

Trabalho Prático Nº.3 - Data Plane

Rui Chaves (PG47637), Tiago Gomes (PG47696), Tânia Teixeira (A89613) e
Conceição Manuel (PG41016)

Mestrado em Engenharia Informática - Universidade do Minho
Redes Definidas por Software
18 de junho de 2022

Introdução

Este trabalho prático tem como principal objetivo a criação de uma *firewall*, programando no contexto das SDN, o *Data Plane*. Será desenvolvido através da linguagem de programação P4 e tendo por base o exemplo alargado disponibilizado pelos docentes.

A topologia consiste em 2 *hosts* interligados através de um *Data Plane Device*, que, neste exercício, corresponderá a uma *firewall*.

Com a *firewall* pretende-se bloquear todo o tráfego, permitindo apenas o tráfego *TCP* gerado nas seguintes operações:

- entre h1(qualquer porta) para a porta 5555 de h2
- entre a porta 5555 de h2 para qualquer porta de h1

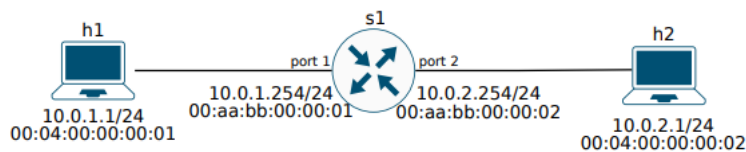


Figura 1. Topologia

Firewall

1 Implementação

1.1 Headers:

Para a implementação deste exercício é necessário em primeiro lugar declarar os *headers* necessários de modo a extrair posteriormente os campos necessários para a implementação da *firewall*, nomeadamente os cabeçalhos dos *PDU*s *Ethernet*, *IPV4* e *TCP*.

```
#definicao dos headers necessarios
```

```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}

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

header tcp_t {
    bit<16> srcPort;
    bit<16> dstPort;
    bit<32> seqNo;
    bit<32> ackNo;
    bit<4> dataOffset;
    bit<3> res;
    bit<3> ecn;
    bit<6> ctrl;
    bit<16> window;
    bit<16> checksum;
    bit<16> urgentPtr;
}
```

1.2 Parser:

O nosso programa começa pelo parser que lê o input do *packet_in* que corresponde a um objeto pré-definido declarado no *core.p4*. O parser P4 define uma máquina de estado com um estado inicial e 2 estados finais que correspondem a:

- *accept* que corresponde ao sucesso do *parsing*;
- *reject* que corresponde ao insucesso;

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata)
```

O output do parsing é escrito no argumento *headers* e as restantes estruturas correspondem a *structs* capazes de trocar informação entre os blocos.

No seguinte excerto de código podemos verificar se o *Ethernet type* é IPv4 e, em caso afirmativo, fazemos uma transição de estado invocando o *state parse ipv4*. De forma análoga, vamos verificar se o protocolo IPv4 é o *TCP* e fazer outra transição de estado para o *state_parse_tcp*

```
state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_IPV4: parse_ipv4;
        default: accept;
    }
}

state parse_ipv4 {
    packet.extract(hdr.ipv4); //extract function populates ipv4 header
    transition select(hdr.ipv4.protocol){
        TYPE_TCP: parse_tcp;
        default: accept;
    }
}

state parse_tcp {
    packet.extract(hdr.tcp); //extract function populates tcp header
    transition accept;
}
```

1.3 Ingress:

O bloco *ingress* é um bloco de controlo, no sentido em que se controla o *flow*, se manipula e transforma os *headers* já anteriormente *parsed*. Neste bloco são definidas as duas componentes de controlo: as ações e as tabelas. As ações manipulam os dados a serem processados e as tabelas definem as ações a implementar consoante as suas entradas. Neste bloco, seguimos os passos do docente detalhados no enunciado do projeto e construímos um router simples capaz de encontrar o próximo salto e mudar tanto o *MAC address* de origem como destino implementando as seguintes tabelas e ações:

```
action ipv4_fwd(ip4Addr_t nxt_hop, egressSpec_t port) {
    standard_metadata.egress_spec = port;
    meta.next_hop_ipv4 = nxt_hop;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}

table ipv4_lpm { //longest prefix matching
    key = { hdr.ipv4.dstAddr :lpm; }
    actions = {
        ipv4_fwd;
        drop;
        NoAction;
    }
    default_action = NoAction();
}

action rewrite_src_mac(macAddr_t src_mac) {
    hdr.ethernet.srcAddr = src_mac;
}

table src_mac {
    key = { standard_metadata.egress_spec :exact; }
    actions = {
        rewrite_src_mac;
        drop;
    }
    default_action = drop;
}

action rewrite_dst_mac(macAddr_t dst_mac) {
    hdr.ethernet.dstAddr = dst_mac;
}

table dst_mac {
    key = { meta.next_hop_ipv4 :exact; }
    actions = {
        rewrite_dst_mac;
        drop;
    }
    default_action = drop;
}
```

Para além destas tabelas e ações, implementamos também a tabela `check_tcp` que recebe uma *key* com 4 parâmetros necessários para a implementação da *firewall*. Nesta tabela, define-se *akey* que serve de pesquisa na tabela e os métodos da mesma. Procura-se uma entrada com um endereço *IP* exato, um endereço *IP* que tenha o maior prefixo comum (definindo uma prioridade), e dois valores entre um intervalo que tenham valores iguais aos dos *headers* em processamento. Caso se encontre uma entrada certa as ações `drop` e `NoAction` são as que se podem aplicar. A ação a aplicar é definida posteriormente por entrada da tabela.

A chave é constituída pelos endereços *IP* de origem e destino, assim com as suas portas de origem e destino.

```
table check_tcp {
    key = {
        hdr.ipv4.srcAddr: exact;
        hdr.ipv4.dstAddr: lpm;
        hdr.tcp.srcPort: range;
        hdr.tcp.dstPort: range;
    }
    actions = {
        NoAction;
        drop;
    }
    default_action = drop;
}
```

Todos os blocos de controlo necessitam de ter declarado um método `apply`, onde se define a ordem e pela qual o código deve ser aplicado.

Inicialmente é feita uma verificação do *header ipv4*, testando se este é válido, isto é, se o seu *parsing* foi efetuado com sucesso. Caso isto se verifique é possível definir a ordem as condições e a ordem pela qual deverão ser aplicadas

```
apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
        src_mac.apply();
        dst_mac.apply();
        check_tcp.apply();
    }
}
```

1.4 Deparser:

No bloco *Deparser*, apenas é necessário adicionar os headers recolhidos no *Parser*, isto é, o header *Ethernet*, *IPv4* e *TCP*, com as modificações das ações no *packet_out packet*.

```
control MyDeparser(packet_out packet, in headers hdr) {  
    apply {  
        packet.emit(hdr.ethernet);  
        packet.emit(hdr.ipv4);  
        packet.emit(hdr.tcp);  
    }  
}
```

2 Entrada das Tabelas

Os comandos adicionados aos já existentes no *template* do professor correspondem aos da nova tabela. Através do comando `table_set_default check_tcp drop` definimos a ação defeito da tabela, o drop. As últimas duas linhas em cima iniciam-se por `table_add` e correspondem a entradas na tabela.

Em ambas, definimos a ação a ser executada caso a linha dê *hit* com os dados recolhidos. O segundo *IP*, que corresponde ao endereço *IP* de destino apresenta a máscara definida /32 por não ser possível ter duas chaves exatas na pesquisa e por ambos os hosts, h1 e h2, serem sistemas terminais e querermos um *match* completo do *IP*. As entradas *range* podem ser interpretadas como uma procura por uma porta de *source* no header *TCP* com valor entre 0 e 65535, range de portas *TCP*, e porta de destino 5555 e vice-versa. Ambas as entradas podem ser lidas da seguinte forma:

```
reset_state  
table_set_default ipv4_lpm drop  
table_set_default src_mac drop  
table_set_default dst_mac drop  
table_set_default check_tcp drop  
table_add ipv4_lpm ipv4_fwd 10.0.1.1/32 =>10.0.1.1 1  
table_add ipv4_lpm ipv4_fwd 10.0.2.1/32 =>10.0.2.1 2  
table_add src_mac rewrite_src_mac 1 =>00:aa:bb:00:00:01  
table_add src_mac rewrite_src_mac 2 =>00:aa:bb:00:00:02  
table_add dst_mac rewrite_dst_mac 10.0.1.1 =>00:04:00:00:00:01  
table_add dst_mac rewrite_dst_mac 10.0.2.1 =>00:04:00:00:00:02  
table_add check_tcp NoAction 10.0.1.1 10.0.2.1/32 0->655355555->5555 =>0  
table_add check_tcp NoAction 10.0.2.1 10.0.1.1/32 5555->55550->65535 =>0
```

A injeção das entradas anteriores nas tabelas tem o seguinte output:

```
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: RuntimeCmd: Setting default action of ipv4_lpm
action: drop
runtime data:
RuntimeCmd: Setting default action of src_mac
action: drop
runtime data:
RuntimeCmd: Setting default action of dst_mac
action: drop
runtime data:
RuntimeCmd: Setting default action of check_tcp
action: drop
runtime data:
RuntimeCmd: Adding entry to lpm match table ipv4_lpm
match key: LPM-0a:00:01:01/32
action: ipv4_fwd
runtime data: 0a:00:01:01 00:01
Entry has been added with handle 0
RuntimeCmd: Adding entry to lpm match table ipv4_lpm
match key: LPM-0a:00:02:01/32
action: ipv4_fwd
runtime data: 0a:00:02:01 00:02
Entry has been added with handle 1
RuntimeCmd: Adding entry to exact match table src_mac
match key: EXACT-00:01
action: rewrite_src_mac
runtime data: 00:aa:bb:00:00:01
Entry has been added with handle 0
RuntimeCmd: Adding entry to exact match table src_mac
match key: EXACT-00:02
action: rewrite_src_mac
runtime data: 00:aa:bb:00:00:02
Entry has been added with handle 1
RuntimeCmd: Adding entry to exact match table dst_mac
match key: EXACT-0a:00:01:01
action: rewrite_dst_mac
runtime data: 00:04:00:00:00:01
Entry has been added with handle 0
RuntimeCmd: Adding entry to exact match table dst_mac
match key: EXACT-0a:00:02:01
action: rewrite_dst_mac
runtime data: 00:04:00:00:00:02
Entry has been added with handle 1
RuntimeCmd: Adding entry to range match table check_tcp
match key: EXACT-0a:00:01:01 LPM-0a:00:02:01/32 RANGE-00:00 -> ff:ff RANGE-15:b3 -> 15:b3
action: NoAction
runtime data:
Entry has been added with handle 0
RuntimeCmd: Adding entry to range match table check_tcp
match key: EXACT-0a:00:02:01 LPM-0a:00:01:01/32 RANGE-15:b3 -> 15:b3 RANGE-00:00 -> ff:ff
action: NoAction
runtime data:
Entry has been added with handle 1
RuntimeCmd:
```

Figura 2. Output da injeção das entradas - também em commands/commands1.log.txt

Através da imagem, podemos verificar que todas as entradas foram aceites e que os valores correspondem aos definidos nos comandos, alguns são apresentados em hexadecimal.

3 Requisitos e Dependências

- Sistema Operativo da família Ubuntu - Ubuntu, Xubuntu, Mint, Pop...
- Software & Packages:
 - Git
 - iperf
 - Mininet
 - p4c
 - behavioral model version 2 (bmv2) (w/ Thrift Server, nanomsg, nnpv)

4 Execução

É possível executar o projeto, após instalação de todas as dependências, através de um script `run.sh`. Este script deverá ser executado com permissões root e, idealmente, deverá ser terminado através da *hotkey* `CTRL` + `C` no terminal onde foi executado. O script abre três terminais `xterm` e corre os comandos necessários para a compilação do programa p4, execução da topologia mininet, captura de logs nanomsg e injeção nas tabelas. Os três terminais ficam abertos e através destes é possível fazer os testes de filtragem de tráfego.

Alternativamente, o processo seguinte pode ser realizado.

```
## compilar p4

p4c-bm2-ss --p4v 16 p4/tp3-firewall.p4 -o json/tp3-firewall.json

## correr o script mininet

sudo python3 mininet/tp3-topo.py --json json/tp3-firewall.json

## capturar as nano messages do software switch (outro terminal)

sudo tools/nanomsg_client.py --thrift-port 9090

## injetar novas entradas nas tabelas

simple_switch_CLI --thrift-port 9090 <commands/commands.txt >commands
                                     /commands_log.txt
```

5 Testes

Recorremos à ferramenta `iperf` para gerar o tráfego, especificando o cliente e servidor, o *IP* deste último e as portas de comunicação.

Recorremos também à utilização do *log client nanomsg*, cujo *output/logs* feito pelo *software switch* no processo de passagem de tráfego foi fulcral no processo de análise da nossa implementação permitindo verificar os diferentes processos a serem realizados e as ações respetivas em todos os pacotes pela ordem correspondente. Em geral, estávamos à espera de verificar os seguintes *logs* na seguinte ordem nos casos de tráfego permitidos e não permitido:

```
# Trafego Permitido
PACKET_IN
PARSER_START
PARSER_EXTRACT (ethernet)
PARSER_EXTRACT (ipv4)
PARSER_EXTRACT (tcp)
PARSER_DONE
PIPELINE_START (ingress)
CONDITION_EVAL (result: TRUE)
TABLE_HIT (MyIngress.ipv4_lpm)
ACTION_EXECUTE (MyIngress.ipv4_fwd)
TABLE_HIT (MyIngress.src_mac)
ACTION_EXECUTE (MyIngress.rewrite_src_mac)
TABLE_HIT (MyIngress.dst_mac)
ACTION_EXECUTE (MyIngress.rewrite_dst_mac)
TABLE_HIT (MyIngress.check_tcp)
ACTION_EXECUTE (NoAction)
PIPELINE_DONE (ingress)
PIPELINE_START (egress)
PIPELINE_DONE (egress)
DEPARSER_START
CHECKSUM_UPDATE
DEPARSER_EMIT (ethernet)
DEPARSER_EMIT (tcp)
DEPARSER_DONE
PACKET_OUT
```

```

# Trafego nao Permitido
PACKET_IN
PARSER_START
PARSER_EXTRACT (ethernet)
PARSER_EXTRACT (ipv4)
PARSER_EXTRACT (tcp)
PARSER_DONE
PIPELINE_START (ingress)
CONDITION_EVAL (result: TRUE)
TABLE_HIT (MyIngress.ipv4_lpm)
ACTION_EXECUTE (MyIngress.ipv4_fwd)
TABLE_HIT (MyIngress.src_mac)
ACTION_EXECUTE (MyIngress.rewrite_src_mac)
TABLE_HIT (MyIngress.dst_mac)
ACTION_EXECUTE (MyIngress.rewrite_dst_mac)
TABLE_MISS (MyIngress.check_tcp)
ACTION_EXECUTE (MyIngress.drop)
PIPELINE_DONE (ingress)

```

5.1 Teste TCP - Tráfego permitido pelo Firewall

Para este teste, recorreremos aos seguintes comandos:

```

h2: iperf -s -p 5555
h1: iperf -c 10.0.2.1 -p 5555

```

```

"Node: h1"
root@rui-vmt:/home/rui/Desktop/RDS-tp3-tut# iperf -c 10.0.2.1 -p 5555
Client connecting to 10.0.2.1, TCP port 5555
TCP window size: 178 KByte (default)
[ 5] local 10.0.1.1 port 36110 connected with 10.0.2.1 port 5555
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  39.0 MBytes 32.7 Mbits/sec
root@rui-vmt:/home/rui/Desktop/RDS-tp3-tut#

"Node: h2"
root@rui-vmt:/home/rui/Desktop/RDS-tp3-tut# iperf -s -p 5555
Server listening on TCP port 5555
TCP window size: 85.3 KByte (default)
[ 6] local 10.0.2.1 port 5555 connected with 10.0.1.1 port 36110
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.2 sec  39.0 MBytes 32.2 Mbits/sec

```

Figura 3. Output iperf TCP porta 5555

```

PARSER_START, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, parser_id: 0 (parser)
DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 10995010645737742719, id: 41961, copy_id: 0, header_id: 3 (ipv4)
DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 10995010645737742719, id: 41961, copy_id: 0, header_id: 4 (tcp)
DEPARSER_DONE, switch_id: 0, cxt_id: 0, sig: 10995010645737742719, id: 41961, copy_id: 0, deparser_id: 0 (deparser)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, header_id: 2 (ethernet)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, header_id: 3 (ipv4)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, header_id: 4 (tcp)
PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, parser_id: 0 (parser)
PACKET_OUT, switch_id: 0, cxt_id: 0, sig: 10995010645737742719, id: 41961, copy_id: 0, port_out: 2
PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, pipeline_id: 0 (ingress)
CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, condition_id: 0 (node 2), result: True
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, table_id: 0 (MyIngress.ipv4.lpm), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, action_id: 6 (MyIngress.ipv4.fwd)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, table_id: 1 (MyIngress.src_mac), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, action_id: 7 (MyIngress.rewrite_src_mac)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, table_id: 2 (MyIngress.dst_mac), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, action_id: 8 (MyIngress.rewrite_dst_mac)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, table_id: 3 (MyIngress.check_tcp), entry_hdl: 0
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, action_id: 1 (NoAction)
PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, pipeline_id: 0 (ingress)
PACKET_IN, switch_id: 0, cxt_id: 0, sig: 6149700294886760983, id: 41991, copy_id: 0, port_in: 2
PARSER_START, switch_id: 0, cxt_id: 0, sig: 10754305610636955939, id: 41963, copy_id: 0, parser_id: 0 (parser)
PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, pipeline_id: 1 (egress)
PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, pipeline_id: 1 (egress)
DEPARSER_START, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, deparser_id: 0 (deparser)
CHECKSUM_UPDATE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, cksum_id: 0 (cksum)
DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, header_id: 2 (ethernet)
DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, header_id: 3 (ipv4)
DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, header_id: 4 (tcp)
DEPARSER_DONE, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, deparser_id: 0 (deparser)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10754305610636955939, id: 41963, copy_id: 0, header_id: 2 (ethernet)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10754305610636955939, id: 41963, copy_id: 0, header_id: 3 (ipv4)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 10754305610636955939, id: 41963, copy_id: 0, header_id: 4 (tcp)
PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 10754305610636955939, id: 41963, copy_id: 0, parser_id: 0 (parser)
PACKET_OUT, switch_id: 0, cxt_id: 0, sig: 10258783296944121674, id: 41962, copy_id: 0, port_out: 2

```

Figura 4. Output nanomsg

Através da análise da figura 4, podemos ver que para o pacote com o ID 41962, os processos seguidos são os que correspondem a tráfego permitido e portanto, após TABLE_HIT, é aplicada a ação *NoAction* e o pacote é *Deparsed* e encaminhado pela porta 2 (port_out: 2).

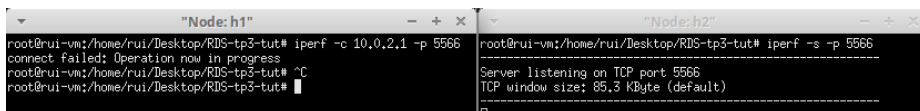
5.2 Teste TCP - Tráfego não permitido pelo Firewall

Para este teste, recorreremos aos seguintes comandos:

```

h2: iperf -s -p 5566
h1: iperf -c 10.0.2.1 -p 5566

```



```

Node:h1 Node:h2
root@rui-vni:/home/rui/Desktop/RDS-tp3-tut# iperf -c 10.0.2.1 -p 5566 root@rui-vni:/home/rui/Desktop/RDS-tp3-tut# iperf -s -p 5566
connect failed: Operation now in progress Server listening on TCP port 5566
root@rui-vni:/home/rui/Desktop/RDS-tp3-tut# ^C TCP window size: 85.3 KByte (default)
root@rui-vni:/home/rui/Desktop/RDS-tp3-tut#

```

Figura 5. output iperf TCP porta 5566

```

PACKET_IN, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, port_in: 1
PARSER_START, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, parser_id: 0 (parser)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, header_id: 2 (ethernet)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, header_id: 3 (ipv4)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, header_id: 4 (tcp)
PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, parser_id: 0 (parser)
PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, pipeline_id: 0 (ingress)
CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, condition_id: 0 (node_2), result: True
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, table_id: 0 (MyIngress.ipv4_lpm), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, action_id: 6 (MyIngress.ipv4_fwd)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, table_id: 1 (MyIngress.src_mac), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, action_id: 7 (MyIngress.rewrite_src_mac)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, table_id: 2 (MyIngress.dst_mac), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, action_id: 8 (MyIngress.rewrite_dst_mac)
TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, table_id: 3 (MyIngress.check_tcp)
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, action_id: 5 (MyIngress.drop)
PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 8977737841566891389, id: 42941, copy_id: 0, pipeline_id: 0 (ingress)
PACKET_IN, switch_id: 0, cxt_id: 0, sig: 11349073767351774733, id: 42942, copy_id: 0, port_in: 1

```

Figura 6. Output nanomsg

Através da análise da figura 6, podemos ver que para o pacote com o ID 42941, os processos seguidos são os que correspondem a tráfego não permitido e portanto, após *TABLE_MISS*, a ação aplicada é *drop* e a *Pipeline* é terminada sem se proceder ao encaminhamento do pacote.

5.3 Teste UDP - Tráfego não permitido pelo Firewall

Para este teste, recorreremos aos seguintes comandos:

```

h2: iperf -u -s -p 5555
h1: iperf -u -c 10.0.2.1 -p 5555

```

```

Node: h1
root@rui-vm:/home/rui/Desktop/RDS-tp3-tut# iperf -u -c 10.0.2.1 -p 5555
5
-----
Client connecting to 10.0.2.1, UDP port 5555
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 5] local 10.0.1.1 port 51616 connected with 10.0.2.1 port 5555
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 5] Sent 892 datagrams
root@rui-vm:/home/rui/Desktop/RDS-tp3-tut#

Node: h2
root@rui-vm:/home/rui/Desktop/RDS-tp3-tut# iperf -u -s -p 5555
-----
Server listening on UDP port 5555
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[]

```

Figura 7. Output iperf UDP porta 5555

```
PACKET_IN, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, port_in: 1
PARSER_START, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, parser_id: 0 (parser)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, header_id: 2 (ethernet)
PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, header_id: 3 (ipv4)
PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, parser_id: 0 (parser)
PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, pipeline_id: 0 (ingress)
CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, condition_id: 0 (node 2), result: True
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, table_id: 0 (MyIngress.ipv4_lpm), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, action_id: 6 (MyIngress.ipv4_fwd)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, table_id: 1 (MyIngress.src_mac), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, action_id: 7 (MyIngress.rewrite_src_mac)
TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, table_id: 2 (MyIngress.dst_mac), entry_hdl: 1
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, action_id: 8 (MyIngress.rewrite_dst_mac)
TABLE_MISS, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, table_id: 3 (MyIngress.check_tcp)
ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, action_id: 5 (MyIngress.drop)
PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 401262675903157235, id: 42902, copy_id: 0, pipeline_id: 0 (ingress)
PACKET_IN, switch_id: 0, cxt_id: 0, sig: 2125890401279861669, id: 42903, copy_id: 0, port_in: 1
```

Figura 8. Output nanomsg

Através da análise da figura 8, podemos ver que para o pacote com o ID 42902, os processos seguidos são os que correspondem a tráfego não permitido e portanto, após TABLE_MISS, a ação aplicada é *drop* e a *Pipeline* é terminada sem se proceder ao encaminhamento do pacote.

Conclusões

Com a realização do seguinte trabalho foi nos possível, ainda que de uma forma simples, interagir com a linguagem de programação de *Data Plane Devices*, P4. Perceber o mecanismo de implementação desta abstraindo os dispositivos de rede tradicionais condensando tudo numa máquina capaz de se tornar o que é pretendido.

Referências Bibliográficas

1. Open Maniak - iPerf
2. P4c P4 Compiler
3. P4 Language Tutorial
4. P416 Portable Switch Architecture (PSA)
5. Working with P4 in Mininet on BMV2
6. Behavioral Model (bmv2)
7. BMv2 Simple Switch
8. Runtime CLI (bmv2)
9. Github - Implementing A Basic Stateful Firewall
10. Github - Implementing A Basic Stateful Firewall
11. Match-Action Tables and P4 Runtime Control
12. Code Display Tool