



Catarina Pereira
Inês Neves
Leonardo Martins

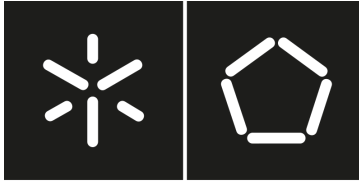
Data plane



Universidade do Minho
Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Inês Cabral Neves
Leonardo Dias Martins

Data plane



Universidade do Minho

Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Inês Cabral Neves
Leonardo Dias Martins

Data plane

Relatório de Gestão e Visualização de Redes
Módulo de Virtualização de Redes
Mestrado em Engenharia
Telecomunicações e Informática

Trabalho efetuado sob a orientação de:



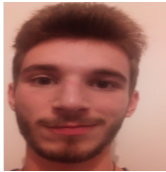
Professor Doutor António Luís Duarte Costa

Professor Doutor Bruno Alexandre Fernandes Dias

Professor Doutor João Fernandes Pereira

Identificação do Grupo

O Grupo 04 é composto pelos seguintes membros, todos pertencentes ao 1º ano do *Mestrado em Engenharia de Telecomunicações e Informática* (METI):

Imagem	Nome / Número Mecanográfico / E-mail institucional
	Catarina da Cunha Malheiro da Silva Pereira PG53733 pg537336@alunos.uminho.pt
	Inês Cabral Neves PG53864 pg53864@alunos.uminho.pt
	Leonardo Dias Martins PG53996 pg53996@alunos.uminho.pt

Índice

Identificação do Grupo	ii
Índice de Figuras	iv
Índice de Código	v
Lista de Acrónimos	vi
1 Introdução	1
2 Revisão da Literatura	2
3 Firewall	3
3.1 Implementação	3
3.2 Tabela de Entrada	5
3.3 Testes	6
4 Conclusão	12
Referências Bibliográficas	12
Bibliografia	13

Índice de Figuras

1	Execução de ficheiros.	6
2	h1 <i>ping</i> h2.	7
3	Wireshark.	7
4	Captura de mensagens.	8
5	Tráfego pela porta 5555, com h2 a servidor e h1 a cliente.	9
6	Wireshark.	9
7	Tráfego pela porta 5555, com h1 a servidor e h2 a cliente.	10
8	Tráfego pela porta 12345.	10
9	<i>Wireshark</i>	11

Índice de Código

header tcp_t function	3
state parse_ipv4 and state parse_tcp function	4
allow_tcp; table tcp_filter and apply function	5
MyDeparser function	5

Acrónimos

METI *Mestrado em Engenharia de Telecomunicações e Informática.*

P4 *Programming Protocolindependent Packet Processors .*

TCP *Transmission Control Protocol.*

UC *Unidade Curricular.*

1 Introdução

Este relatório faz parte da *Unidade Curricular* (UC) Gestão e Virtualização de Redes, do 1º semestre do 1º ano do Mestrado em Engenharia de Telecomunicações e Informática. Este projeto foi desenvolvido como resposta a um problema apresentado pela docente para o módulo de Virtualização de Redes.

No mundo em constante evolução das tecnologias de comunicação e virtualização de redes, a exigência por uma maior eficiência e flexibilidade no processamento de pacotes tem impulsionado o desenvolvimento de novas abordagens na programação *Data Plane*. Este trabalho tem como objetivo apresentar a programação *Data Plane* utilizando a linguagem *Programming Protocolindependent Packet Processors* (P4) projetada para dispositivos de rede, como *switches*, placas de interface de rede (NICs) e routers.

O P4 destaca-se como uma linguagem flexível e poderosa, permitindo a especificação de como dispositivos de plano de dados processam pacotes de maneira independente de protocolos. Ao fornecer uma abordagem programática para o processamento de pacotes, o P4 capacita os engenheiros de redes a personalizar e otimizar o comportamento de seus dispositivos para atender a requisitos específicos de aplicação.

A presente introdução oferece uma visão geral do contexto do projeto, destacando a importância da gestão de redes e da segurança no âmbito da tecnologia da informação. Além disso, esta fornece uma breve descrição dos principais objetivos do projeto, a sua estrutura e o conteúdo abordado no relatório.

O relatório seguirá com uma discussão detalhada sobre as estratégias adotadas, a apresentação de testes funcionais, bem como as conclusões obtidas durante a realização do projeto.

No âmbito deste exercício prático, foi proposta a implementação de um *firewall* utilizando a linguagem P4. O objetivo principal consistiu em modificar um programa P4 existente no repositório "tp3-simple-router.p4", nomeando-o como "tp3-firewall.p4". O desafio envolveu a criação de um mecanismo de filtragem de tráfego de rede, permitindo apenas a comunicação *Transmission Control Protocol* (TCP) entre dois *hosts* específicos (h1 e h2) em portas designadas.

2 Revisão da Literatura

Este capítulo apresenta os conceitos teóricos necessários ao desenvolvimento deste projeto, assim como a respetiva revisão da literatura.

Para o desenvolvimento do projeto foi importante aprofundar os conhecimentos a nível dos requisitos do sistema e *software*.

- *P4 (Programming Protocol-independent Packet Processors)*: P4 é uma linguagem de programação de alto nível projetada para controlar o processamento de pacotes em *switches* de rede e dispositivos similares. O objetivo principal é permitir que os programadores definam o processamento de pacotes, incluindo tabelas de encaminhamento e regras de filtragem. A principal vantagem é oferecer maior flexibilidade e controlo em comparação com abordagens tradicionais limitadas pelo *hardware*.
- *Mininet*: Mininet é uma ferramenta de emulação de rede que cria redes virtuais completas numa única máquina. Utilizado em pesquisa e desenvolvimento de redes, permite simular ambientes complexos para testar protocolos. Suporta diversas topologias e tecnologias, incluindo *OpenFlow*, facilitando testes sem a necessidade de *hardware* real.
- *Sistema Operacional Ubuntu*: Ubuntu é um sistema operacional baseado em Linux conhecido pela facilidade de uso e suporte robusto da comunidade. Possui uma interface amigável e suporta uma ampla gama de aplicações.

Através do tutorial fornecido, [1], que orienta o desenvolvimento de um *router* simples usando a linguagem P4, foram abrangidos vários aspectos, como:

- Configuração do Mininet e compreensão do *script* fornecido.
- Introdução à linguagem P4, incluindo as secções de análise, ingresso e *deparser*.
- Desenvolvimento de um *router* simples P4, explicando diferentes blocos como analisador, ingresso e *deparser*.
- Detalhes sobre como criar entradas de tabela e testar a configuração usando emulação.

3 Firewall

No âmbito do exercício proposto, o objetivo foi aplicar os conhecimentos adquiridos no tutorial mencionado no enunciado para desenvolver um mecanismo de filtragem de tráfego de rede. A tarefa envolveu a modificação do programa P4 presente no repositório fornecido, denominado “tp3-simple-router.p4”, com o intuito de criar um *firewall* capaz de bloquear todo o tráfego de rede, com exceção da comunicação TCP entre dois *hosts* específicos, h1 e h2, em portas designadas. O processo de implementação foi dividido em etapas bem definidas, desde a cópia do repositório original até a criação de tabelas e ações para filtrar o tráfego desejado. O cabeçalho TCP foi declarado com base no arquivo “tp3-base.p4”, e a lógica de análise e desanálise foi implementada para extrair informações relevantes e reconstruir o cabeçalho TCP para o tráfego permitido. A definição de critérios de filtragem, como permitir tráfego TCP de “h1” para “h2” na porta 555 e vice-versa, foi crucial para a configuração eficaz do *firewall*. A implementação das regras na tabela “tcp1_filter” e a aplicação destas permitiram controlar granularmente o tráfego, contribuindo para a segurança e eficiência da rede.

3.1 Implementação

O objetivo do exercício proposto, é aproveitar o conhecimento adquirido no tutorial descrito no enunciado para implementar um mecanismo de filtragem de tráfego de rede. A problema envolve a modificação do programa P4 presente no repositório fornecido “tp3-simple-router.p4” para criar um firewall que bloqueia todo o tráfego de rede, exceto a comunicação TCP entre dois *hosts*, h1 e h2, em portas específicas.

Para o desenvolvimento do projeto definiu-se alguns passos:

- 1. Cópia do Repositório:** Criar uma cópia do repositório existente “tp3-simple-router.p4” e nomeia-lo como “tp3-firewall.p4”.
- 2. Definição Cabeçalho TCP:** Declarar o cabeçalho TCP com base no ficheiro “tp3-base.p4”, de modo a garantir que os campos e atributos necessários estejam corretamente especificados.
- 3. Análise do Cabeçalho TCP:** Implementar a análise do cabeçalho TCP para extrair informações relevantes para processamento adicional.
- 4. Criação de Tabela e Ação:** Criar pelo menos uma tabela e uma ação associada. Essas estruturas serão responsáveis por definir os critérios de filtragem e as ações correspondentes a serem tomadas para o tráfego permitido.
- 5. Aplicação da Tabela:** Aplicar a tabela definida para processar pacotes de rede recebidos, permitindo apenas o tráfego TCP que atenda às condições especificadas.
- 6. Desanálise do Cabeçalho TCP:** Implementar a lógica de desanálise para reconstruir o cabeçalho TCP para o tráfego permitido antes de encaminhá-lo para o destino.

Os critérios de filtragem são:

- Permitir tráfego TCP de “h1” em qualquer porta para “h2” na porta 555
- Permitir tráfego TCP de “h2” na porta 5555 para “h1” em qualquer porta.

A declaração do cabeçalho TCP, foi com base no ficheiro *tp3-base.p4*, fornecido no tutorial, como apresentado no Código 3.1.

```
1 header tcp_t {
2   bit<16> srcPort;
3   bit<16> dstPort;
4   bit<32> seqNo;
5   bit<32> ackNo;
6   bit<4>  dataOffset; // how long the TCP header is
7   bit<3>  res;
8   bit<3>  ecn;        // Explicit congestion notification
```

3.1. IMPLEMENTAÇÃO

```
9   bit<6>  ctrl;          // URG,ACK,PSH,RST,SYN,FIN
10  bit<16> window;
11  bit<16> checksum;
12  bit<16> urgentPtr;
13 }
```

Código 3.1: header tcp_t function

O Código 3.2 extrai e analisa os cabeçalhos IPv4 e TCP num pacote de rede. Após extrair o cabeçalho IPv4, o programa verifica o tipo de protocolo. Se for TCP, ele extrai o cabeçalho TCP e transita para o estado de aceitação, indicando que a análise do pacote foi concluída. Caso o protocolo não seja TCP, o pacote é aceito sem análises adicionais.

```
1   state parse_ipv4{
2       packet.extract(hdr.ipv4); // extract function populates the ipv4 header
3       transition select(hdr.ipv4.protocol){
4           TYPE_TCP: parse_tcp;
5           default: accept;
6       }
7   }
8
9   state parse_tcp {
10      packet.extract(hdr.tcp);
11      transition accept;
12  }
```

Código 3.2: state parse_ipv4 and state parse_tcp function

O código 3.3 apresentado abaixo define uma tabela chamada *tcp_filter* e uma ação chamada *allow_tcp*. De seguida está uma explicação detalhada do que cada parte do código realiza:

- **action allow_tcp()** : É uma ação chamada *allow_tcp*, que é definida, mas não contém instruções específicas. Essa ação é utilizada para permitir o pacote. Noutras palavras, quando um pacote atende aos critérios especificados na tabela *tcp_filter*, a ação *allow_tcp* é acionada, indicando que o pacote deve ser aceito.
- **table tcp_filter** : A tabela *tcp_filter* é definida com a seguinte configuração:
 - Chave (Key):
 - * A chave da tabela é composta pelos endereços de origem e destino do IPv4, bem como pelos números de porta de origem e destino do TCP. Isso significa que a tabela será indexada com base nesses campos.
 - * Os endereços IP são correspondências exatas, enquanto os números de porta são correspondências por intervalo (*range*), permitindo considerar faixas de portas em vez de valores exatos.
 - Ações (Actions): Três ações são definidas: *allow_tcp*, *drop*, e *NoAction*
 - * *allow_tcp*: Aciona a ação permitindo o pacote.
 - * *drop*: Aciona a ação descartando o pacote.
 - * *NoAction*: Indica que nenhuma ação específica deve ser realizada, ou seja, o pacote continua o seu processamento sem nenhuma alteração.
 - Tamanho (Size): A tabela tem um tamanho máximo definido como 1024 entradas. O tamanho da tabela deve ser ajustado com base nos requisitos específicos da aplicação.
 - Ação Padrão (Default Action): A ação padrão é definida como *drop*, o que significa que, a menos que uma regra específica na tabela seja correspondida, o comportamento padrão será descartar o pacote.

3.2. TABELA DE ENTRADA

- **apply** : A secção *apply* especifica a aplicação das operações definidas em outras partes do código. Neste caso, são aplicadas as seguintes operações:
 - `ipv4_lpm.apply()`: Aplicação de operações relacionadas ao roteamento IPv4 com prefixo mais longo.
 - `src_mac.apply()`: Aplicação de operações relacionadas ao endereço MAC de origem.
 - `dst_mac.apply()`: Aplicação de operações relacionadas ao endereço MAC de destino.
 - `tcp_filter.apply()`: Aplicação da tabela `tcp_filter` para filtragem de tráfego TCP.

```
1  action allow_tcp() {
2      // This action can be empty, as it is just used to allow the packet.
3  }
4
5  table tcp_filter {
6  key = {
7      hdr.ipv4.srcAddr : exact;
8      hdr.ipv4.dstAddr : exact;
9      hdr.tcp.srcPort  : range;
10     hdr.tcp.dstPort  : range;
11 }
12 actions = {
13     allow_tcp;
14     drop;
15     NoAction;
16 }
17 size = 1024; // Set an appropriate size based on your requirements
18 default_action = drop;
19 }
20
21 apply {
22
23     ipv4_lpm.apply();
24     src_mac.apply();
25     dst_mac.apply();
26     tcp_filter.apply();
27 }
28 }
```

Código 3.3: `allow_tcp`; `table tcp_filter` and `apply` function

O Código 3.4 da função *MyDeparser* tem como objetivo preparar um pacote para transmissão física, convertendo a representação interna dos cabeçalhos do pacote numa forma serializada. Utilizando instruções `emit`, o código adiciona sequencialmente os cabeçalhos Ethernet, IPv4 e TCP ao pacote de saída.

```
1 control MyDeparser(packet_out packet, in headers hdr) {
2     apply {
3         packet.emit(hdr.ethernet);
4         packet.emit(hdr.ipv4);
5         packet.emit(hdr.tcp);
6     }
7 }
```

Código 3.4: *MyDeparser* function

3.2 Tabela de Entrada

No contexto da configuração P4, as duas últimas linhas do código são cruciais para definir as regras de filtragem de tráfego na tabela *tcp_filter*. Estas regras são fundamentais para permitir comunicação específica entre os *hosts* 10.0.1.1 e 10.0.2.1, aplicando restrições detalhadas às portas envolvidas.

A primeira linha estabelece uma regra que autoriza o tráfego TCP originado de 10.0.1.1 com destino a 10.0.2.1, onde a porta de destino é fixada em 5555, enquanto a porta de origem pode ser qualquer porta válida (1 a 65535). O valor associado à regra é definido como 1.

3.3. TESTES

A segunda linha complementa a configuração, permitindo o tráfego TCP no sentido oposto, de 10.0.2.1 para 10.0.1.1. Neste caso, a porta de origem é configurada como 5555, e novamente, a porta de destino pode ser qualquer porta válida. O valor associado a esta regra é 2.

Estas regras simétricas e específicas definem uma política de filtragem que habilita a comunicação bidirecional entre os *hosts* mencionados apenas quando ocorre em uma porta específica (5555). Esta abordagem oferece um controlo granular sobre o tráfego TCP entre *hosts*, contribuindo para a segurança e eficiência da rede.

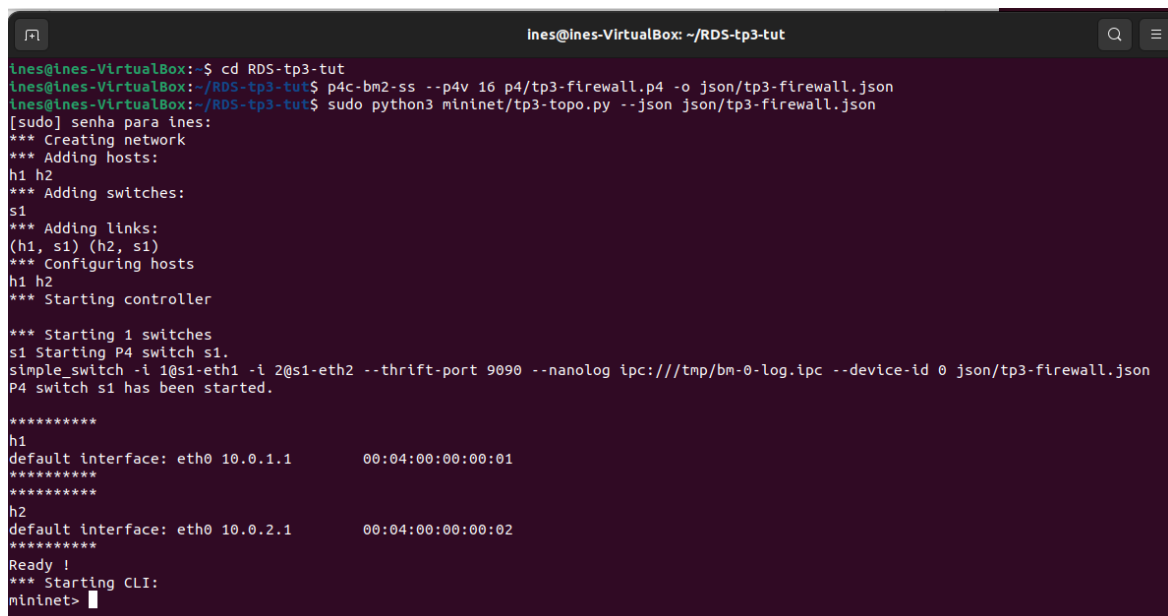
Cada valor associado a uma regra pode ser usado para identificar ações ou metadados específicos relacionados a essas regras durante o processamento posterior dos pacotes. Esta configuração exemplifica a flexibilidade e a capacidade de especificar políticas de filtragem personalizadas em ambientes de programação P4.

```
1 reset_state
2 table_set_default ipv4_lpm drop
3 table_set_default src_mac drop
4 table_set_default dst_mac drop
5 table_set_default tcp_filter drop
6 table_add ipv4_lpm ipv4_fwd 10.0.1.1/32 => 10.0.1.1 1
7 table_add ipv4_lpm ipv4_fwd 10.0.2.1/32 => 10.0.2.1 2
8 table_add src_mac rewrite_src_mac 1 => 00:aa:bb:00:00:01
9 table_add src_mac rewrite_src_mac 2 => 00:aa:bb:00:00:02
10 table_add dst_mac rewrite_dst_mac 10.0.1.1 => 00:04:00:00:00:01
11 table_add dst_mac rewrite_dst_mac 10.0.2.1 => 00:04:00:00:00:02
12 table_add tcp_filter allow_tcp 10.0.1.1 10.0.2.1 1>65535 5555>5555 => 1
13 table_add tcp_filter allow_tcp 10.0.2.1 10.0.1.1 5555>5555 1>65535 => 2
```

3.3 Testes

Este capítulo serve para demonstrar que o código funciona corretamente e que satisfaz todos os requisitos solicitados no enunciado.

A Figura 1 demonstra que os ficheiros `tp3-firewall.p4` e `commands.txt` compilam corretamente. Na figura também apresenta a compilação do ficheiro `p4` e do script do *mininet*.



```
ines@ines-VirtualBox: ~/RDS-tp3-tut
ines@ines-VirtualBox:~/RDS-tp3-tut$ p4c-bn2-ss --p4v 16 p4/tp3-firewall.p4 -o json/tp3-firewall.json
ines@ines-VirtualBox:~/RDS-tp3-tut$ sudo python3 mininet/tp3-topo.py --json json/tp3-firewall.json
[sudo] senha para ines:
*** Creating network
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1 Starting P4 switch s1.
simple_switch -i 1@s1-eth1 -i 2@s1-eth2 --thrift-port 9090 --nanolog ipc:///tmp/bn-0-log.ipc --device-id 0 json/tp3-firewall.json
P4 switch s1 has been started.

*****
h1
default interface: eth0 10.0.1.1          00:04:00:00:00:01
*****
h2
default interface: eth0 10.0.2.1          00:04:00:00:00:02
*****
Ready !
*** Starting CLI:
mininet>
```

Figura 1: Execução de ficheiros.

3.3. TESTES

Na Figura 2 e na Figura 3 pode-se observar que como é suposto ao realizar h1 *ping* h2 este bloqueia o ICMPv4. Ao observar através do *wireshark*, os pacotes mostrados estão usando o protocolo ICMP, especificamente mensagens de “echo request”, que são comumente usadas para testar a acessibilidade de um *host* na rede (comandos *ping*). As mensagens marcadas como “(no response found)” indicam que o pedido de “echo” enviado não recebeu uma resposta de “echo reply”, o que pode sugerir que o *host* de destino não está acessível ou não está configurado para responder a *pings*.

```
mininet> h1 ping h2
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
^C
--- 10.0.2.1 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6224ms
```

Figura 2: h1 ping h2.

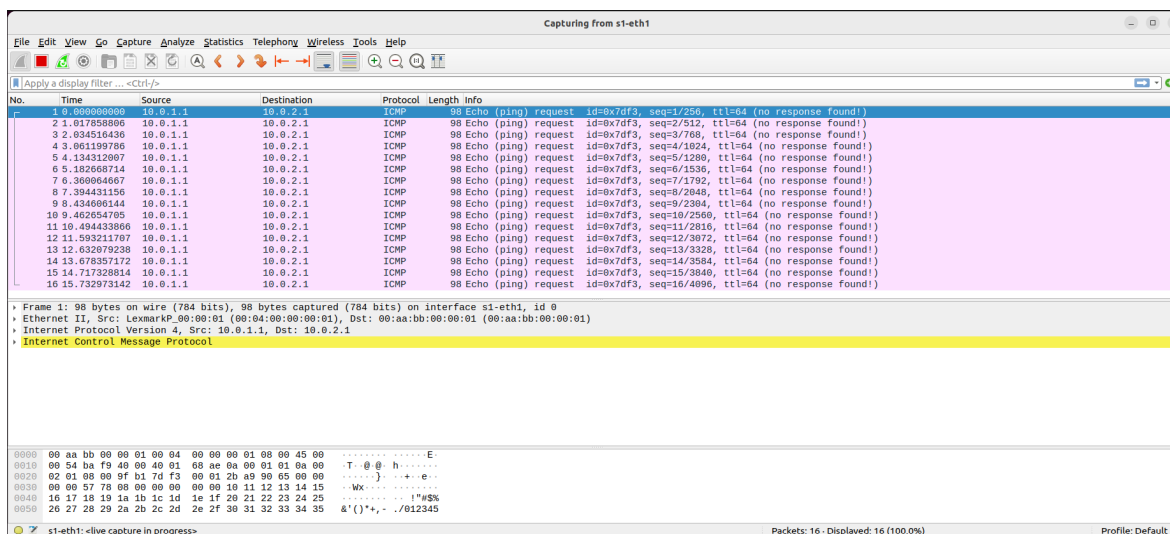


Figura 3: Wireshark.

Através dos comandos presentes pode-se observar na Figura 4 a captura das mensagens do Nano do *switch* de software (registros ou *logs*).

```
1 $ cd ~/RDStp3tut/tools/
2 $& sudo ./nanomsg_client.py thriftport 9090
```

3.3. TESTES

```
lnes@lnes-VirtualBox: ~/RDS-tp3-tut/tools
lnes@lnes-VirtualBox:~/RDS-tp3-tut/tools$ sudo ./nanomsg_client.py --thrift-port 9090
'--socket' not provided, using ipc:///tmp/bm-0-log.ipc (obtained from switch)
Obtaining JSON from switch...
Done
type: PACKET_IN, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, port_in: 1
type: PARSE_START, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, parser_id: 0 (parser)
type: PARSE_EXTRACT, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, header_id: 2 (ethernet)
type: PARSE_EXTRACT, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, header_id: 3 (ipv4)
type: PARSE_DONE, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, parser_id: 0 (parser)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, pipeline_id: 0 (ingress)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, table_id: 0 (MyIngress.ipv4_lpm)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, action_id: 0 (NoAction)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, table_id: 1 (MyIngress.src_mac)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, action_id: 3 (MyIngress.drop)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, table_id: 2 (MyIngress.dst_mac)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, action_id: 4 (MyIngress.drop)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, table_id: 3 (MyIngress.tcp_filter)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, action_id: 5 (MyIngress.drop)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 3674682871624542785, id: 74, copy_id: 0, pipeline_id: 0 (ingress)
type: PACKET_IN, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, port_in: 1
type: PARSE_START, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, parser_id: 0 (parser)
type: PARSE_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, header_id: 2 (ethernet)
type: PARSE_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, header_id: 3 (ipv4)
type: PARSE_DONE, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, parser_id: 0 (parser)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, pipeline_id: 0 (ingress)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, table_id: 0 (MyIngress.ipv4_lpm)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, action_id: 0 (NoAction)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, table_id: 1 (MyIngress.src_mac)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, action_id: 3 (MyIngress.drop)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, table_id: 2 (MyIngress.dst_mac)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, action_id: 4 (MyIngress.drop)
type: TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, table_id: 3 (MyIngress.tcp_filter)
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, action_id: 5 (MyIngress.drop)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 10757757541258855483, id: 75, copy_id: 0, pipeline_id: 0 (ingress)
```

Figura 4: Captura de mensagens.

Agora irá injetar-se as entradas na tabela, num novo terminal e executa-se os seguintes comandos apresentados

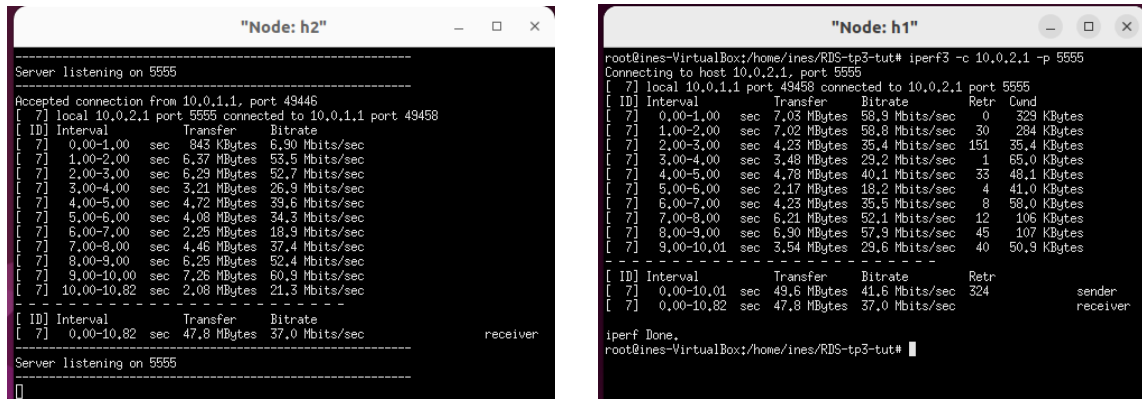
```
1 $ cd ~/RDStp3tut/commands/
2 $ simple_switch_CLI thriftport 9090 < commands.txt
```

A Figura 5(a) apresenta a *interface* da linha de comando onde foi executado o teste “iperf”. A figura indica que o servidor está a escutar na porta 5555 e exige intervalos de transferência de dados e medições de taxa de bits.

Na Figura 5(b) é semelhante à anterior, mas no ponto de vista de um *host* diferente, apresenta o comando “iperf” a ser executado com opções para conectar-se a um *host* no endereço IP 10.0.2.1 na porta 5555. Este é o lado do cliente onde se pode ver a transferência de dados do cliente para o servidor, juntamente com a taxa de bits, retransmissões (Retr) e tamanho da janela de congestionamento (Cwnd).

A Figura 5 apresenta a interface do *wireshark*, que mostra a captura de pacotes TCP na porta 5555. A tabela lista pacotes individuais com endereços IP de origem e destino, protocolos utilizados e informações sobre o tamanho do segmento TCP e o tamanho da janela. Através destas imagens consegue-se observar que o *firewall* está a permitir o tráfego conforme as regras especificadas.

3.3. TESTES



(a) Nodo h2.

(b) Nodo h1.

Figura 5: Tráfego pela porta 5555, com h2 a servidor e h1 a cliente.

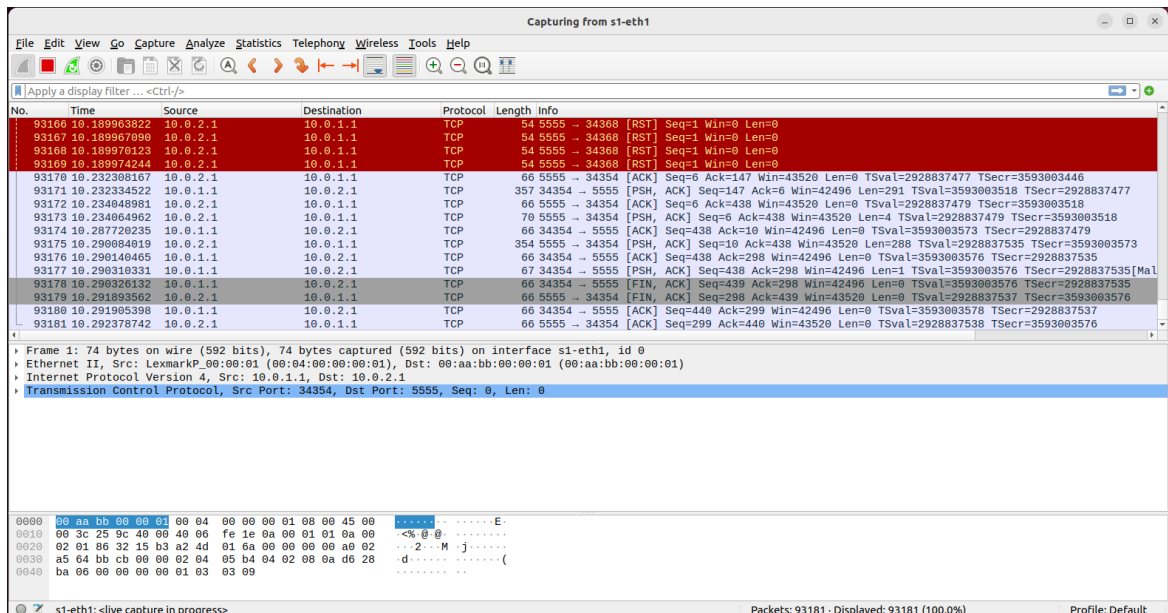


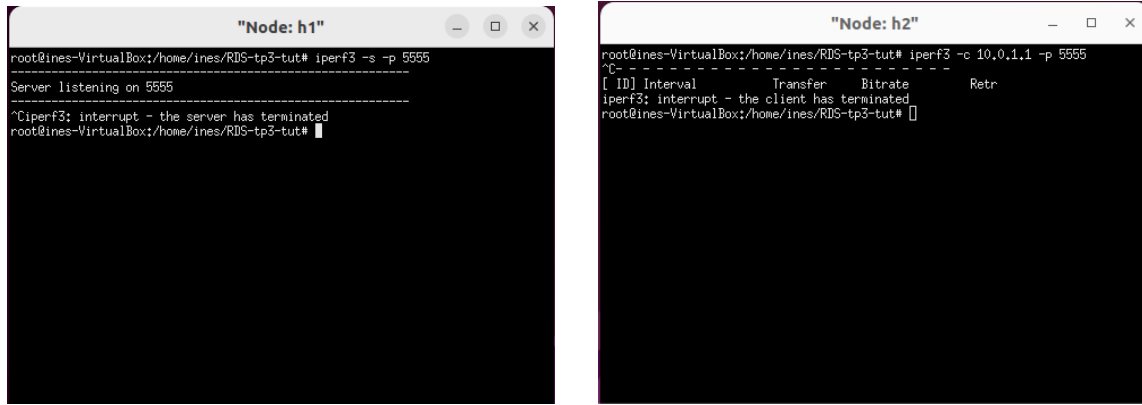
Figura 6: Wireshark.

A Figura 7 mostra a interface da linha de comando para os testes de conectividade usando o “iperf”. No contexto do exercício de *firewall* proposto, a interrupção dos testes sem transferência de dados indica que as regras do *firewall* foram aplicadas com sucesso, bloqueando todo tráfego exceto o tráfego TCP específico solicitado no exercício.

A Figura 7(a), mostra o terminal onde o servidor “iperf” a escutar na porta 5555. Este foi interrompido uma vez que o *host* h1 estava pronto para receber conexões na porta 5555, mas nenhuma transferência de dados ocorreu.

A Figura 7(b), apresenta o lado do cliente onde a tentativa de conexão ao *host* h1 na porta 5555 também foi interrompida. A falta de transferência de dados confirma que as regras do firewall estão a funcionar corretamente, pois h2 só consegue enviar tráfego pela porta 5555 e não o inverso.

3.3. TESTES



((a)) Nodo h1.

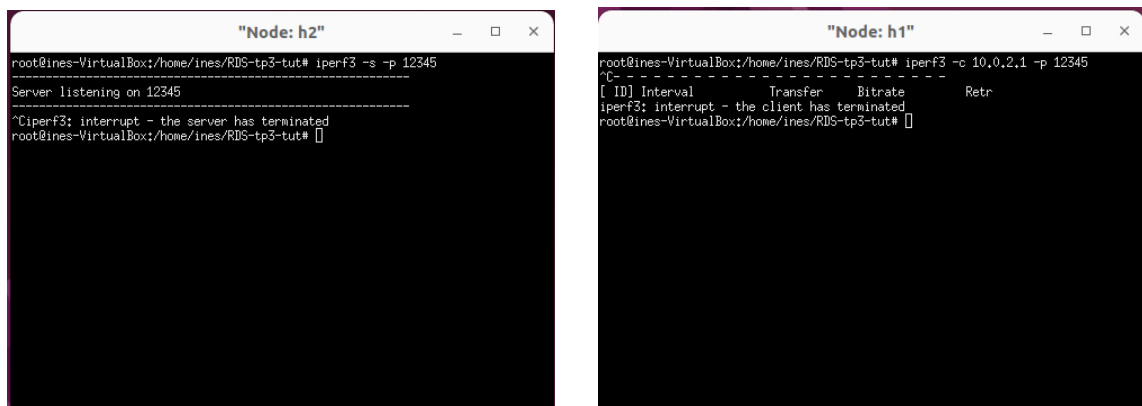
((b)) Nodo h2.

Figura 7: Tráfego pela porta 5555, com h1 a servidor e h2 a cliente.

A Figura 8(a) mostra o servidor com o comando “iperf” configurado para escutar na porta 12345. O servidor foi interrompido, uma vez que era suposto não ocorreu transferência de dados.

A Figura 8(b) mostra o cliente com o comando “iperf” a tentar-se conectar ao servidor no endereço IP 10.0.2.1 na porta 12345. O cliente foi interrompido, uma vez que este não conseguia iniciar a transferência de dados.

Na Figura 9 representa um print do *wireshark*, que mostra as tentativas de retransmissão de pacotes TCP do *host* h1 para o *host* h2 na porta 12345. Estas tentativas de conexão não foram concluídas devido às regras do firewall que estavam a bloquear o tráfego, uma vez que este bloqueia qualquer tráfego para h2 que não tenha como destino a porta 5555.



((a)) Nodo 2.

((b)) Nodo 1.

Figura 8: Tráfego pela porta 12345.

3.3. TESTES

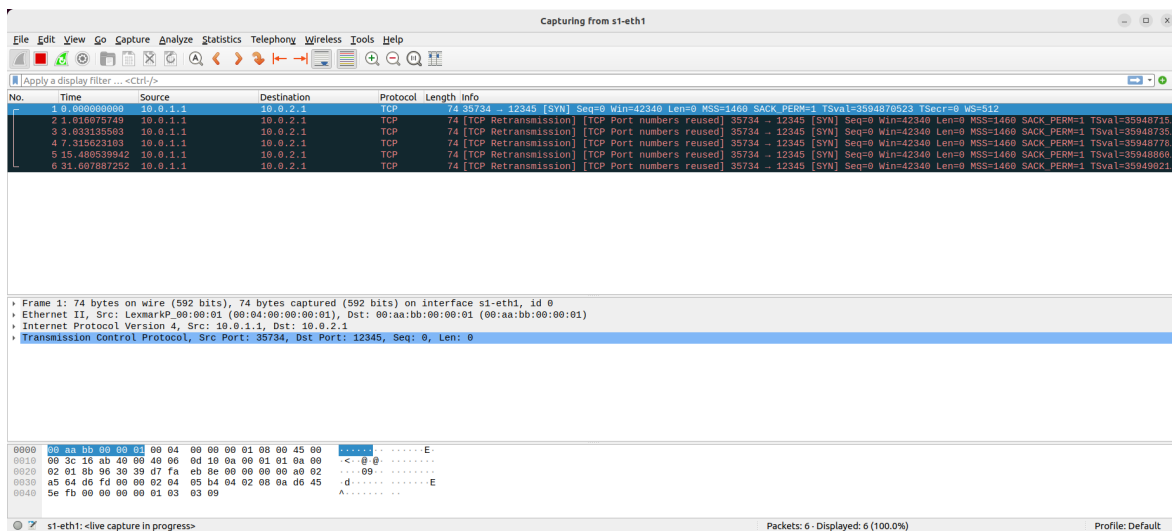


Figura 9: *Wireshark*.

4 Conclusão

A realização deste trabalho foi extremamente proveitosa para o desenvolvimento acadêmico do grupo. O projeto envolveu a configuração de um *router* simples e a implementação de regras de *firewall* para filtrar o tráfego TCP, permitindo a comunicação entre dois *hosts* em portas designadas.

Durante a execução do projeto, foi estabelecida uma topologia usando *Mininet*, e o comportamento esperado do plano de dados foi programado usando P4. O exercício proposto de criar regras de *firewall* que permitissem apenas o tráfego TCP em portas específicas foi elaborado com sucesso.

Os testes realizados com o comando “iperf” e com a monitorização do tráfego com o *Wireshark* confirmam que as regras de *firewall* funcionam como esperado. As regras permitem o tráfego entre o *host* h1 e o *host* h2 na porta 5555, enquanto todo o outro tráfego é bloqueado com sucesso. Este comportamento foi verificado tanto pela ausência de transferência de dados nas portas não autorizadas bem como com captura de pacotes que mostram retransmissões TCP.

Em suma, o projeto cumpriu todos objetivos de familiarização com as técnicas de programação de rede p4 e de compreensão prática de como os dispositivos de rede podem ser configurados e manipulados para atender a requisitos específicos.

Bibliografia

- [1] João Fernandes Pereira. *TP2 - Data plane Version: 1*. Blackboard. Acedido em 20 de dezembro de 2023. Nov. de 2023.