Universidade do Minho
Escola de Engenharia

Master's in Telecommunication and Informatics Engineering

Network Management and Virtualization

Data Plane - Project Report

Camila Pinto - PG53712

Barbara Fonseca - PG53677

Eduarda Dinis- PG53793

Contents

1 Introduction

This report is being developed as part of the Network Management and Virtualization course, in the 1st semester of the 1st year of the Master's degree in Telecommunications and Informatics Engineering.

The goal of the project is to introduce Software Defined Networks(SDN's), which are divided in data and control planes.However,for this project, we will focus especifically on learning data plane programming with Programming Protocol-independent Packet Processors (P4), which is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packet.

To gain proficiency in using these tools, we set up the essential software for our development environment. This included the following components: BMv2, or Behavioral Model version 2, which is the second iteration of the reference P4 software switch functioning as a platform for executing P4 programs; p4c, a reference compiler designed for the P4 programming language; and mininet, an open-source network emulation tool enabling the creation of virtual networks on a single computer.

Initially, we followed the proposed tutorial to develop a simple router. Through this process, we gained an understanding of the V1 Model architecture, learned about the different blocks, how they function, and acquired programming skills in the P4 language.Next, we embarked on developing the proposed exercise:firewall. The firewall's objective is to restrict all traffic except TCP traffic flowing from any port to h2 port 5555 and from h2 port 5555 to h1 on any port. To achieve this, we follow the instructions: declare the TCP header, parse the TCP header, create a minimum of one table and one action, apply the table, and finally, deparse the TCP header.

2 Firewall

2.1 Implementation

To implement the proposed exercise, we need to develop a stateful firewall. A stateful firewall is a type of firewall that monitors the connection state and makes decisions based on the context of the ongoing communication. Considering this, there are two ways to configure a stateful firewall: reject everything except for cases a, b, c...; or allow everything except for cases a, b, c... .

For this scenario, we have chosen to reject traffic except for the following TCP traffic:

- From h1 on any port to h2 on port 5555
- From h2 on port 5555 to h1 on any port.

Additionally, it is also necessary to block UDP traffic from h1 to port 5555 on h2 and from h2 port 5555 to h1, this way only TCP traffic passes. There are various ways to achieve this, and we opted to create a table that only has the "drop" action to filter UDP traffic.

For this, we programmed the following components:

2.1.1 Parser

For the project, we require the following headers: Ethernet, IPv4, TCP, and UDP. These are the headers that we intend to parse. The parsing process operates by extracting headers from the received packet. Subsequently, it determines the type of payload that follows. For instance, if the IPv4 header is extracted, the system checks which type of payload follows: TCP or UDP. Based on the identified payload type, the parsing transitions to the respective parse for that specific payload.

2.1.2 Ingress

The Ingress block, or top pipeline, is a control block used to control the flow, manipulate, and transform headers. In this block, during the router tutorial, we had already defined three tables: ipv4_lmp, src_mac, dst_mac. In the first table, the desired action is to find the next hop, i.e., forward the packet; in the second table, the action is to rewrite the Ethernet source MAC address; finally, in the third table, the action is to change the destination MAC address.

For the firewall, we created a table with four search keys: IPsrc, IPdst, Portsrc, Portdst, and two actions: NoAction, which occurs if there is a corresponding entry (allowed traffic), and drop, which occurs if the traffic is not allowed. To determine the type of keys, we decided to set the ports as range type, considering that they can be any value, based on reference [6] from the statement.

We also created a table to filter UDP traffic, which takes four search keys:IPsrc, IPdst, Portsrc, Portdst and the only action is to drop, meaning to block the traffic.

Every block defined by the keyword 'control' needs to implement the 'apply' statement. In the apply statement, we define the conditions and order in which our code should be applied. The logic for our apply statement is as follows: if the packet is IPv4 and has a valid UDP protocol, it applies a filter for UDP. If it is IPv4 with a valid TCP protocol, it applies a series of actions, including IPv4 forwarding, modifying source and destination MAC addresses, and applying filter for TCP.

2.1.3 Deparser

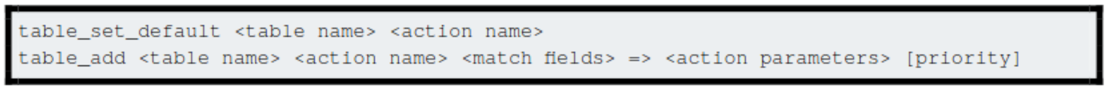
In the deparser block we simply add the headers(Ethernet,IPv4,TCP e UDP), extrated in the parser, to the packet_out packet.

2.2 Table Entries

In this section we will explain how we created the table entries for the following tables:

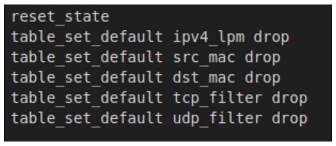
- tcp_filter;
- udp_filter.

Entry structure should be:



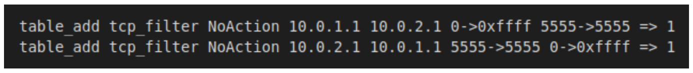
```
table_set_default <table name> <action name>
table_add <table name> <action name> <match fields> => <action parameters> [priority]
```

First we defined the default behaviour for each table:



```
reset_state
table_set_default ipv4_lpm drop
table_set_default src_mac drop
table_set_default dst_mac drop
table_set_default tcp_filter drop
table_set_default udp_filter drop
```

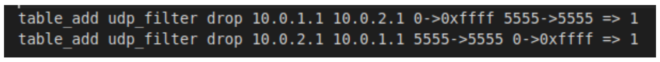
Next for tcp_filter we added two entry tables (for each case:h1 any port to h2 port 5555 and h2 port 5555 to h1 any port):



```
table_add tcp_filter NoAction 10.0.1.1 10.0.2.1 0->0xffff 5555->5555 => 1
table_add tcp_filter NoAction 10.0.2.1 10.0.1.1 5555->5555 0->0xffff => 1
```

The first entry means: when table tcp_filter matches 10.0.1.1 (IPsrc-h1), 10.0.2.1 (IPdst-h2) 0->0xffff(range for any port with 16 bits) 5555->5555 (range for specific port 5555) execute action *NoAction* with priority 1.The same criteria and priority are applied for the alternate case.

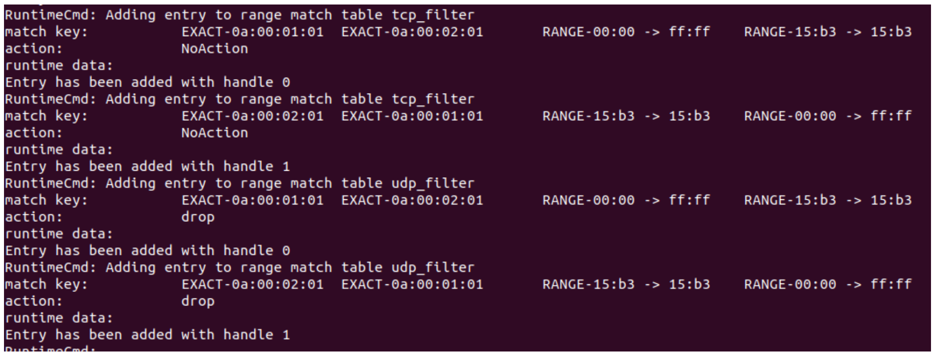
Finally, for udp_filter we added two entry tables with the intention to block UDP traffic.



```
table_add udp_filter drop 10.0.1.1 10.0.2.1 0->0xffff 5555->5555 => 1
table_add udp_filter drop 10.0.2.1 10.0.1.1 5555->5555 0->0xffff => 1
```

2.3 Tests

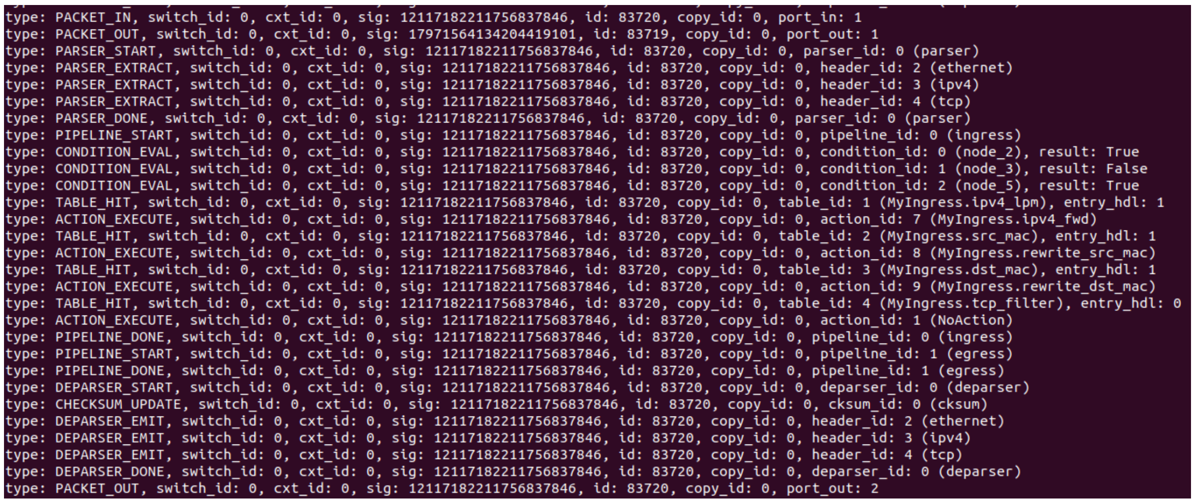
To test the firewall we used the commands given in the statement. So, we compile our p4 program and mininet script. Then we injected our table entries and got this:



```
RuntimeCmd: Adding entry to range match table tcp_filter
match key:           EXACT-0a:00:01:01  EXACT-0a:00:02:01       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 tcp_filter
match key:           EXACT-0a:00:02:01  EXACT-0a:00:01:01       RANGE-15:b3 -> 15:b3    RANGE-00:00 -> ff:ff
action:              NoAction
runtime data:
Entry has been added with handle 1
RuntimeCmd: Adding entry to range match table udp_filter
match key:           EXACT-0a:00:01:01  EXACT-0a:00:02:01       RANGE-00:00 -> ff:ff    RANGE-15:b3 -> 15:b3
action:              drop
runtime data:
Entry has been added with handle 0
RuntimeCmd: Adding entry to range match table udp_filter
match key:           EXACT-0a:00:02:01  EXACT-0a:00:01:01       RANGE-15:b3 -> 15:b3    RANGE-00:00 -> ff:ff
action:              drop
runtime data:
Entry has been added with handle 1
```

Based on this log, it appears that the P4 tables are being configured for packet processing, and specific actions are defined for different types of packets based on matching criteria. The log indicates successful runtime commands for adding entries to these tables.

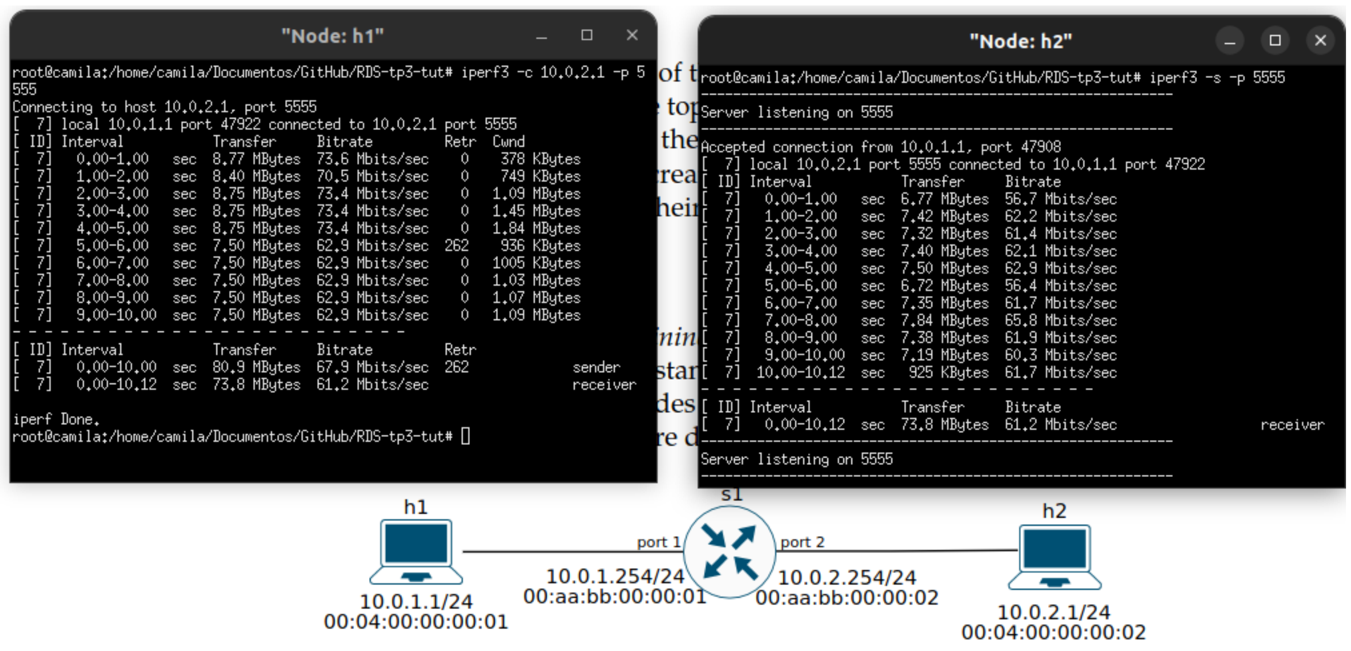
Let's check the log messages:



```
type: PACKET_IN, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, port_in: 1
type: PACKET_OUT, switch_id: 0, cxt_id: 0, sig: 17971564134204419101, id: 83719, copy_id: 0, port_out: 1
type: PARSER_START, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, parser_id: 0 (parser)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, header_id: 2 (ethernet)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, header_id: 3 (ipv4)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, header_id: 4 (tcp)
type: PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, parser_id: 0 (parser)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, pipeline_id: 0 (ingress)
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, condition_id: 0 (node_2), result: True
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, condition_id: 1 (node_3), result: False
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, condition_id: 2 (node_5), result: True
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, table_id: 1 (MyIngress.ipv4_lpm), entry_hdl: 1
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, action_id: 7 (MyIngress.ipv4_fwd)
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, table_id: 2 (MyIngress.src_mac), entry_hdl: 1
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, action_id: 8 (MyIngress.rewrite_src_mac)
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, table_id: 3 (MyIngress.dst_mac), entry_hdl: 1
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, action_id: 9 (MyIngress.rewrite_dst_mac)
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, table_id: 4 (MyIngress.tcp_filter), entry_hdl: 0
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, action_id: 1 (NoAction)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, pipeline_id: 0 (ingress)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, pipeline_id: 1 (egress)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, pipeline_id: 1 (egress)
type: DEPARSER_START, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, deparser_id: 0 (deparser)
type: CHECKSUM_UPDATE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, cksum_id: 0 (cksum)
type: DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, header_id: 2 (ethernet)
type: DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, header_id: 3 (ipv4)
type: DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, header_id: 4 (tcp)
type: DEPARSER_DONE, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, deparser_id: 0 (deparser)
type: PACKET_OUT, switch_id: 0, cxt_id: 0, sig: 12117182211756837846, id: 83720, copy_id: 0, port_out: 2
```

The log provides evidence of a seamless process in which header parsing is successfully accomplished. Notably, it reveals precise matches within the tables, leading to the execution of actions that align with the defined configurations, for example, the table entry for the tcp_filter must macth the table entries and the action must be NoAction, as we can see in the log that is happening.Finally,the deparse also is done successfully.

As a final test, let's check if the TCP traffic is forwarded correctly. For that lets run an iperf3 server on h2, on port 5555, and a client on h1.



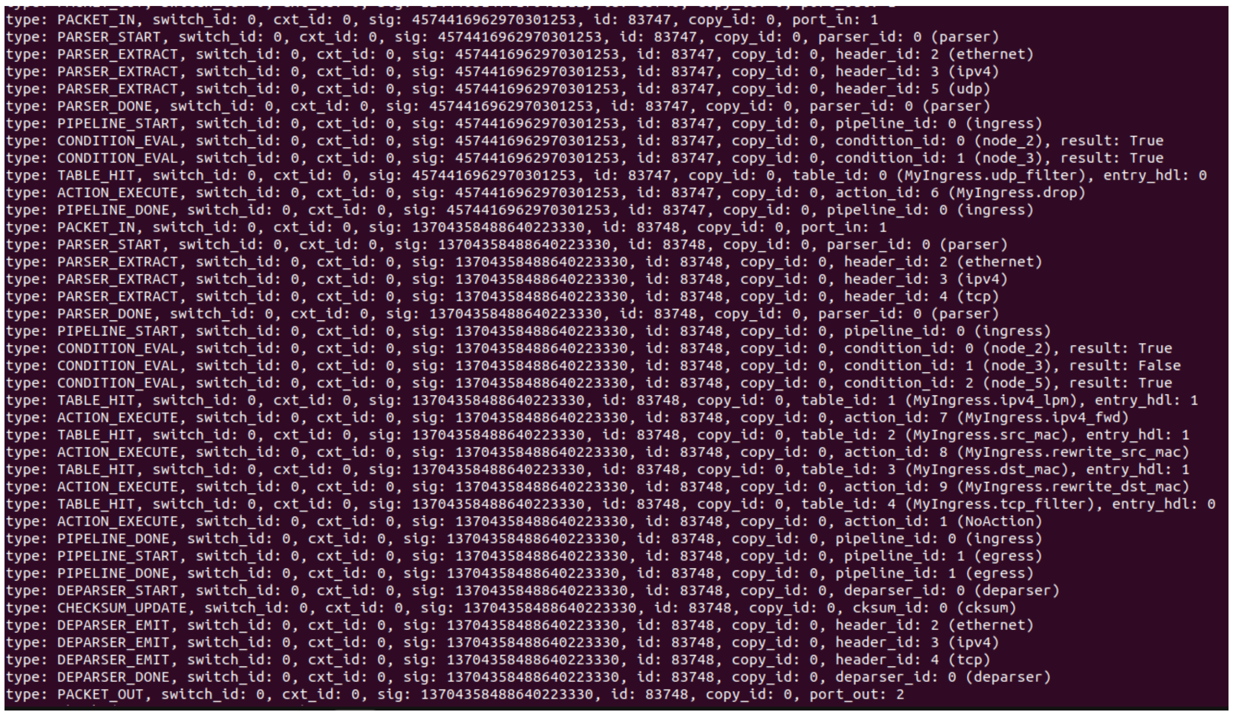
```
root@camila:/home/camila/Documentos/GitHub/RDS-tp3-tut# iperf3 -c 10.0.2.1 -p 5
555
Connecting to host 10.0.2.1, port 5555
[  7] local 10.0.1.1 port 47922 connected to 10.0.2.1 port 5555
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  7]   0.00-1.00   sec  8.77 MBytes  73.6 Mbits/sec    0    378 KBytes
[  7]   1.00-2.00   sec  8.40 MBytes  70.5 Mbits/sec    0    749 KBytes
[  7]   2.00-3.00   sec  8.75 MBytes  73.4 Mbits/sec    0   1.09 MBytes
[  7]   3.00-4.00   sec  8.75 MBytes  73.4 Mbits/sec    0   1.45 MBytes
[  7]   4.00-5.00   sec  8.75 MBytes  73.4 Mbits/sec    0   1.84 MBytes
[  7]   5.00-6.00   sec  7.50 MBytes  62.9 Mbits/sec  262    936 KBytes
[  7]   6.00-7.00   sec  7.50 MBytes  62.9 Mbits/sec    0   1005 KBytes
[  7]   7.00-8.00   sec  7.50 MBytes  62.9 Mbits/sec    0   1.03 MBytes
[  7]   8.00-9.00   sec  7.50 MBytes  62.9 Mbits/sec    0   1.07 MBytes
[  7]   9.00-10.00  sec  7.50 MBytes  62.9 Mbits/sec    0   1.09 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  7]   0.00-10.00  sec  80.9 MBytes  67.9 Mbits/sec  262             sender
[  7]   0.00-10.12  sec  73.8 MBytes  61.2 Mbits/sec                  receiver

iperf Done.
root@camila:/home/camila/Documentos/GitHub/RDS-tp3-tut# 
```

```
root@camila:/home/camila/Documentos/GitHub/RDS-tp3-tut# iperf3 -s -p 5555
-----------------------------------------------------------
Server listening on 5555
-----------------------------------------------------------
Accepted connection from 10.0.1.1, port 47908
[  7] local 10.0.2.1 port 5555 connected to 10.0.1.1 port 47922
[ ID] Interval           Transfer     Bitrate
[  7]   0.00-1.00   sec  6.77 MBytes  56.7 Mbits/sec
[  7]   1.00-2.00   sec  7.42 MBytes  62.2 Mbits/sec
[  7]   2.00-3.00   sec  7.32 MBytes  61.4 Mbits/sec
[  7]   3.00-4.00   sec  7.40 MBytes  62.1 Mbits/sec
[  7]   4.00-5.00   sec  7.50 MBytes  62.9 Mbits/sec
[  7]   5.00-6.00   sec  6.72 MBytes  56.4 Mbits/sec
[  7]   6.00-7.00   sec  7.35 MBytes  61.7 Mbits/sec
[  7]   7.00-8.00   sec  7.84 MBytes  65.8 Mbits/sec
[  7]   8.00-9.00   sec  7.38 MBytes  61.9 Mbits/sec
[  7]   9.00-10.00  sec  7.19 MBytes  60.3 Mbits/sec
[  7]  10.00-10.12  sec   925 KBytes  61.7 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  7]   0.00-10.12  sec  73.8 MBytes  61.2 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5555
-----------------------------------------------------------
```

As we can see, the traffic is being allowed and forwarded correctly.

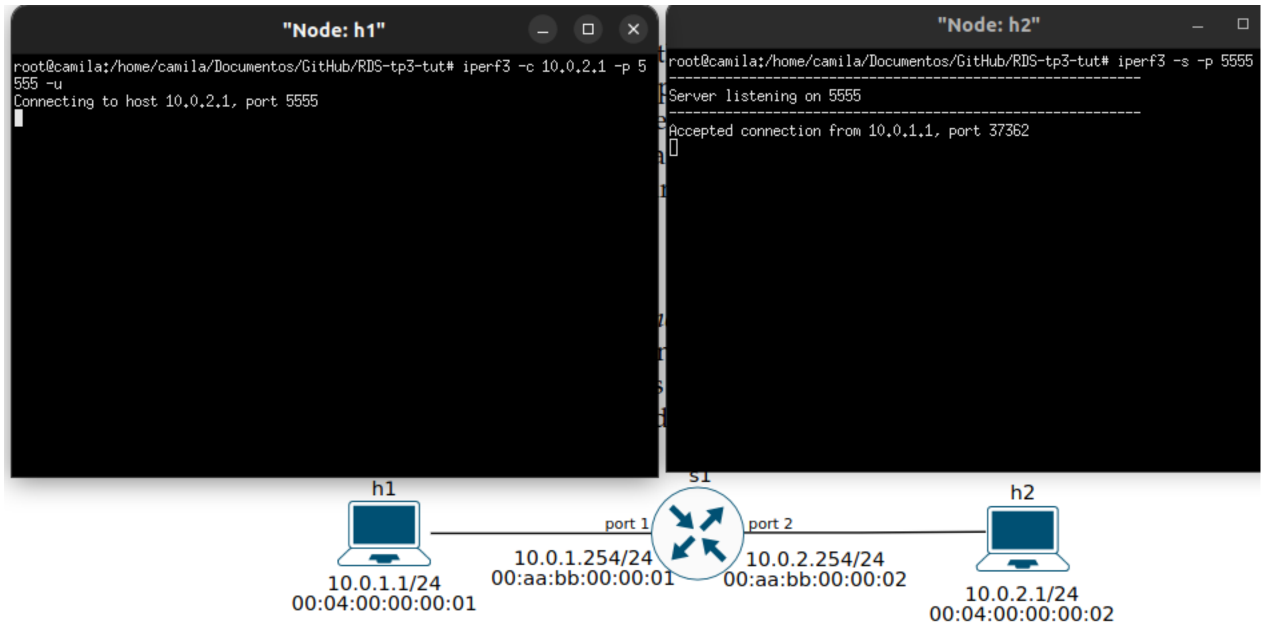
Now, let's verify whether UDP traffic is being dropped. To assess this, we will use the command 'iperf3 -u' to transmit UDP packets and check the log messages.



```
type: PACKET_IN, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, port_in: 1
type: PARSER_START, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, parser_id: 0 (parser)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, header_id: 2 (ethernet)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, header_id: 3 (ipv4)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, header_id: 5 (udp)
type: PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, parser_id: 0 (parser)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, pipeline_id: 0 (ingress)
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, condition_id: 0 (node_2), result: True
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, condition_id: 1 (node_3), result: True
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, table_id: 0 (MyIngress.udp_filter), entry_hdl: 0
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, action_id: 6 (MyIngress.drop)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 4574416962970301253, id: 83747, copy_id: 0, pipeline_id: 0 (ingress)
type: PACKET_IN, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, port_in: 1
type: PARSER_START, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, parser_id: 0 (parser)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, header_id: 2 (ethernet)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, header_id: 3 (ipv4)
type: PARSER_EXTRACT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, header_id: 4 (tcp)
type: PARSER_DONE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, parser_id: 0 (parser)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, pipeline_id: 0 (ingress)
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, condition_id: 0 (node_2), result: True
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, condition_id: 1 (node_3), result: False
type: CONDITION_EVAL, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, condition_id: 2 (node_5), result: True
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, table_id: 1 (MyIngress.ipv4_lpm), entry_hdl: 1
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, action_id: 7 (MyIngress.ipv4_fwd)
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, table_id: 2 (MyIngress.src_mac), entry_hdl: 1
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, action_id: 8 (MyIngress.rewrite_src_mac)
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, table_id: 3 (MyIngress.dst_mac), entry_hdl: 1
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, action_id: 9 (MyIngress.rewrite_dst_mac)
type: TABLE_HIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, table_id: 4 (MyIngress.tcp_filter), entry_hdl: 0
type: ACTION_EXECUTE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, action_id: 1 (NoAction)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, pipeline_id: 0 (ingress)
type: PIPELINE_START, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, pipeline_id: 1 (egress)
type: PIPELINE_DONE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, pipeline_id: 1 (egress)
type: DEPARSER_START, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, deparser_id: 0 (deparser)
type: CHECKSUM_UPDATE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, cksum_id: 0 (cksum)
type: DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, header_id: 2 (ethernet)
type: DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, header_id: 3 (ipv4)
type: DEPARSER_EMIT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, header_id: 4 (tcp)
type: DEPARSER_DONE, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, deparser_id: 0 (deparser)
type: PACKET_OUT, switch_id: 0, cxt_id: 0, sig: 13704358488640223330, id: 83748, copy_id: 0, port_out: 2
```

Upon inspecting the log messages, it becomes apparent that matches are present in the udp _table (Table HIT ... MyIngress.udp _filter), with the corresponding action set to 'drop' (MyIngress.drop). Consequently, it is evident that the traffic is not permitted.

Next, let's examine the outcome of the 'iperf3 -u' command to determine whether the packets are being received or not:



```
root@camila:/home/camila/Documentos/GitHub/RDS-tp3-tut# iperf3 -c 10.0.2.1 -p 5
555 -u
Connecting to host 10.0.2.1, port 5555
```

```
root@camila:/home/camila/Documentos/GitHub/RDS-tp3-tut# iperf3 -s -p 5555
-----------------------------------------------------------
Server listening on 5555
-----------------------------------------------------------
Accepted connection from 10.0.1.1, port 37362
```

As observed, the connection is being established, but no packets are being received. This indicates that UDP traffic is not allowed, affirming the proper functioning of the firewall: for h1 any port to h2 port 5555 and h2 port 5555 to h1 any port, UDP traffic is being blocked, and only TCP traffic is being allowed.

2.4 User Guide

(In mininet terminal)

- Server(Node h2): iperf3 -s -p 5555
- Client(Node h1): iperf3 -c 10.0.2.1 -p 5555

For testing udp packets:

- Server(Node h2): iperf3 -s -p 5555
- Client(Node h1): iperf3 -c -u 10.0.2.1 -p 5555

3 Conclusion

In summary, this project has been instrumental in providing a hands-on exploration of Software Defined Networks (SDNs), with a specific focus on data plane programming using the P4 language. The tutorial on constructing a simple router helped us gain insights into the V1 Model architecture, understand different blocks, and acquire programming proficiency in P4. This way we were able to learn and apply the knowledge to the proposed exercise for building a firewall, which we successfully implemented.