ICMP Host-Router

Começamos por testar que de facto os hosts conseguem todos comunicar em ICMP com qualquer interface de qualquer router. Comecemos por fazer um pedido ICMP através do Ping com origem no host `h11` com IP `10.0.1.100` para a interface `r2-eth3` com IP `10.0.2.253`. O resultado obtido segue em baixo, onde é possível observar que o pacote é enviado e recebe uma resposta. Abrindo as tramas 1 e 2 dos pacotes capturados, é possível observar a correta alteração no pacote, com o tipo alterado de 8 para 0 e o valor de checksum correto.

```

mininet> h1 ping -c 3 10.0.2.253
PING 10.0.2.253 (10.0.2.253) 56(84) bytes of data.
64 bytes from 10.0.2.253: icmp_seq=1 ttl=61 time=1.82 ms
64 bytes from 10.0.2.253: icmp_seq=2 ttl=61 time=1.08 ms
64 bytes from 10.0.2.253: icmp_seq=3 ttl=61 time=2.23 ms

--- 10.0.2.253 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.076/1.708/2.226/0.476 ms
mininet>

```

No.	Time	Source	Destination	Protocol	Length Info
1	0.0000000000	10.0.1.100	10.0.2.253	ICMP	98 Echo (ping) request id=0xf4e1, seq=1/256, ttl=64 (reply in 2)
2	0.001771381	10.0.2.253	10.0.1.100	ICMP	98 Echo (ping) reply id=0xf4e1, seq=1/256, ttl=61 (request in 1)
3	1.002385884	10.0.1.100	10.0.2.253	ICMP	98 Echo (ping) request id=0xf4e1, seq=2/512, ttl=64 (reply in 4)
4	1.003418560	10.0.2.253	10.0.1.100	ICMP	98 Echo (ping) reply id=0xf4e1, seq=2/512, ttl=61 (request in 3)
5	2.002899099	10.0.1.100	10.0.2.253	ICMP	98 Echo (ping) request id=0xf4e1, seq=3/788, ttl=64 (reply in 6)
6	2.005058906	10.0.2.253	10.0.1.100	ICMP	98 Echo (ping) reply id=0xf4e1, seq=3/788, ttl=61 (request in 5)

Fig. 3: h11, ping, r2-eth3

```

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface r1-eth1, id 0
▶ Ethernet II, Src: LexmarkP_00:00:03 (00:04:00:00:00:03), Dst: 00:aa:bb:00:00:11 (00:aa:bb:00:00:11)
▶ Internet Protocol Version 4, Src: 10.0.1.100, Dst: 10.0.2.253
└ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xcef6 [correct]
  [Checksum Status: Good]
  Identifier (BE): 39085 (0x98ad)
  Identifier (LE): 44440 (0xad98)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Jun 11, 2024 15:59:51.000000000 BST
  [Timestamp from icmp data (relative): 0.179100382 seconds]
  Data (48 bytes)

```

```

> Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface r1-eth1, id 0
  > Ethernet II, Src: 00:aa:bb:00:00:11 (00:aa:bb:00:00:11), Dst: LexmarkP_00:00:03 (00:04:00:00:00:03)
  > Internet Protocol Version 4, Src: 10.0.2.253, Dst: 10.0.1.100
    > Internet Control Message Protocol
      Type: 0 (Echo (ping) reply)
      Code: 0
      Checksum: 0xd6f6 [correct]
      [Checksum Status: Good]
      Identifier (BE): 39085 (0x98ad)
      Identifier (LE): 44440 (0xad98)
      Sequence Number (BE): 1 (0x0001)
      Sequence Number (LE): 256 (0x0100)
      [Request frame: 1]
      [Response time: 1.272 ms]
      Timestamp from icmp data: Jun 11, 2024 15:59:51.000000000 BST
      [Timestamp from icmp data (relative): 0.180372120 seconds]
    > Data (48 bytes)

```

Fig. 4: Trama 1 e 2 captadas em r1-eth1

A seguir testou-se comunicação ICMP entre h21 (10.0.2.100) e r1-eth2 (10.0.1.252) tendo-se obtido com sucesso o seguinte resultado.

```

mininet> h21 ping -c 3 10.0.1.252
PING 10.0.1.252 (10.0.1.252) 56(84) bytes of data.
64 bytes from 10.0.1.252: icmp_seq=1 ttl=61 time=3.82 ms
64 bytes from 10.0.1.252: icmp_seq=2 ttl=61 time=2.47 ms
64 bytes from 10.0.1.252: icmp_seq=3 ttl=61 time=1.68 ms

--- 10.0.1.252 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.680/2.655/3.821/0.884 ms
mininet>

```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	ff:ff:ff:ff:ff:ff	ff:ff:ff:ff:ff:ff	ICMP	98	Router Solicitation from 00:aa:bb:00:00:12
2	4.724323097	10.0.2.100	10.0.1.252	ICMP	98	Echo (ping) request id=0xb001, seq=1/256, ttl=64 (reply in 3)
3	4.72621925	10.0.1.252	10.0.2.100	ICMP	98	Echo (ping) reply id=0xb001, seq=1/256, ttl=61 (request in 2)
4	5.72479177	10.0.2.100	10.0.1.252	ICMP	98	Echo (ping) request id=0xb001, seq=2/512, ttl=64 (reply in 5)
5	5.727190931	10.0.1.252	10.0.2.100	ICMP	98	Echo (ping) reply id=0xb001, seq=2/512, ttl=61 (request in 4)
6	6.727028553	10.0.2.100	10.0.1.252	ICMP	98	Echo (ping) request id=0xb001, seq=3/768, ttl=64 (reply in 7)
7	6.728640555	10.0.1.252	10.0.2.100	ICMP	98	Echo (ping) reply id=0xb001, seq=3/768, ttl=61 (request in 6)

Fig. 5: h11, ping, r1-eth2

Finalmente testou-se um ping entre um host e o seu router adjacente, servindo como origem o h31 (10.0.3.100) e como destino o r3-eth1 (10.0.3.254). O resultado obtido foi novamente positivo.

```
mininet> h31 ping -c 3 10.0.3.254
PING 10.0.3.254 (10.0.3.254) 56(84) bytes of data.
64 bytes from 10.0.3.254: icmp_seq=1 ttl=63 time=1.13 ms
64 bytes from 10.0.3.254: icmp_seq=2 ttl=63 time=0.504 ms
64 bytes from 10.0.3.254: icmp_seq=3 ttl=63 time=0.679 ms

--- 10.0.3.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 0.504/0.771/1.131/0.264 ms
mininet>
```

No.	Time	Source	Destination	Protocol	Length Info
1	0.0000000000	10.0.3.100	10.0.3.254	ICMP	98 Echo (ping) request id=0xdeda, seq=1/256, ttl=64 (reply in 2)
2	0.000619830	10.0.3.254	10.0.3.100	ICMP	98 Echo (ping) reply id=0xdeda, seq=1/256, ttl=63 (request in 1)
3	1.002634723	10.0.3.100	10.0.3.254	ICMP	98 Echo (ping) request id=0xdeda, seq=2/512, ttl=64 (reply in 4)
4	1.0030997709	10.0.3.254	10.0.3.100	ICMP	98 Echo (ping) reply id=0xdeda, seq=2/512, ttl=63 (request in 3)
5	2.011274464	10.0.3.100	10.0.3.254	ICMP	98 Echo (ping) request id=0xdeda, seq=3/768, ttl=64 (reply in 6)
6	2.011881378	10.0.3.254	10.0.3.100	ICMP	98 Echo (ping) reply id=0xdeda, seq=3/768, ttl=63 (request in 5)

Fig. 6: h31, ping, r3-eth1

2.5 Teste de Comunicação ICMP HostX-HostY

Agora partimos para um teste que permite garantir que as regras da firewall continuam a ser cumpridas apesar da injeção das novas regras ICMP. Aqui, iremos tentar estabelecer uma comunicação em ICMP entre dois hosts de diferentes LANs, nomeadamente o h11 (10.0.1.100) e o h21 (10.0.2.100).

```
mininet> h11 ping -c 3 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
|
```

Fig. 7: h11, ping, h21

No.	Time	Source	Destination	Protocol	Length Info
1	0.0000000000	10.0.1.100	10.0.2.100	ICMP	98 Echo (ping) request id=0x01a4, seq=1/256, ttl=64 (no response found!)
2	1.031674485	10.0.1.100	10.0.2.100	ICMP	98 Echo (ping) request id=0x01a4, seq=2/512, ttl=64 (no response found!)
3	2.057121894	10.0.1.100	10.0.2.100	ICMP	98 Echo (ping) request id=0x01a4, seq=3/768, ttl=64 (no response found!)

No.	Time	Source	Destination	Protocol	Length Info
1	0.0000000000	10.0.1.100	10.0.2.100	ICMP	98 Echo (ping) request id=0x2eb4, seq=1/256, ttl=63 (no response found!)
2	1.010466780	10.0.1.100	10.0.2.100	ICMP	98 Echo (ping) request id=0x2eb4, seq=2/512, ttl=63 (no response found!)
3	2.033649556	10.0.1.100	10.0.2.100	ICMP	98 Echo (ping) request id=0x2eb4, seq=3/768, ttl=63 (no response found!)

Fig. 8: Pacotes captados em r1 e r2

Como é possível ver nas imagens acima, o h11 não obteve resposta ICMP do h21, sendo possível observar nos pacotes captados que tanto num lado como outro, os routers não sabem responder a um pedido ICMP que não seja exclusivamente nas interfaces dos routers. Logo, fica comprovado que a firewall continua ativa e a ser respeitada.

2.6 Dificuldades

No que diz respeito a dificuldades sentidas durante a resolução do trabalho, a maior de todas foi a falha na implementação do exercício 2 do enunciado, que pedia a introdução de um Load Balancer com Network Address Translation (NAT). Restrições de tempo impediram a execução desta fase, bem como o desconhecimento de como tal tarefa poderia ser executada.

Outra dificuldade que deveria ter sido mencionada no trabalho anterior, consiste na forma pouco ortodoxa como foi implementada a firewall. Ao invés de apenas indicar quais os pacotes que devem ser aceites e aplicar as ações devidas, por motivos que não sabemos explicar, esta estratégia não funcionou. Tivemos assim de seguir uma rota mais verbosa, escrevendo inúmeras regras de exclusão, cuja ação envolvia descartar os pacotes que não devem ser aceites.

Existe também a questão do cálculo do valor do checksum, que é feita de forma *hardcoded*. Idealmente, o checksum deveria ser recalculado utilizando uma função de hash, mas não se conseguiu realizar este passo, tendo-se por isso recorrido a uma alternativa que, apesar de funcionar no nosso cenário, poderia não funcionar em qualquer outro.

3 Conclusão

Neste relatório, abordamos a implementação de routers simples capazes de responder a pedidos ICMP utilizando P4, no contexto de Redes Definidas por Software (SDN). O exercício realizado proporcionou uma compreensão prática e aprofundada sobre a construção de planos de dados e a sua integração em ambientes SDN. Este desafio envolveu a identificação e manipulação de campos específicos de cabeçalhos. Durante o desenvolvimento, foi crucial entender a estrutura dos pacotes e a forma de processá-los com P4, destacando a importância de manipular mensagens ICMP para a funcionalidade básica de redes. Os testes realizados no Mininet confirmaram a eficácia da implementação. Os routers simples responderam corretamente aos pedidos ICMP, demonstrando a precisão na manipulação dos pacotes. Estes testes validaram a robustez da solução desenvolvida e a aplicabilidade dos conceitos aprendidos. Em suma, o projeto demonstrou como a programação de redes com P4 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>