TRABALHO PRÁTICO *Redes Definidas por Software*

TECHNICAL REPORT

Ivo Miguel Alves Ribeiro
Mestrado em Engenharia Informática
Universidade do Minho
Pg53886
pg53886@alunos.uminho.pt

Ricardo Lopes Santos Silva
Mestrado Integrado em Engenharia Informática
Universidade do Minho
pg54188
pg54188@alunos.uminho.pt

Paulo Miguel Freitas Oliveira
Mestrado Integrado em Engenharia Informática
Universidade do Minho
Pg54133
pg54133@alunos.uminho.pt

15 de junho de 2024

ABSTRACT

A segurança de redes é um aspeto crucial para gerir infraestruturas de TI, especialmente em ambientes que demandam alta disponibilidade e proteção contra ameaças externas. Este trabalho é dividido em três fases, cada uma focada num aspeto específico da configuração de uma rede segura e eficiente.

Na **primeira fase**, iremos definir a nossa topologia de testes que nos acompanhará em todas as etapas e será realizada a construção de uma *firewall*. Esta etapa é fundamental para estabelecer uma barreira de proteção inicial, permitindo controlar e monitorizar o tráfego de entrada e saída nos diferentes *routers*.

Na **segunda fase**, o foco será permitir o tráfego *ICMP (Internet Control Message Protocol)* para todas as interfaces dos *routers*. O *ICMP* é essencial para o diagnóstico e solução de problemas na rede, permitindo operações como *ping* e *traceroute*, que ajudam a identificar problemas de conectividade e desempenho.

Por fim, na **terceira fase**, será implementado um *Controller* capaz de injetar regras nos *routers* dinamicamente. Este *Controller* permitirá a aplicação e atualização de políticas de segurança de forma centralizada e eficiente, facilitando a gestão de regras e a resposta rápida a incidentes de segurança.

Cada uma dessas fases é projetada para aumentar progressivamente a segurança, a funcionalidade e a gestão da rede, culminando num sistema robusto e adaptável às necessidades dinâmicas de um ambiente de TI moderno.

Keywords *firewall* · tráfego *ICMP* · *Controller* · sistema robusto e adaptável

Primeira Fase (TP1)

1 Introdução

No contexto deste projeto, desenvolvemos um *firewall stateful* que opera sobre Camada 3 (L3) e Camada 4 (L4) utilizando linguagem P4 (*Programming Protocol-independent Packet Processors)*.

Iremos definir uma topologia que servirá de apoio a uma implementação de uma rede da *TechSecure Inc.*, que envolve três *LANs* distintas:

- Vendas (LAN1)
- Pesquisa & Desenvolvimento (LAN2)
- Gestão (LAN3)

Isto requer um controlo meticuloso sobre o tráfego de rede, assegurando que apenas as comunicações autorizadas ocorram entre servidores e *hosts* específicos.

Os principais problemas desta fase incluem uma configuração e integração da topologia, onde iremos implementar uma topologia de rede num ambiente virtualizado usando *Mininet* e dispositivos *BMv2 P4*. Para além disso, implementaremos Regras de *Firewall*, onde aceitaremos ou rejeitaremos pacotes com base em portas *TCP/UDP* e endereços *IP* específicos. Por fim, descreveremos os testes de validação ao nosso ambiente, de modo a mostrar o funcionamento do nosso sistema e garantir que todas as regras de segurança estejam a funcionar conforme o esperado. Assim, incluiremos capturas de tráfego de pacotes para verificar se os pacotes foram aceites ou rejeitados assim como uma análise detalhada de como as regras foram aplicadas nas diferentes situações de tráfego.

Todos os problemas que tivemos nesta fase serão identificados, destacando a complexidade envolvida na criação de uma solução de *firewall stateful* utilizando P4, exigindo o conhecimento de técnicas avançadas e ainda um conhecimento mais profundo das redes definidas por software.

2 Descrição da Topologia

Nesta secção, vai ser abordada a topologia virtualizada através do **Mininet**, que é baseada na topologia que é descrita no enunciado deste TP, descrita e exemplificada na figura que se segue.

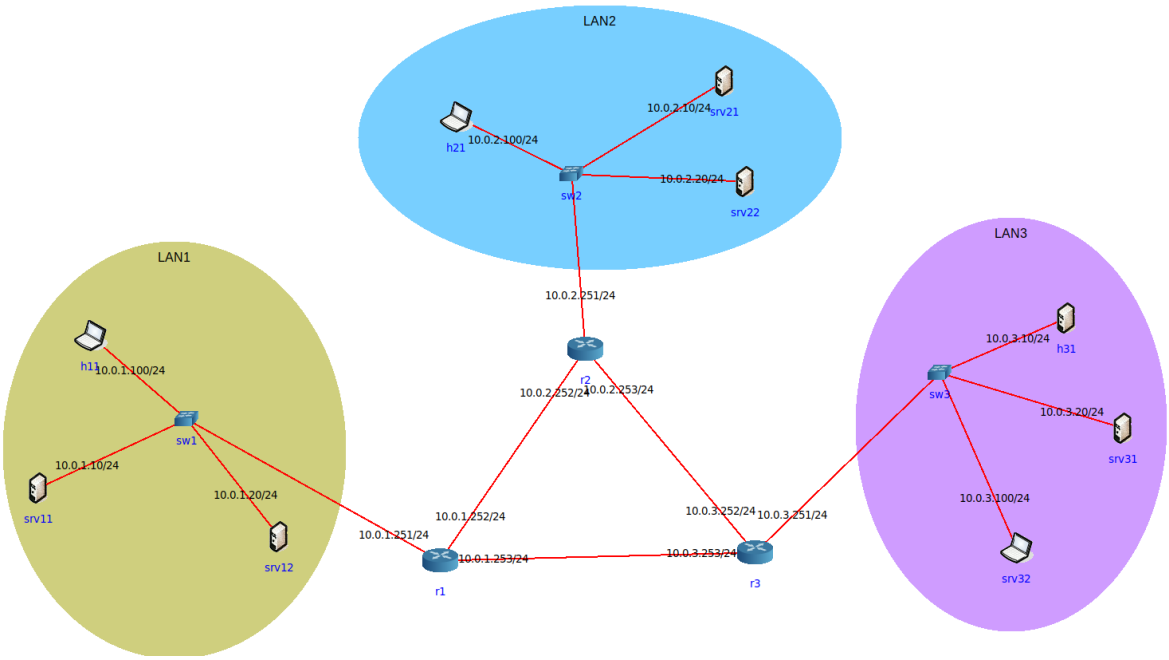

Figura 1: Topologia

Como podemos ver pela imagem acima exposta, temos 3 *LANs*, cada uma com 1 *host* e 2 servidores, ambos conectados a um *Open vSwitch (OVS)*. Utilizamos os *OVS* para conectar os dispositivos dentro de cada *LAN* aos *routers P4 BMv2* que desempenham o papel de *gateways* entre as *LANs*. Cada *router* é um *switch_p4* por nós implementado, isto é, é um *router* que responde a regras, também estas por nós descritas, de modo a operar em conformidade com as ações que são necessárias realizar.

3 Comunicação na Topologia

Depois de definida a forma da topologia, mostrava-se necessário implementá-la e garantir que exista comunicação entre os vários dispositivos da mesma.

A esta etapa chamamos de Fase 0 do projeto, uma vez que, apesar de simples, foi necessário um conhecimento sobre as técnicas de programação em *SDNs*, como perceber de que forma se preenchem as tabelas de encaminhamento nos *routers* para permitir o envio de pacotes na rede, bem como garantir que os *OVSs* permitam que os pacotes sejam enviados e recebidos pelos dispositivos em cada uma das redes.

Após esta etapa, realizamos testes que permitem visualizar essa comunicação, dos quais destacamos os seguintes:

```
mininet> pingall
*** Ping: testing ping reachability
h11 -> h21 h31 sv11 sv12 sv21 sv22 sv31 sv32
h21 -> h11 h31 sv11 sv12 sv21 sv22 sv31 sv32
h31 -> h11 h21 sv11 sv12 sv21 sv22 sv31 sv32
sv11 -> h11 h21 h31 sv12 sv21 sv22 sv31 sv32
sv12 -> h11 h21 h31 sv11 sv21 sv22 sv31 sv32
sv21 -> h11 h21 h31 sv11 sv12 sv22 sv31 sv32
sv22 -> h11 h21 h31 sv11 sv12 sv21 sv31 sv32
sv31 -> h11 h21 h31 sv11 sv12 sv21 sv22 sv32
sv32 -> h11 h21 h31 sv11 sv12 sv21 sv22 sv31
*** Results: 0% dropped (72/72 received)
mininet>
```

Figura 2: Conectividade *ICMP* entre todos os dispositivos da topologia

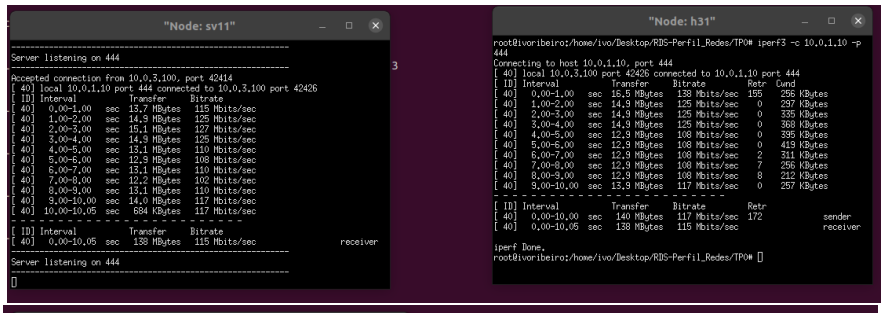

Figura 3: Comunicação *TCP* entre h31 e sv11 na porta 44

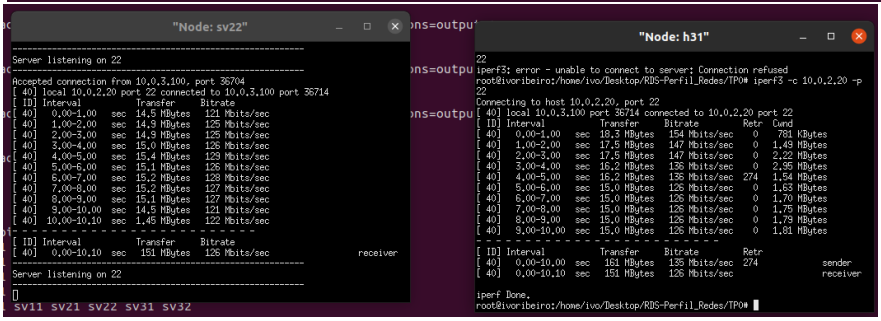

Figura 4: Comunicação *TCP* entre h31 e sv22 na porta 22

Como podemos observar, não há nenhuma limitação nas comunicações na nossa topologia nesta fase, sendo possível estabelecer comunicações entre os diferentes dispositivos segundo protocolos como *TCP* e *ICMP*, sem problemas.

4 Implementação da Firewall

Após garantir a total funcionalidade da nosso topologia, iremos implementar uma *firewall* nos *routers* de modo a conseguir controlar parte do tráfego na nossa topologia.

No nosso caso, a *firewall* por nós implementada irá apenas bloquear tráfego *TCP* entre os dispositivos. Optamos por esta alternativa de forma a conseguirmos realizar testes de conectividade entre dispositivos através de *ICMP*, de modo a garantir que realmente está a ser ou não bloqueado o tráfego *TCP*.

Nesta fase, e de modo a facilitar a compreensão do sistema por nós desenvolvido, não abdicamos das regras de *forwarding* de pacotes que implementamos anteriormente. Assim, garantimos que temos uma verificação extra após estas serem aplicadas, o que permite definir se um pacote deve ser ou não descartado, controlando o tráfego *TCP* na nossa rede.

Quanto ao cenário por nós implementado, este é idêntico ao fornecido no enunciado, em que destacamos:

- **Sales** (LAN1):
  - Permite acesso ao CRM system (sv11), segundo TCP na porta 443;
  - Permite acesso aos email servers (sv12), segundo TCP na porta 25;
  - Tem acesso limitado à R&D (LAN2) e à Management (LAN3) para serviços específicos, como:
    - Acesso ao Research Server 1 (sv21), segundo TCP na porta 80.
    - Acesso ao Financial Data Server 1 (sv31), segundo TCP na porta 8080.
- **Research & Development (R&D)** (LAN2):
  - Permite acesso research Server 1 (sv21), segundo TCP na porta 8080;
  - Permite acesso research Server 2 (sv22), segundo TCP na porta 22;
  - Tem acesso limitado à Sales (LAN1) e à Management (LAN3) para serviços específicos, como:
    - Acesso ao Email server (sv12), segundo TCP na porta 25.
    - Acesso ao Financial Data Server 2 (sv32), segundo TCP na porta 443.
- **Management** (LAN3):
  - Permite acesso aos financial data server 1 (sv31), segundo TCP na porta 8080;
  - Permite acesso aos financial data server 2 (sv32), segundo TCP na porta 443;
  - Deve ter acesso limitado a Sales (LAN1) e P&D (LAN2) para serviços específicos:
    - Acesso ao CRM system (sv11), segundo TCP na porta 443.
    - Acesso ao Research Server 2 (sv22), segundo TCP na porta 22.

De notar que, dentro da própria *LAN*, não há qualquer aplicação de regras de *firewall* e, portanto, todo o tipo de comunicações dentro da própria *LAN* são sempre permitidas.

Em suma, a nossa estratégia consistiu em criar uma nova tabela que é aplicada sempre que o pacote corresponde a um pacote sobre o *protocol TCP*. Essa tabela é então preenchida com combinações de endereços *source* e *destination* em portas também eles pré-definidas. Assim, sempre que o pacote der *match* com uma das combinações possíveis para o *router* em questão, é então aplicada a função *NoAction do core p4*, que não faz qualquer alteração ao pacote, deixando-o seguir o seu caminho pré-definido e já recalculado pelas regras de *forwarding*. Caso o pacote que chega a um *router* não der *match* com nenhuma das combinações nele possíveis, é descartado.

5 Cenários de Testes Firewall

Concluindo a configuração dos cenários acima descritos, testamos o funcionamento dos mesmos e desse teste seguem-se as seguintes conclusões:

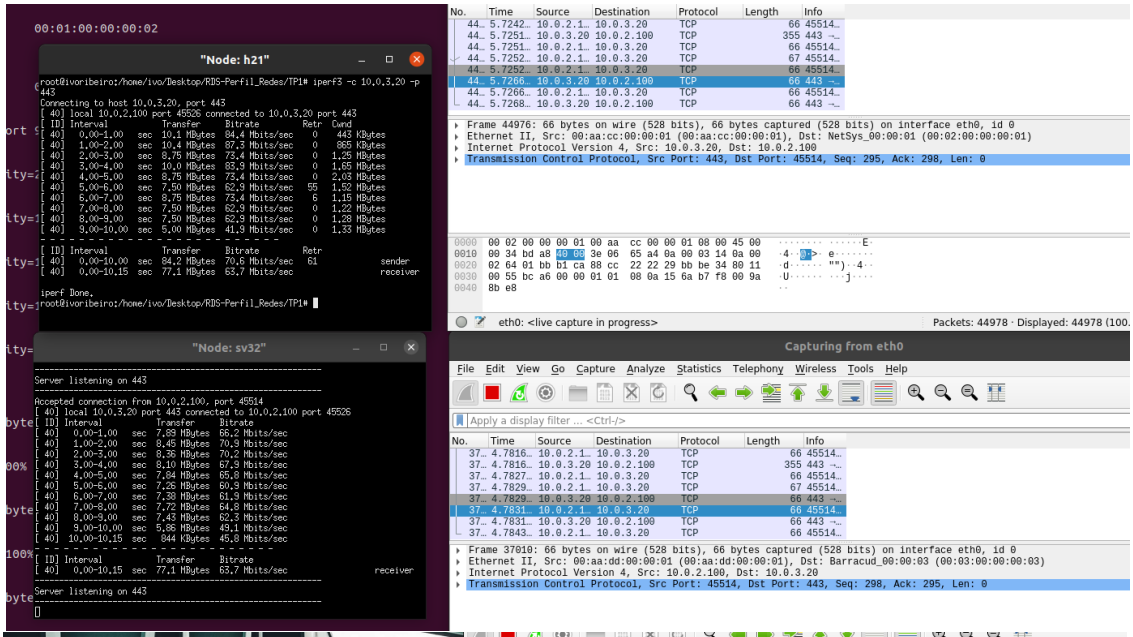

Figura 5: Comunicação *TCP* entre h21 e sv32 na porta 443

Figura 6: Teste sem comunicação *TCP* entre h21 e sv32 na porta 555

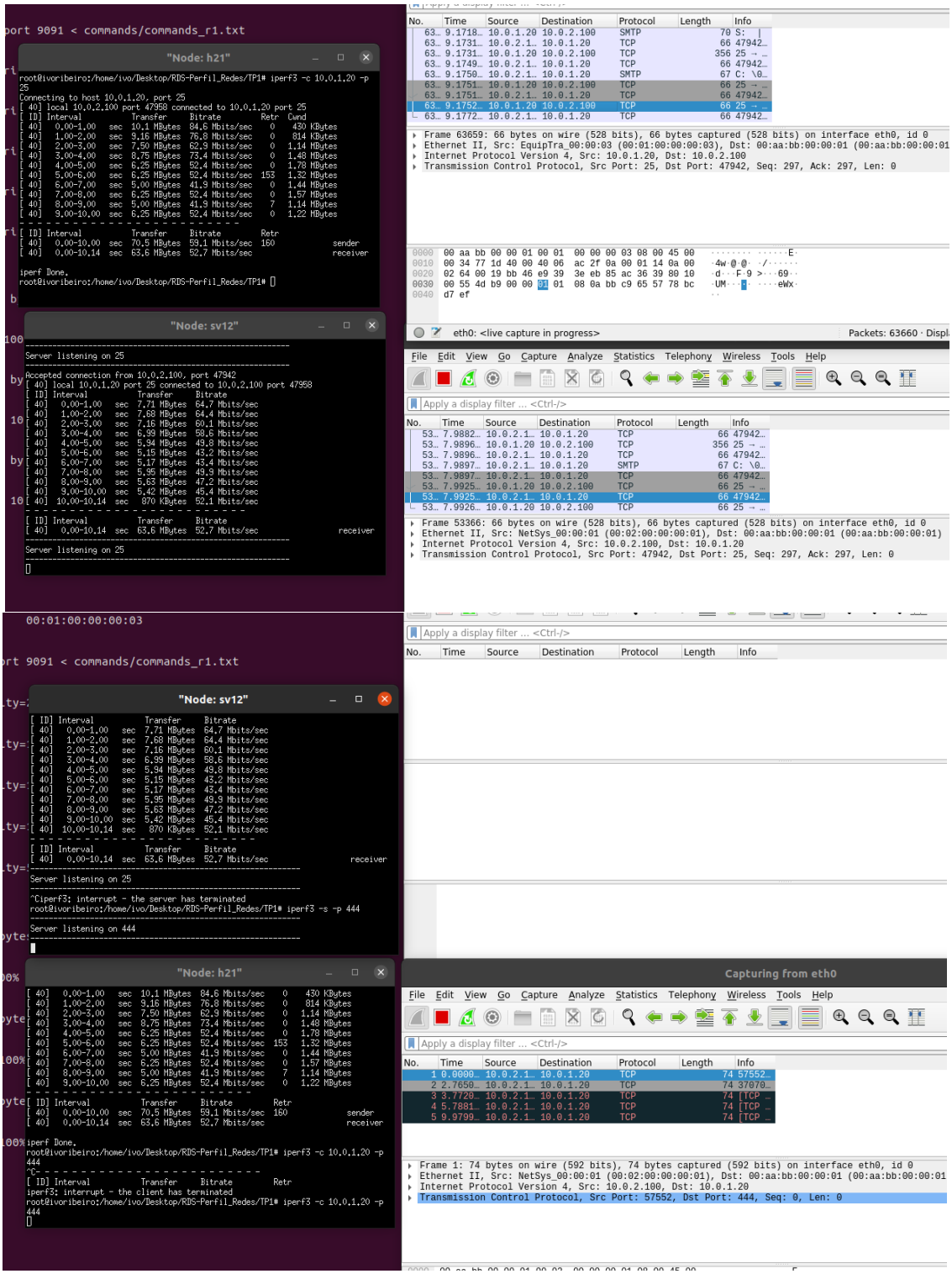

Figura 7: Comunicação *TCP* entre h21 e sv12 na porta 25

Figura 8: Teste de comunicação *TCP* entre h21 e sv12 na porta 444

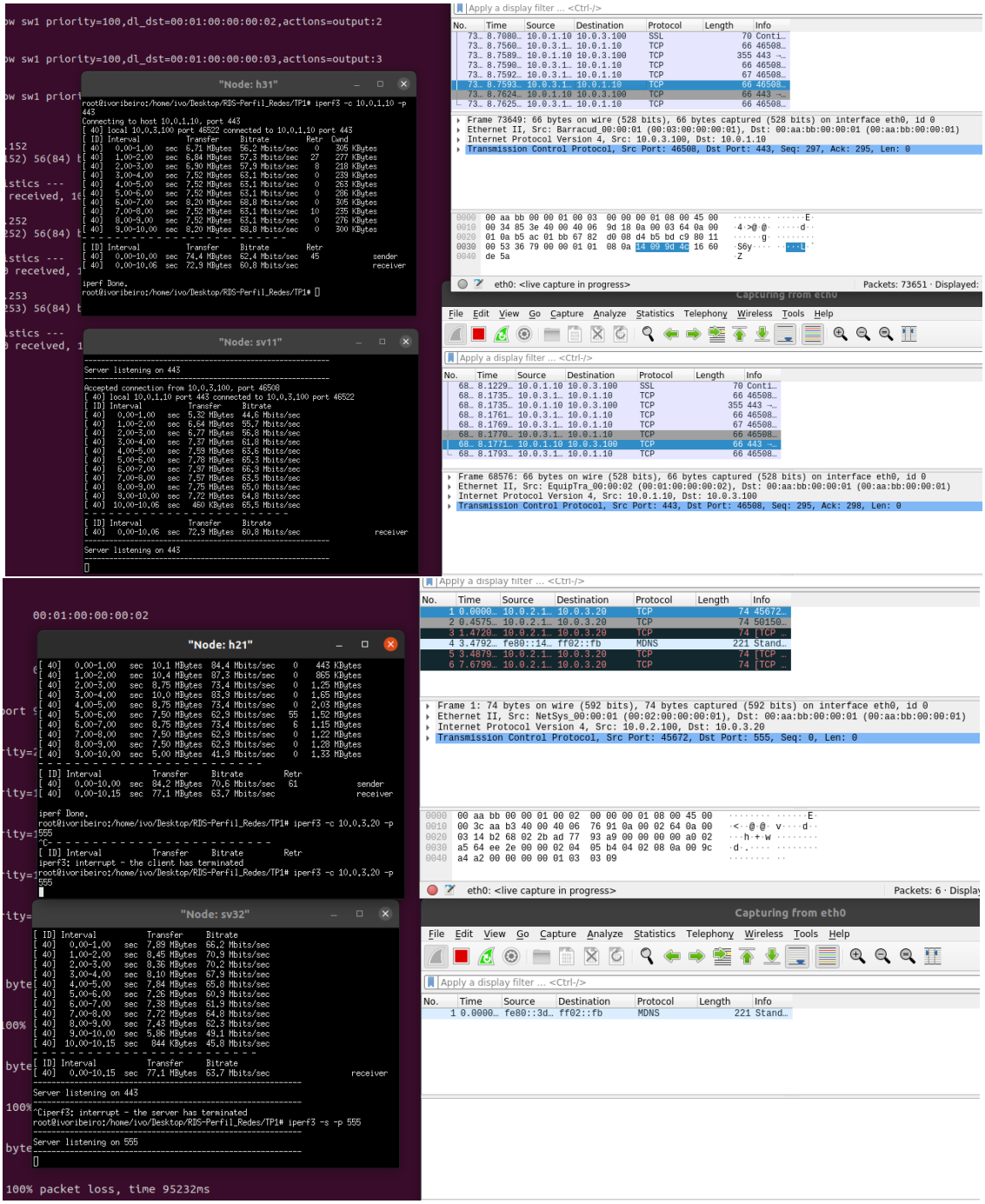

Figura 9: Comunicação *TCP* entre h21 e sv32 na porta 443

Figura 10: Teste de comunicação *TCP* entre h21 e sv32 na porta 55

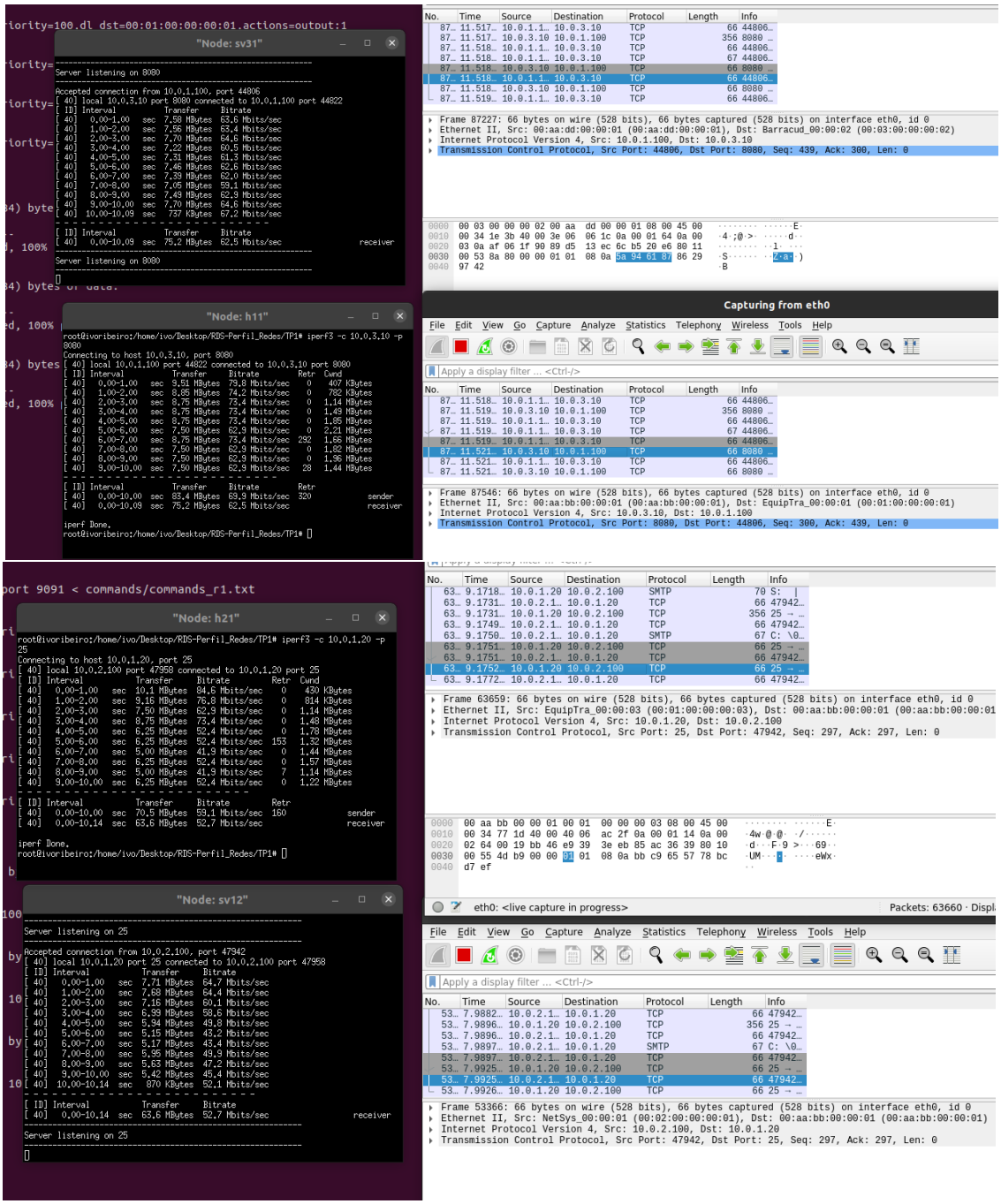

Figura 11: Comunicação *TCP* entre h11 e sv31 na porta 8080

Figura 12: Comunicação *TCP* entre h21 e sv12 na porta 25

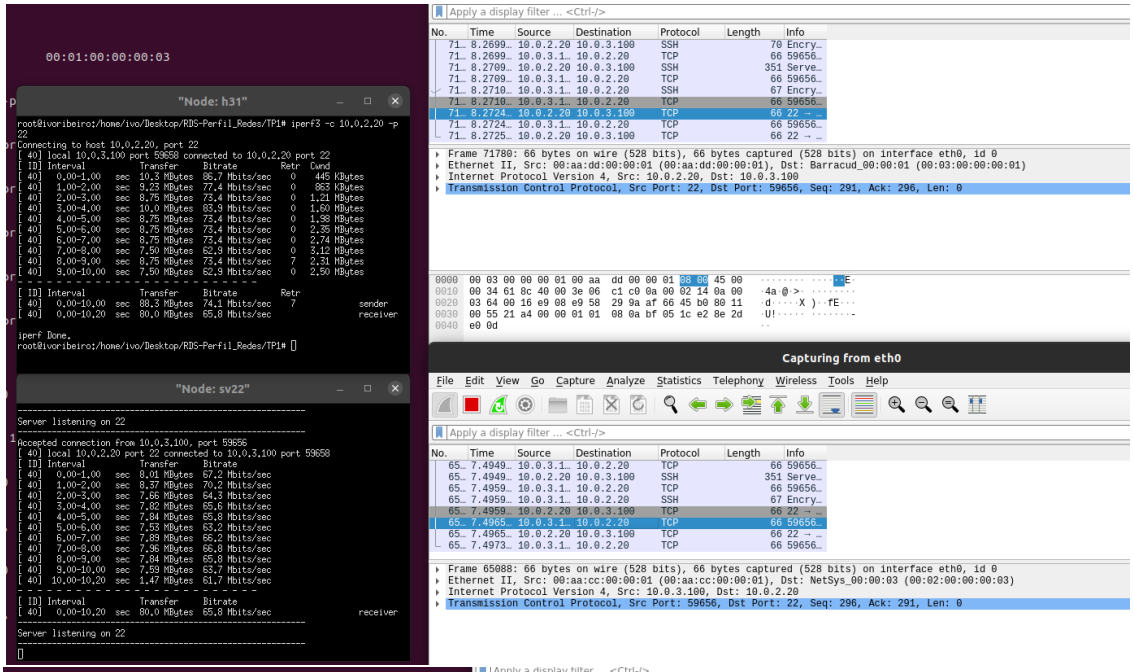

Figura 13: Comunicação *TCP* entre h31 e sv22 na porta 22

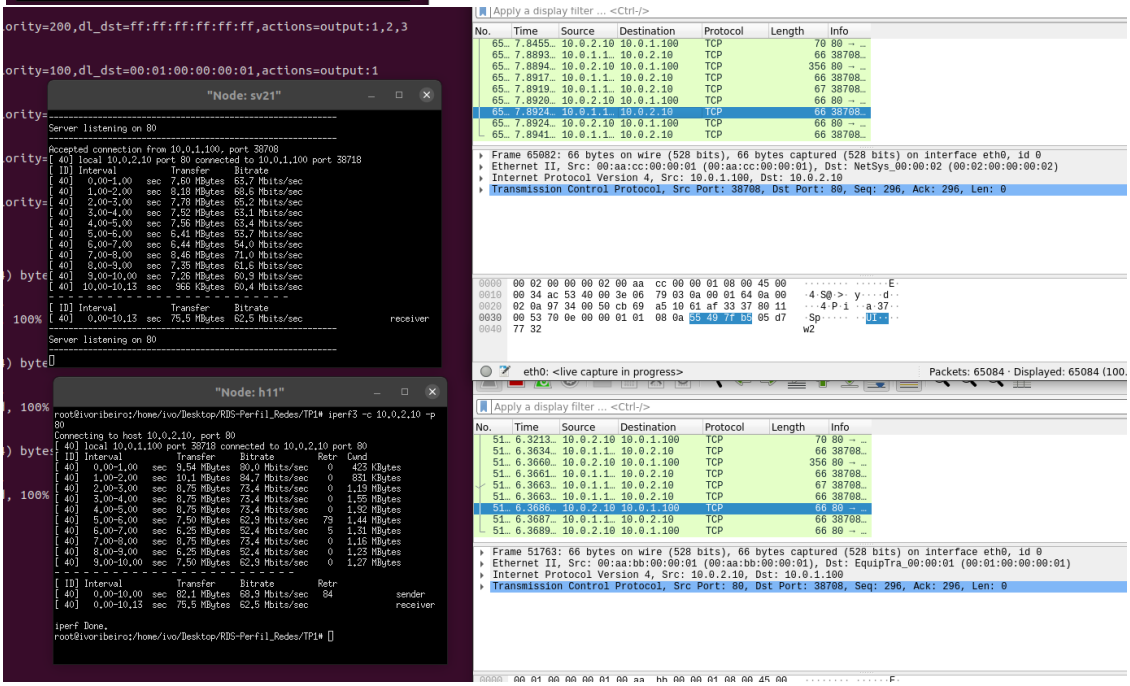

Figura 14: Comunicação *TCP* entre h11 e sv21 na porta 80

6 Conclusão

A implementação de um *firewall stateful* de Camada 3 (L3) e Camada 4 (L4) utilizando P4 para a rede da *TechSecure Inc.* apresentou desafios significativos, desde a configuração inicial da topologia de rede, até a aplicação de regras de *firewall* complexas. Este projeto exigiu um entendimento profundo dos conceitos de redes definidas por software, bem como conhecimentos avançados em programação P4.

A fase inicial de configuração da topologia em Mininet, incluindo a integração com dispositivos *BMv2 P4*, foi fundamental para estabelecer uma base sólida para a implementação do *firewall*.

A implementação das regras de *firewall* envolveu a criação de políticas específicas para controlar o acesso entre diferentes segmentos da rede em portas especificas o que garante que apenas o tráfego autorizado fosse permitido.

Os cenários de teste desenvolvidos permitiram validar a eficácia do *firewall*, demonstrando a aplicação correta dessas regras.

Em resumo, o desenvolvimento deste *firewall stateful* utilizando P4 revelou-se uma experiência enriquecedora e desafiante. O projeto reforçou o nosso conhecimento sobre redes definidas por software. Para além disso, as soluções implementadas atenderam aos requisitos de segurança descritos no enunciado, proporcionando uma camada adicional de proteção e controlo sobre a sua infraestrutura de rede.

Segunda Fase (TP2)

7 Introdução

A segunda fase do projeto tem como foco melhorar o *router P4* simples, para lidar com solicitações de eco *ICMP*, conhecidas como solicitações de *ping*. Ao modificar os *routers* através da introdução de capacidades básicas de tratamento de *ICMP*, desvendamos uma função essencial para diagnósticos e comunicação em redes.

Em ambientes de Redes Definidas por Software *(SDN)*, os planos de controlo e de dados são desagregados, permitindo uma gestão de rede mais flexível e programável.

O objetivo principal deste exercício é estender a funcionalidade do *router* simples em P4 para responder a solicitações de eco *ICMP* enviadas para qualquer uma das suas interfaces. Uma solicitação de eco *ICMP* solicita uma resposta do dispositivo de destino, facilitando a medição do tempo de ida e volta e a verificação da conectividade.

Esta fase envolve algumas etapas para alcançar a funcionalidade desejada, sendo o ponto crucial da implementação o reconhecimento de pacotes *ICMP*. Mais especificamente, as solicitações de eco *ICMP* que são identificadas pelo campo de *protocol* dentro dos parâmetros do pacote *IPv4*. Para além disso, serão feitas alterações no código P4 para intercetar solicitações de eco *ICMP*. Isso envolve adicionar regras para corresponder a pacotes *ICMP* e gerar respostas de eco apropriadas. Depois de identificados esses pacotes, temos de garantir uma resposta do dispositivo de destino, e, para tal, é necessário configurar uma ação que o faça. Por fim, o *router* modificado será testado usando o *Mininet*, enviando solicitações de eco *ICMP* para diferentes interfaces do *router* e verificando se as respostas corretas são retornadas, uma vez que, até esta fase, o resultado destas operações é o demonstrado nas imagens que se seguem.

```
mininet> pingall
*** Ping: testing ping reachability
h11 -> h21 h31 sv11 sv12 sv21 sv22 sv31 sv32
h21 -> h11 h31 sv11 sv12 sv21 sv22 sv31 sv32
h31 -> h11 h21 sv11 sv12 sv21 sv22 sv31 sv32
sv11 -> h11 h21 h31 sv12 sv21 sv22 sv31 sv32
sv12 -> h11 h21 h31 sv11 sv21 sv22 sv31 sv32
sv21 -> h11 h21 h31 sv11 sv12 sv22 sv31 sv32
sv22 -> h11 h21 h31 sv11 sv12 sv21 sv31 sv32
sv31 -> h11 h21 h31 sv11 sv12 sv21 sv22 sv32
sv32 -> h11 h21 h31 sv11 sv12 sv21 sv22 sv31
*** Results: 0% dropped (72/72 received)
```

Figura 15: Conectividade entre os dispositivos

```
mininet> h11 ping 10.0.2.252
PING 10.0.2.252 (10.0.2.252) 56(84) bytes of data.
^C
--- 10.0.2.252 ping statistics ---
43 packets transmitted, 0 received, 100% packet loss, time 42998ms

mininet> h11 ping 10.0.3.253
PING 10.0.3.253 (10.0.3.253) 56(84) bytes of data.
^C
--- 10.0.3.253 ping statistics ---
94 packets transmitted, 0 received, 100% packet loss, time 95232ms

mininet> h11 ping 10.0.1.251
PING 10.0.1.251 (10.0.1.251) 56(84) bytes of data.
^C
--- 10.0.1.251 ping statistics ---
56 packets transmitted, 0 received, 100% packet loss, time 56322ms
```

Figura 16: Falhas nas solicitações de eco *ICMP*

8 Implementação de Resposta a Tráfego ICMP

De modo a garantir a conectividade com todas as interfaces dos *routers*, até mesmo pela utilidade que isso revela no diagnóstico sobre o estado da rede, era necessária uma resposta por parte dos *routers* aos pacotes *ICMP* destinados às suas interfaces.

A tabela *icmp_flow* é utilizada para determinar a ação a ser executada em pacotes com base no endereço de destino do cabeçalho *IPv4* cujo tipo de protocolo seja *ICMP*. Assim, sempre que um pacote recebido segue o protocolo ICMP, é analisado por esta tabela, que irá pesquisar nas suas entradas se o endereço de destino *IPv4 (‘dstAddr‘)* pertence a uma das regras pré-definidas para o *router* em especifico, utilizando a correspondência de prefixo mais longo *(LPM, Longest Prefix Match)*.

Caso haja correspondência, é executada a função de *reply*, garantindo que o destino do pacote era uma interface do *router* que o recebeu e irá, então, responder e essa mesma solicitação de eco. Caso não haja correspondência, o pacote não sofre qualquer alteração, uma vez que é invocada a função *NoAction* do core P4, e é simplesmente propagado na rede conforme as regras de *forwarding* já existentes.

Esse comportamento é típico em respostas automáticas a mensagens de *ping (ICMP Echo Request)*, permitindo que o dispositivo envie uma resposta de *ping (ICMP Echo Reply)* de volta ao solicitante.

8.1 Descrição da função de resposta

Devido à impossibilidade dos hosts e servidores nesta topologia da criação de uma mensagem *icmp reply*, de forma a devolver uma resposta ao pacote inicial *icmp*, este mesmo pacote recebido terá de ser modificado e reencaminhado por forma a desempenhar a função de um *eco response*. Com isto, o objetivo da função de resposta principalmente da *action reply* presente na tabela *icmp_flow* consiste na alteração de alguns cabeçalhos do pacote *eco request* inicialmente enviado, designadamente o ip de origem, o ip de destino, o endereço mac de origem e destino, o campo *icmp_type* que inicialmente é *08*(tipo de pacote eco request) e passa a ser *00* (tipo de pacote eco reply).

A função *reply* é uma ação definida para tratar pacotes *ICMP (Internet Control Message Protocol)*, acionada quando o endereço de destino do pacote, segundo um protocolo *ICMP*, tem correspondência com uma regra descrita no *router* que o recebeu.

Assim, e de modo a responder a tal solicitação, a função *reply* necessita de fazer alterações ao pacote de modo a garantir uma resposta coerente, e para tal descreve o seguinte comportamento:

1. **(hdr.ipv4.dstAddr = hdr.ipv4.srcAddr)** - Atribui o endereço IP de origem *(srcAddr)* do cabeçalho *IPv4* ao endereço IP de destino *(dstAddr)*. Por outras palavras, troca o endereço IP de destino pelo endereço IP de origem.

2. **(hdr.ipv4.srcAddr = nxt_hop)** - Define o endereço IP de origem *(srcAddr)* do cabeçalho *IPv4* como o endereço IP passado como argumento *nxt_hop*.

3. **(hdr.ethernet.dstAddr = hdr.ethernet.srcAddr)** - Atribui o endereço MAC de origem *(srcAddr)* do cabeçalho Ethernet ao endereço MAC de destino *(dstAddr)*. Ou seja, troca o endereço MAC de destino pelo endereço MAC de origem.

4. **(hdr.ethernet.srcAddr = nt_mac)** - Define o endereço MAC de origem *(srcAddr)* do cabeçalho Ethernet como o endereço MAC passado como argumento *nt_mac*.

5. **(hdr.icmp.type = 0x00)** - Define o campo de tipo *ICMP* como *‘0x00‘*, que corresponde a uma mensagem de resposta de *ping (Echo Reply)*.

6. **(standard_metadata.egress_spec = standard_metadata.ingress_port)** - Define a porta de saída *(egress_spec)* como a mesma porta pela qual o pacote entrou *(ingress_port)*, essencialmente refletindo o pacote de volta ao remetente.

9 Cenários de Teste

```
mininet>
mininet> h11 ping 10.0.1.251
PING 10.0.1.251 (10.0.1.251) 56(84) bytes of data.
64 bytes from 10.0.1.251: icmp_seq=1 ttl=64 time=0.758 ms
64 bytes from 10.0.1.251: icmp_seq=2 ttl=64 time=1.20 ms
64 bytes from 10.0.1.251: icmp_seq=3 ttl=64 time=1.23 ms
64 bytes from 10.0.1.251: icmp_seq=4 ttl=64 time=1.08 ms
64 bytes from 10.0.1.251: icmp_seq=5 ttl=64 time=1.03 ms
64 bytes from 10.0.1.251: icmp_seq=6 ttl=64 time=1.00 ms
64 bytes from 10.0.1.251: icmp_seq=7 ttl=64 time=1.18 ms
^C
--- 10.0.1.251 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6035ms
rtt min/avg/max/mdev = 0.758/1.069/1.234/0.150 ms
mininet> h11 ping 10.0.3.251
PING 10.0.3.251 (10.0.3.251) 56(84) bytes of data.
64 bytes from 10.0.3.251: icmp_seq=1 ttl=62 time=2.12 ms
64 bytes from 10.0.3.251: icmp_seq=2 ttl=62 time=2.58 ms
64 bytes from 10.0.3.251: icmp_seq=3 ttl=62 time=2.65 ms
64 bytes from 10.0.3.251: icmp_seq=4 ttl=62 time=2.48 ms
64 bytes from 10.0.3.251: icmp_seq=5 ttl=62 time=2.78 ms
64 bytes from 10.0.3.251: icmp_seq=6 ttl=62 time=2.77 ms
^C
--- 10.0.3.251 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 2.121/2.564/2.782/0.224 ms
mininet> h11 ping 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=62 time=1.87 ms
64 bytes from 10.0.2.253: icmp_seq=2 ttl=62 time=3.30 ms
64 bytes from 10.0.2.253: icmp_seq=3 ttl=62 time=2.61 ms
64 bytes from 10.0.2.253: icmp_seq=4 ttl=62 time=2.10 ms
^C
--- 10.0.2.253 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.867/2.469/3.302/0.550 ms
mininet>
```

```
--- 10.0.2.252 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, tim
pipe 4
mininet> h31 ping 10.0.2.252
PING 10.0.2.252 (10.0.2.252) 56(84) bytes of data.
64 bytes from 10.0.2.252: icmp_seq=1 ttl=62 time=0.969 ms
64 bytes from 10.0.2.252: icmp_seq=2 ttl=62 time=2.28 ms
64 bytes from 10.0.2.252: icmp_seq=3 ttl=62 time=2.40 ms
64 bytes from 10.0.2.252: icmp_seq=4 ttl=62 time=2.32 ms
^C
--- 10.0.2.252 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.969/1.990/2.397/0.591 ms
mininet> h31 ping 10.0.1.253
PING 10.0.1.253 (10.0.1.253) 56(84) bytes of data.
64 bytes from 10.0.1.253: icmp_seq=1 ttl=62 time=2.46 ms
64 bytes from 10.0.1.253: icmp_seq=2 ttl=62 time=1.35 ms
64 bytes from 10.0.1.253: icmp_seq=3 ttl=62 time=2.47 ms
64 bytes from 10.0.1.253: icmp_seq=4 ttl=62 time=2.38 ms
64 bytes from 10.0.1.253: icmp_seq=5 ttl=62 time=2.59 ms
^C
--- 10.0.1.253 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 1.353/2.252/2.594/0.454 ms
mininet> h31 ping 10.0.3.251
PING 10.0.3.251 (10.0.3.251) 56(84) bytes of data.
64 bytes from 10.0.3.251: icmp_seq=1 ttl=64 time=0.775 ms
64 bytes from 10.0.3.251: icmp_seq=2 ttl=64 time=0.990 ms
64 bytes from 10.0.3.251: icmp_seq=3 ttl=64 time=0.893 ms
64 bytes from 10.0.3.251: icmp_seq=4 ttl=64 time=0.941 ms
^C
--- 10.0.3.251 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3030ms
rtt min/avg/max/mdev = 0.775/0.899/0.990/0.079 ms
mininet>
```

Figura 17: Conectividade *ICMP* com as interfaces dos diferentes *routers*

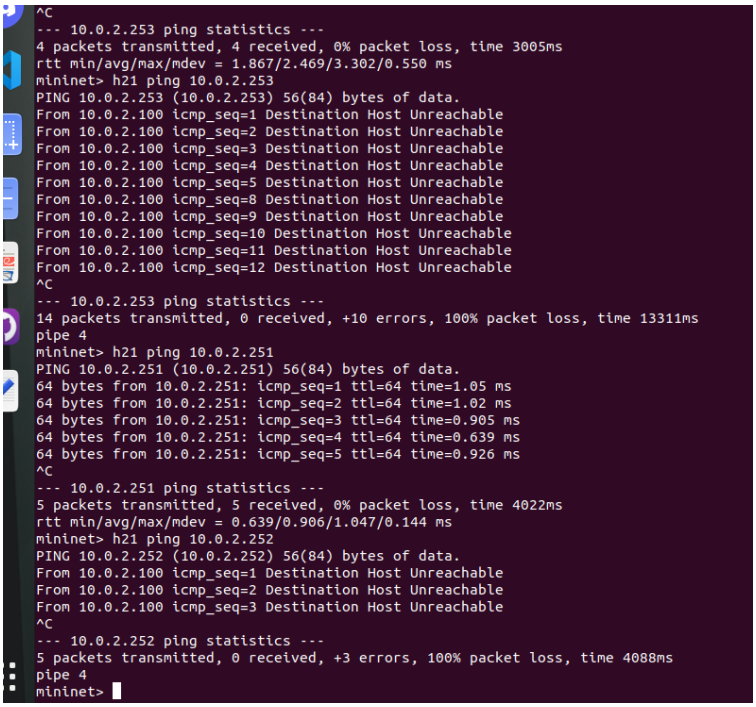

Figura 18: Falha na conectividade *ICMP* com as interfaces do *router* na própria *LAN*

Como podemos observar na figura 18, não é possível, no nosso sistema, garantir a conectividade com todas as interfaces do *router* presentes na própria *LAN*. As duas interfaces desse mesmo *router*, que não sejam diretamente ligadas ao *switch* presente na *LAN*, quando solicitadas com um eco *ICMP*, é devolvida uma notificação de **"Destination Host Unreachable"**.

Uma possível razão para este problema seria a de o *router* poder não ter uma rota para as interfaces inacessíveis ou essas interfaces não terem uma rota de retorno para a *LAN*. Se um dispositivo na *LAN* envia um pacote para uma interface do *router* que não está na mesma sub-rede, o *router* precisa de ter uma rota configurada para encaminhar esses pacotes corretamente. Porém, tentamos introduzir regras de *forwarding* a esses mesmos pacotes, de modo a retirá-los do *router* e fazer com que eles sejam retornados ao mesmo por uma outra interface, contudo sem sucesso, uma vez que, independentemente da estratégia, o resultado era o mesmo: **"Destination Host Unreachable"**.

10 Conclusão

A capacidade do *router* responder a solicitações de eco *ICMP* é uma funcionalidade essencial que aumenta a confiabilidade e a capacidade de diagnóstico da rede. Este projeto ilustra como o P4 pode ser utilizado para implementar características específicas de rede de maneira eficiente, permitindo uma maior flexibilidade e controlo sobre o tráfego de dados.

Com a conclusão deste exercício, estamos numa boa posição para enfrentar desafios mais complexos no campo das redes programáveis. Os conhecimentos aqui adquiridos servirão de base para futuras implementações, como a integração de protocolos adicionais, melhorias na segurança da rede e a otimização do desempenho do *router*. Este projeto também abre portas para explorar outros aspetos do P4, como o gerir dinamicamente as tabelas de encaminhamento e a integração com controladores *SDN*.

No caso, o exercício 2 desta fase, o de *Load Balancer with Network Address Translation (NAT)*, onde o objetivo é garantir o balanceamento do tráfego pela rede alcançando métricas de largura de banda e delay mais controlados.

Em suma, o sucesso deste projeto alcança o objetivo específico de lidar com solicitações de eco *ICMP*, mas, para além disso, solidifica os nossos conhecimentos e competências relativos a redes programáveis, o que sem dúvida nos prepara para contribuições mais avançadas e inovadoras no campo da engenharia de redes.

Terceira Fase (TP3)

11 Introdução

Neste projeto, o objetivo é desenvolver um controlador *P4Runtime*, avançando no conhecimento e nas habilidades adquiridas em exercícios anteriores relacionados ao P4 e ao *SDN*. O controlador *P4Runtime* permitirá a transição de uma injeção manual de regras no *simple_switch_CLI* para um controlo dinâmico e programável de dispositivos P4 utilizando o protocolo *P4Runtime*. Este controlador será capaz de gerir a topologia de rede mais complexa desenvolvida ao longo do semestre, que inclui múltiplas *LANs*, *switches Open vSwitch (OVS)* e *routers P4*.

O *P4Runtime* é um protocolo que facilita a comunicação entre um controlador *SDN* e dispositivos de rede programáveis, no nosso caso os *routers P4*, permitindo a configuração dinâmica de regras de encaminhamento, *firewall* e outras funcionalidades.

O desenvolvimento do projeto envolve várias etapas, começando por instanciar a topologia de rede mais complexa desenvolvida durante o semestre, incluindo múltiplas *LANs, switches OVS* e *routers simples P4*, neste novo ambiente. De seguida desenvolvemos um controlador *P4Runtime* capaz de se conectar a dispositivos P4 na topologia de rede, no qual implementamos funcionalidades para injeção dinâmica de regras de encaminhamento, *firewall* e *ICMP* nos dispositivos P4 utilizando o protocolo *P4Runtime*.

12 Configuração Inicial

Para esta terceira fase o ambiente de simulação mudou um pouco, uma vez que o *router p4* teria de ser capaz de receber as entradas das tabelas com regras de forma dinâmica através de um controlador *SDN*.

Num primeiro momento, e sempre com base no tutorial desta terceira fase, implementamos uma nova Fase 0 em relação ao projeto como um total, onde, desta vez, o objetivo seria incorporar a nossa topologia até aqui usada neste novo ambiente.

Como a inserção de regras de *forwarding* e reescrita de endereços *MAC* já estavam implementadas, assim como na Fase 0 da Etapa 1, apenas queríamos ver funcionar o tráfego de pacotes entre todos os dispositivos presentes na topologia funcional.

Nesta implementação inicial, decidimos usar métricas de implementação sobre comandos a serem realizados pelo controlador de uma forma não tão manual e fazer do *dummy_controller* não tão *dummy*. Para tal, colocamos o controlador a obedecer a regras inseridas em ficheiros *json*. Um dos ficheiros é chamado de *controller.json*, que serve para informar o controlador sobre quais os dispositivos P4 sobre o qual ele irá injetar regras, sendo assim por ele controlados, assim como as portas onde receberão os comandos e regras necessárias.

Existe ainda um outro ficheiro json, chamado *rules.json*, que permite indicar ao controlador as entradas nas tabelas que deve inserir em cada dispositivo, bem como os argumentos das mesmas.

Como resultado final desta fase, obtivemos uma topologia onde os dispositivos conseguem comunicar entre si, uma vez que, as regras de *forwarding* estão implementadas o que permite a troca de pacotes pelos mesmos.

13 Implementação das funções de escrita das novas tabelas

Dada ainda a pouca experiência com este novo ambiente de inserção de regras nas tabelas, foi desafiante a implementação das novas regras e tabelas por nós criadas nas fases anteriores.

Assim, podemos afirmar que existiram adversidades na implementação das entradas nestas tabelas, uma vez que, nem todo o formato era aceite pelas mesmas. Era necessário termos todos os fatores alinhados para que não existisse nenhum erro de sintaxe. Temos, por exemplo, o caso do *range* na nossa tabela de controlo de *firewall* que estava definido entre *0->65355* para delimitar as portas em tráfego *TCP* (todas as possibilidades para este tipo de comunicação) e este controlador não estava a saber lidar com o facto do *range* começar a 0 e daí a mudança deste valor para 1.

Todas as novas entradas serão invocadas consoante o conteúdo do ficheiro *json* com as regras para esse efeito, tornando o nosso controlador um pouco mais dinâmico e não tão manual.

Contudo, a adaptação foi rápida e dinâmica e os resultados foram os previstos, possibilitando assim um controlo sobre os dispositivos P4 dinâmico e adaptativo.

14 Cenários de Teste

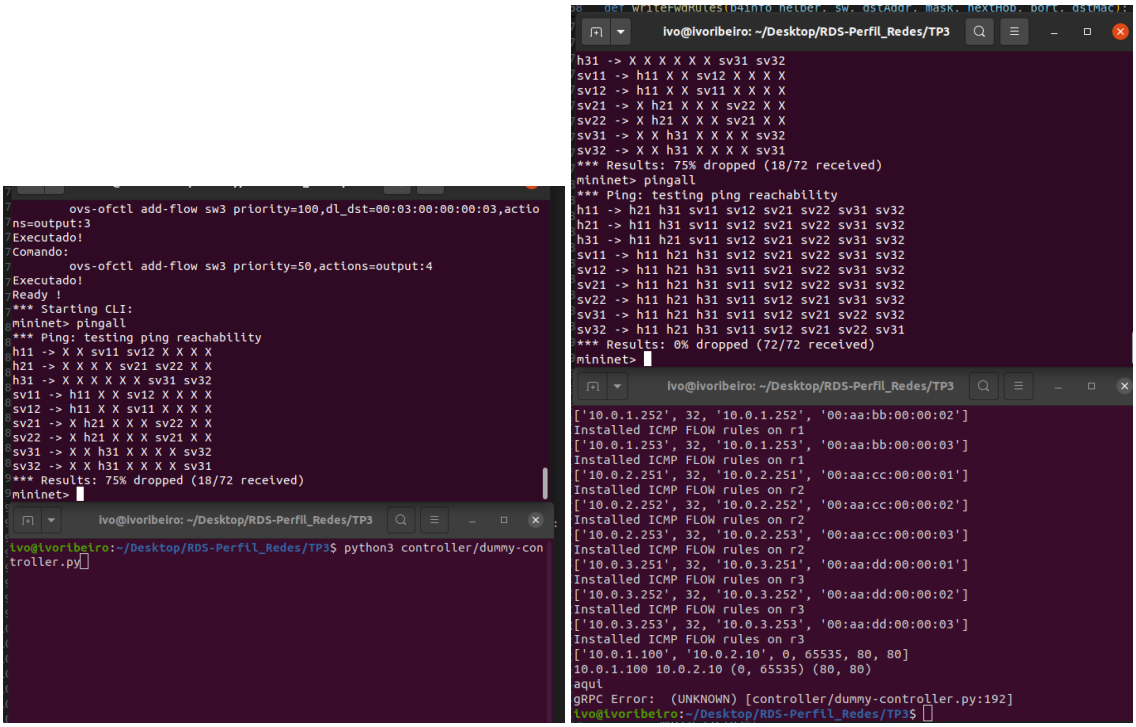

Figura 19: Conectividade entre os dispositivos com controller desligado / ligado

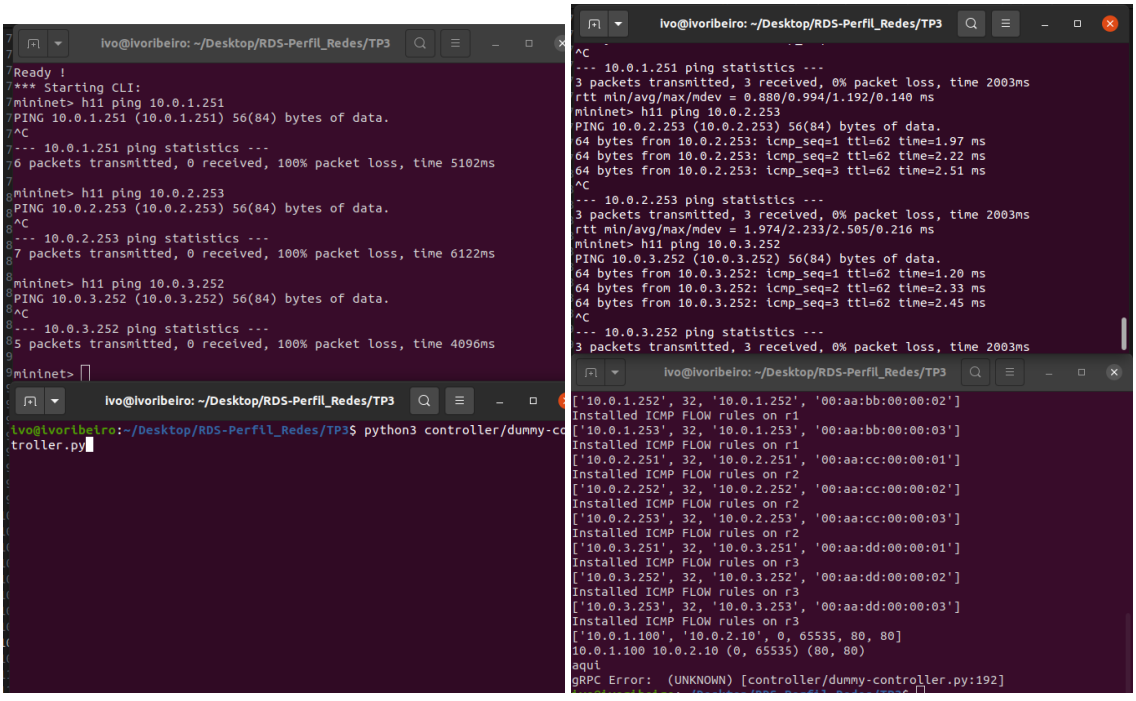

Figura 20: Solicitações de eco *ICMP* com controller desligado / ligado

Figura 21: Tráfego *TCP* com controller desligado / ligado

15 Conclusão

A conclusão deste projeto de desenvolvimento de um controlador *P4Runtime* marca um passo significativo na evolução das capacidades de redes definidas por software *(SDN)* e programação de dispositivos P4. Através da implementação e teste de um controlador dinâmico e programável, conseguimos demonstrar a transição bem-sucedida de uma configuração manual, para um sistema automatizado e adaptável de redes complexas.

Ao longo do projeto, enfrentamos vários desafios, como a complexidade na implementação do protocolo *P4Runtime* e a necessidade de garantir a interoperabilidade entre diferentes dispositivos de rede, uma vez que é necessário aprender uma dinâmica diferente de criar entradas nas tabelas e colocar os *routers* p4 a usar essas tabelas.

Este projeto demonstra a viabilidade e os benefícios do uso de controladores *P4Runtime* para a gestão de redes complexas. A capacidade de inserção dinâmica de regras e a resposta adaptativa a eventos de rede destacam a flexibilidade e o potencial das redes programáveis.

Em suma, o desenvolvimento e a implementação do controlador *P4Runtime* foram bem-sucedidos, cumprindo os objetivos propostos e proporcionando uma compreensão mais profunda das capacidades das redes definidas por software. Este projeto não apenas reforça o nosso conhecimento técnico em *P4* e *SDN*, como também abre novas possibilidades para inovações futuras no campo das redes programáveis.

16 Futuras melhorias

- Expansão de Funcionalidades: Integrar suporte para mais tipos de regras e políticas, aumentando a abrangência do controlador.
- Otimização do Desempenho: Implementar técnicas de otimização para melhorar o desempenho do controlador em redes de grande escala.
- Segurança: Fortalecer os mecanismos de segurança para proteger contra possíveis ataques e garantir a integridade das comunicações na rede.