# Hardwarepraktikum Internet-Technologien

*Task 8: Software-Defined Networking*

---


**Julius-Maximilians-Universität Würzburg**

Chair of Computer Science III

A project report submitted by **Group 11**

Enrique Mesonero Ronco

Manuel Calvo Martín

Pablo E. Ortega Ureña


Summer Term 2022

July, 2022

# Contents

# 8. Software-Defined Networking

## 8.2. Installation of the SDN Switch

Our connections were as follows:
SDN Switch

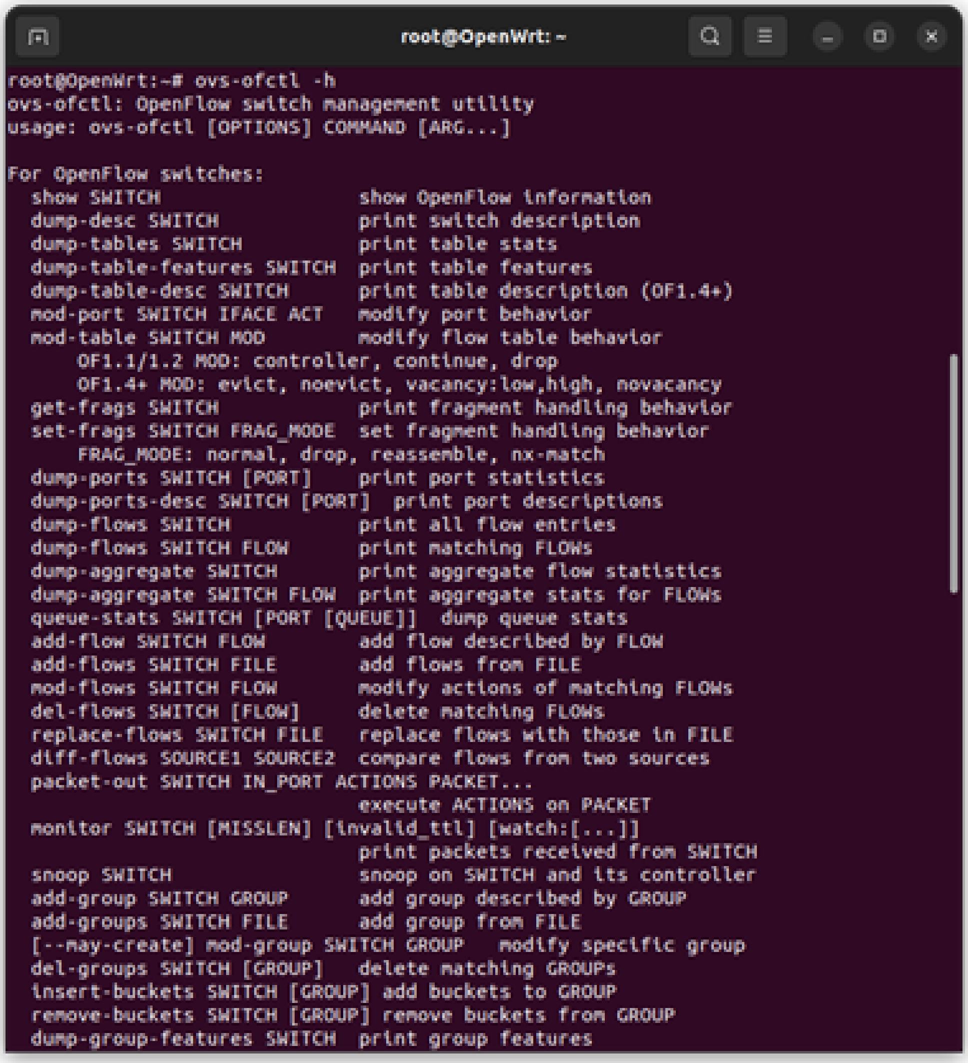
- Port 1 (Internet/Management) to Port 2 of the Switch
- Port 2 to Port 4 of the Switch
- Port 3 to PC A

Switch

- Port 2 to Port 1 (Internet/Management) of the SDN Switch
- Port 3 to PC B (acting as the controller)
- Port 4 to Port 2 of the SDN Switch
- Port 5 to Raspberry Pi

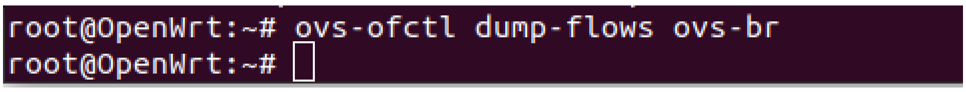
The connection between the controller PC and the SDN Switch is successful. We check that, as it has been just turned on, there are no existing table entries or flows, using the dump-flows command. We also check the manual for understanding future commands.


*Figure 8.1: Physical topology of the session*



```
root@OpenWrt: ~
root@OpenWrt:~# ovs-ofctl -h
ovs-ofctl: OpenFlow switch management utility
usage: ovs-ofctl [OPTIONS] COMMAND [ARG...]

For OpenFlow switches:
  show SWITCH                 show OpenFlow information
  dump-desc SWITCH            print switch description
  dump-tables SWITCH          print table stats
  dump-table-features SWITCH  print table features
  dump-table-desc SWITCH      print table description (OF1.4+)
  mod-port SWITCH IFACE ACT   modify port behavior
  mod-table SWITCH MOD        modify flow table behavior
      OF1.1/1.2 MOD: controller, continue, drop
      OF1.4+ MOD: evict, noevict, vacancy:low,high, novacancy
  get-frags SWITCH            print fragment handling behavior
  set-frags SWITCH FRAG_MODE  set fragment handling behavior
      FRAG_MODE: normal, drop, reassemble, nx-match
  dump-ports SWITCH [PORT]    print port statistics
  dump-ports-desc SWITCH [PORT]  print port descriptions
  dump-flows SWITCH           print all flow entries
  dump-flows SWITCH FLOW      print matching FLOWs
  dump-aggregate SWITCH       print aggregate flow statistics
  dump-aggregate SWITCH FLOW  print aggregate stats for FLOWs
  queue-stats SWITCH [PORT [QUEUE]]  dump queue stats
  add-flow SWITCH FLOW        add flow described by FLOW
  add-flows SWITCH FILE       add flows from FILE
  mod-flows SWITCH FLOW       modify actions of matching FLOWs
  del-flows SWITCH [FLOW]     delete matching FLOWs
  replace-flows SWITCH FILE   replace flows with those in FILE
  diff-flows SOURCE1 SOURCE2  compare flows from two sources
  packet-out SWITCH IN_PORT ACTIONS PACKET...
                              execute ACTIONS on PACKET
  monitor SWITCH [MISSLEN] [invalid_ttl] [watch:[...]]
                              print packets received from SWITCH
  snoop SWITCH                snoop on SWITCH and its controller
  add-group SWITCH GROUP      add group described by GROUP
  add-groups SWITCH FILE      add group from FILE
  [--may-create] mod-group SWITCH GROUP   modify specific group
  del-groups SWITCH [GROUP]   delete matching GROUPs
  insert-buckets SWITCH [GROUP] add buckets to GROUP
  remove-buckets SWITCH [GROUP] remove buckets from GROUP
  dump-group-features SWITCH  print group features
```

*Figure 8.2: Checking the manual*



```
root@OpenWrt:~# ovs-ofctl dump-flows ovs-br
root@OpenWrt:~#
```

*Figure 8.3: Checking active flows*

## 8.3. Simple OpenFlow rules

We started pinging the Raspberry, the Switch and the PC B from PC A, connected to the SDN. We checked there was no connection yet. After that, we added two flow rules to our SDN, one for each direction of the network. After adding them, the ping was now possible.

```
athenyx@athenyx-boreas:~$ ping 10.11.1.1
PING 10.11.1.1 (10.11.1.1) 56(84) bytes of data.
^C
--- 10.11.1.1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2053ms

athenyx@athenyx-boreas:~$ ping 10.11.1.3
PING 10.11.1.3 (10.11.1.3) 56(84) bytes of data.
^C
--- 10.11.1.3 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5110ms

athenyx@athenyx-boreas:~$ ping 10.11.1.5
PING 10.11.1.5 (10.11.1.5) 56(84) bytes of data.
^C
--- 10.11.1.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3054ms
```

*Figure 8.4: Pinging the devices from PC A before adding the rules*

```
root@OpenWrt: ~
=== WARNING! =====================================
There is no root password defined on this device!
Use the "passwd" command to set up a new password
in order to prevent unauthorized SSH logins.
--------------------------------------------------
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 in_port=2 actions=3"
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 in_port=3 actions=2"
root@OpenWrt:~# ovs-ofctl dump-flows ovs-br
 cookie=0x0, duration=22.589s, table=0, n_packets=11, n_bytes=660, in_port="eth0
.2" actions=output:"eth0.3"
 cookie=0x0, duration=15.748s, table=0, n_packets=0, n_bytes=0, in_port="eth0.3"
 actions=output:"eth0.2"
root@OpenWrt:~# ovs-ofctl del-flows
ovs-ofctl: 'del-flows' command requires at least 1 arguments
root@OpenWrt:~# ovs-ofctl del-flows ovs-br
root@OpenWrt:~# ovs-ofctl dump-flows ovs-br
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 in_port=3 actions=2"
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 in_port=2 actions=3"
root@OpenWrt:~# ovs-ofctl dump-flows ovs-br
 cookie=0x0, duration=103.791s, table=0, n_packets=39, n_bytes=3290, in_port="et
h0.3" actions=output:"eth0.2"
 cookie=0x0, duration=47.384s, table=0, n_packets=44, n_bytes=3096, in_port="eth
0.2" actions=output:"eth0.3"
root@OpenWrt:~#
```

*Figure 8.5: Rules added, testing the packet flow*

We checked that both the packet and the bytes number under the rules increased with each ping and reply.

```
athenyx@athenyx-boreas:~$ ping 10.11.1.1
PING 10.11.1.1 (10.11.1.1) 56(84) bytes of data.
64 bytes from 10.11.1.1: icmp_seq=1 ttl=64 time=3.47 ms
64 bytes from 10.11.1.1: icmp_seq=2 ttl=64 time=0.409 ms
64 bytes from 10.11.1.1: icmp_seq=3 ttl=64 time=0.438 ms
64 bytes from 10.11.1.1: icmp_seq=4 ttl=64 time=0.420 ms
^C
--- 10.11.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3037ms
rtt min/avg/max/mdev = 0.409/1.185/3.473/1.321 ms
athenyx@athenyx-boreas:~$ ping 10.11.1.3
PING 10.11.1.3 (10.11.1.3) 56(84) bytes of data.
64 bytes from 10.11.1.3: icmp_seq=1 ttl=64 time=3.41 ms
64 bytes from 10.11.1.3: icmp_seq=2 ttl=64 time=0.344 ms
64 bytes from 10.11.1.3: icmp_seq=3 ttl=64 time=0.459 ms
^C
--- 10.11.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.344/1.404/3.410/1.418 ms
athenyx@athenyx-boreas:~$ ping 10.11.1.5
PING 10.11.1.5 (10.11.1.5) 56(84) bytes of data.
64 bytes from 10.11.1.5: icmp_seq=1 ttl=64 time=6.49 ms
64 bytes from 10.11.1.5: icmp_seq=2 ttl=64 time=2.46 ms
64 bytes from 10.11.1.5: icmp_seq=3 ttl=64 time=2.40 ms
^C
--- 10.11.1.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 2.403/3.783/6.490/1.914 ms
```

*Figure 8.6: Pinging the devices from PC A after adding the rules*

Following the next step, we deleted the existing flows and added new ones based on the MAC addresses of our devices. Since ARP requests should be also transmitted correctly, we added an additional rule where packets with MAC address equal to the broadcast address must be flooded.

```
root@OpenWrt:~# ovs-ofctl del-flows ovs-br
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 dl_dst=e4:5f:01:05:f9:e0 acti
on=2"
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 dl_dst=0c:9d:92:c7:4f:e5 acti
on=3"
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 dl_dst=08:00:27:6f:0c:ef acti
on=2"
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 dl_dst=ff:ff:ff:ff:ff:ff acti
on=flood"
root@OpenWrt:~# ovs-ofctl add-flow ovs-br "table=0 dl_dst=64:d1:54:9a:4a:71 acti
on=2"
```

*Figure 8.7: New layer 2 flow rules added*

The connection was again successfully established without issues.

Afterwards, we removed those rules and tried forwarding based on layer 3. We retained the flood on broadcast rule, but changed those referring to MAC addresses to their IP address.

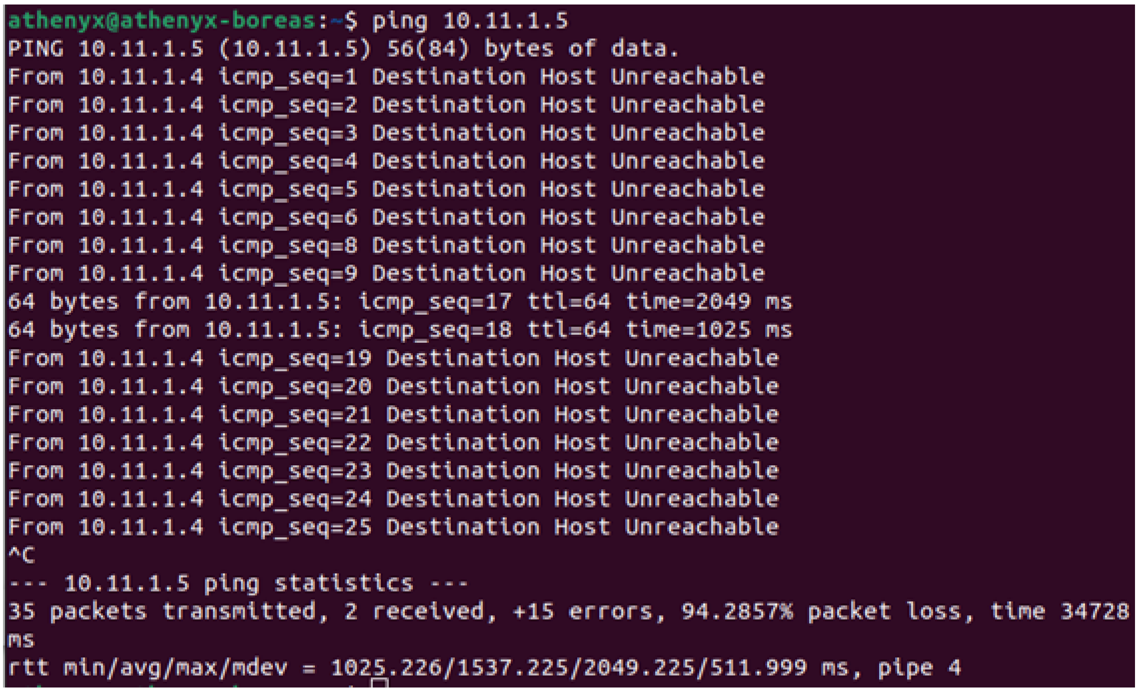


*Figure 8.8: Awkward behavior of the ping after the IP flow rules*

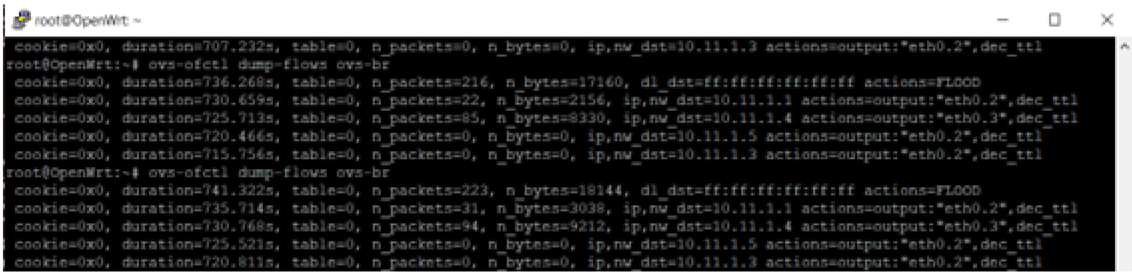


*Figure 8.9: IP flow rules and their data flow*

This configuration failed most of the time, but looking at Wireshark we realized our problem. The ARP request was reaching the computer which we were trying to obtain the MAC address from, thanks to the flood on broadcast rule. However, since the ARP reply was directed to a specific MAC address, once it reached the SDN switch, there were no rules regarding that MAC address, only the IP address, so the ARP reply got discarded.

It was then, after looking at the OVS fields document, that we found out that our SDN could look at the packet, and if it was ARP, check the target IP address contained in that packet.

```
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.1.1 priority=2 action=dec_ttl,2" #
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.1.4 priority=2 action=dec_ttl,3" #
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.1.5 priority=2 action=dec_ttl,2" #
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.1.3 priority=2 action=dec_ttl,2" #
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.1.2 priority=2 action=dec_ttl,2" #
```

*Figure 8.10: IP flow rules*

```
ovs-ofctl add-flow ovs-br "table=0 arp arp_tpa=10.11.1.1 priority=1 action=2" #
ovs-ofctl add-flow ovs-br "table=0 arp arp_tpa=10.11.1.4 priority=1 action=3" #
ovs-ofctl add-flow ovs-br "table=0 arp arp_tpa=10.11.1.5 priority=1 action=2" #
ovs-ofctl add-flow ovs-br "table=0 arp arp_tpa=10.11.1.3 priority=1 action=2" #
ovs-ofctl add-flow ovs-br "table=0 arp arp_tpa=10.11.1.2 priority=1 action=2" #
```

*Figure 8.11: ARP flow rules*

With these settings, our pings started working again. We were also able to check that if we set the TTL of the packet under 2, the packet would get discarded by the SDN and never reach its target for a response.

| | | | | | |
|---|---|---|---|---|---|
| 576 | 2453.7199915… | e4:5f:01:05:f9:e0 | Tp-LinkT_d0:25:ac | ARP | 60 Who has 10.11.1.4? Tell 10.11.1.1 |
| 577 | 2453.7200116… | Tp-LinkT_d0:25:ac | e4:5f:01:05:f9:e0 | ARP | 42 10.11.1.4 is at 50:3e:aa:d0:25:ac |
| 578 | 2454.5945553… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=1/256, ttl=1 (no response… |
| 579 | 2455.5957198… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=2/512, ttl=1 (no response… |
| 580 | 2456.6196914… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=3/768, ttl=1 (no response… |
| 581 | 2457.6437436… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=4/1024, ttl=1 (no respons… |
| 582 | 2458.6677248… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=5/1280, ttl=1 (no respons… |
| 583 | 2459.6916835… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=6/1536, ttl=1 (no respons… |
| 584 | 2460.7157128… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=7/1792, ttl=1 (no respons… |
| 585 | 2461.7396750… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=8/2048, ttl=1 (no respons… |
| 586 | 2462.7637085… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=9/2304, ttl=1 (no respons… |
| 587 | 2463.7876993… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=10/2560, ttl=1 (no respon… |
| 588 | 2464.8116840… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0025, seq=11/2816, ttl=1 (no respon… |
| 589 | 2468.8184743… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0026, seq=1/256, ttl=2 (reply in 59… |
| 590 | 2468.8208133… | 10.11.1.1 | 10.11.1.4 | ICMP | 98 Echo (ping) reply id=0x0026, seq=1/256, ttl=63 (request in… |
| 591 | 2469.8201745… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0026, seq=2/512, ttl=2 (reply in 59… |
| 592 | 2469.8221845… | 10.11.1.1 | 10.11.1.4 | ICMP | 98 Echo (ping) reply id=0x0026, seq=2/512, ttl=63 (request in… |
| 593 | 2470.8214266… | 10.11.1.4 | 10.11.1.1 | ICMP | 98 Echo (ping) request id=0x0026, seq=3/768, ttl=2 (reply in 59… |
| 594 | 2470.8230950… | 10.11.1.1 | 10.11.1.4 | ICMP | 98 Echo (ping) reply id=0x0026, seq=3/768, ttl=63 (request in… |

*Figure 8.11: Replies to ICMP in Wireshark with different TTLs*

## 8.4. Replication of a static NAT router with firewall

For this subtask, we adjusted the IP address of PC A from 10.11.1.4 to 10.11.123.1. We also set 10.11.123.254 as its gateway, and also changed the Raspberry Pi gateway to be 10.11.1.254.

| | |
|---|---|
| Verbindungsgeschwindigkeit | 1000 Mb/s |
| IPv4-Adresse | 10.11.123.1 |
| IPv6-Adresse | fe80::cff8:a3f1:b007:4c72 |
| Geräteadresse | 50:3E:AA:D0:25:AC |
| Vorgabestrecke | 10.11.123.254 |
| DNS | |

*Figure 8.12: New IP and gateway for PC A*

The connection didn't work yet between the two subnets, as there is no way for the ARP requests to reach the other port.

```
ovs-ofctl add-flow ovs-br "table=0 in_port=2 priority=100 arp nw_dst=10.11.1.254
actions=load:0x2->arp_op,move:arp_spa->arp_tpa,move:arp_sha->arp_tha,move:eth_src->eth_ds
t,set_field:64:d1:54:22:22:22->eth_src,set_field:64:d1:54:22:22:22->arp_sha,set_field:10.
11.1.254->arp_spa,output:in_port" #
ovs-ofctl add-flow ovs-br "table=0 in_port=3 priority=100 arp nw_dst=10.11.123.254
actions=load:0x2->arp_op,move:arp_spa->arp_tpa,move:arp_sha->arp_tha,move:eth_src->eth_ds
t,set_field:64:d1:54:33:33:33->eth_src,set_field:64:d1:54:33:33:33->arp_sha,set_field:10.
11.123.254->arp_spa,output:in_port" #
```

*Figure 8.13: ARP configuration*

For that communication to be made, we used the two flow rules above, where whenever a connection from the switch port (Port 2) was directed to the gateway IP, the SDN would set-up an ARP reply (as indicated by the 2 opcode), directed to the MAC address of the sender (changed to target now), with the made-up address of our port 2, and the gateway address. The SDN sends that through the same port. This way, now every device connected to the switch knows about the SDN switch gateway. In similar fashion, another rule is made for port 3, but with the subnet 10.11.123.0/24.

Now that the devices can reach the gateways, the SDN router only needs to know (as this is proactive forwarding) what to do whenever a packet from our end devices reaches those gateways. We used PC A and the Raspberry for this.

```
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.1.0/24
action=set_field:64:d1:54:22:22:22->eth_src,set_field:e4:5f:01:05:f9:e0->eth_dst,
dec_ttl,2" #
ovs-ofctl add-flow ovs-br "table=0 ip nw_dst=10.11.123.0/24
action=set_field:64:d1:54:33:33:33->eth_src,set_field:50:3e:aa:d0:25:ac->eth_dst,
dec_ttl,3" #
```

*Figure 8.14: IP configuration*

These are pretty similar to the IP flow rules of the previous subtask. However now, the layer 2 source and destination addresses are set, with the MAC addresses of the used port and the end device, respectively.

| | | | | | | |
|---|---|---|---|---|---|---|
| 53 | 106.059655115 | Tp-LinkT_d0:25:ac | Broadcast | ARP | 42 | Who has 10.11.123.254? Tell 10.11.123.1 |
| 54 | 106.062173093 | Routerbo_33:33:33 | Tp-LinkT_d0:25:ac | ARP | 60 | 10.11.123.254 is at 64:d1:54:33:33:33 |
| 55 | 106.062193256 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=1/256, ttl=64 (reply in 5 |
| 56 | 106.064540368 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=1/256, ttl=63 (request in |
| 57 | 107.060780195 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=2/512, ttl=64 (reply in 5 |
| 58 | 107.063050774 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=2/512, ttl=63 (request in |
| 59 | 108.062330759 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=3/768, ttl=64 (reply in 6 |
| 60 | 108.063020960 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=3/768, ttl=63 (request in |
| 61 | 109.071243838 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=4/1024, ttl=64 (reply in |
| 62 | 109.073387366 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=4/1024, ttl=63 (request i |
| 63 | 110.072621837 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=5/1280, ttl=64 (reply in |
| 64 | 110.073409792 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=5/1280, ttl=63 (request i |
| 65 | 111.083234095 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=6/1536, ttl=64 (reply in |
| 66 | 111.085595758 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=6/1536, ttl=63 (request i |
| 67 | 112.084866347 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002a, seq=7/1792, ttl=64 (reply in |
| 68 | 112.087192351 | 10.11.1.1 | 10.11.123.1 | ICMP | 98 | Echo (ping) reply id=0x002a, seq=7/1792, ttl=63 (request i |
| 69 | 129.327078878 | fe80::cff8:a3f1:b00… | ff02::fb | MDNS | 203 | Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR |
| 70 | 129.327248854 | 10.11.123.1 | 224.0.0.251 | MDNS | 183 | Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR |
| 71 | 129.963995949 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002b, seq=1/256, ttl=1 (no response |
| 72 | 130.987286703 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002b, seq=2/512, ttl=1 (no response |
| 73 | 132.011260266 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002b, seq=3/768, ttl=1 (no response |
| 74 | 133.035287846 | 10.11.123.1 | 10.11.1.1 | ICMP | 98 | Echo (ping) request id=0x002b, seq=4/1024, ttl=1 (no respons |

*Figure 8.15: Checking with Wireshark*

With this, communication is again restored. We also checked that with a TTL of 1, it still wouldn't work. This matches what our router did in previous sessions, and what ofproto/trace tells us.

root@OpenWrt: ~

```
ovs-appctl: ovs-vswitchd: server returned an error
root@OpenWrt:~# ovs-appctl ofproto/trace ovs-br in_port=3,icmp,nw_src=10.11.123.
1,nw_dst=10.11.1.1
Flow: icmp,in_port=3,vlan_tci=0x0000,dl_src=00:00:00:00:00:00,dl_dst=00:00:00:00
:00:00,nw_src=10.11.123.1,nw_dst=10.11.1.1,nw_tos=0,nw_ecn=0,nw_ttl=0,icmp_type=
0,icmp_code=0

bridge("ovs-br")
----------------
 0. ip,nw_dst=10.11.1.0/24, priority 32768
    mod_dl_src:64:d1:54:22:22:22
    mod_dl_dst:e4:5f:01:05:f9:e0
    dec_ttl
     >> IPv4 decrement TTL exception

Final flow: icmp,in_port=3,vlan_tci=0x0000,dl_src=64:d1:54:22:22:22,dl_dst=e4:5f
:01:05:f9:e0,nw_src=10.11.123.1,nw_dst=10.11.1.1,nw_tos=0,nw_ecn=0,nw_ttl=0,icmp
_type=0,icmp_code=0
Megaflow: recirc_id=0,eth,ip,in_port=3,dl_src=00:00:00:00:00:00,dl_dst=00:00:00:
00:00:00,nw_src=8.0.0.0/5,nw_dst=10.11.1.0/25,nw_ttl=0,nw_frag=no
Datapath actions: set(eth(src=64:d1:54:22:22:22,dst=e4:5f:01:05:f9:e0)),userspac
e(pid=2255733610,controller(reason=2,dont_send=1,continuation=0,recirc_id=80,rul
e_cookie=0,controller_id=0,max_len=65535))
root@OpenWrt:~#
```

*Figure 8.16: Ofproto/trace result*

Next, we moved towards the NAT/Firewall part. For that, we set that any packet directed to the subnet 10.11.1.0/24, incoming from port 3 (PC A's port) would be drop, unless of course, it was directed specifically to the gateway's address (10.11.1.254) and to port 12221. Whenever the NAT would receive this kind of packet, it would change the IP field to the Raspberry one, and the port to 22, the SSH one.

This must be bidirectional, so we also set that any packet incoming from port 2 towards PC A, would have its IP masked to the gateway one, and the port changed back to 12221. It at first didn't work for us, as we forgot to add this to the previous rules and instead made new rules. Since both rules matched, but were different, and didn't have any priority attached, only the first added rule (which was more general) was acting. However, to solve this, we combined both the previous IP flow rules and the NAT/Firewall rules. The SSH to port 12221 was now working, with all other connections inside the 10.11.1.0/24 subnet rejected, and the IP correctly masked behind the NAT.

```
ovs-ofctl add-flow ovs-br "table=0 in_port=3 tcp nw_dst=10.11.1.254 tcp_dst=12221
action=set_field:10.11.1.1->nw_dst,set_field:22->tcp_dst,set_field:64:d1:54:22:22:22->eth
_src,set_field:e4:5f:01:05:f9:e0->eth_dst, dec_ttl,2" #

ovs-ofctl add-flow ovs-br "table=0 in_port=3 tcp nw_dst=10.11.1.0/24 action=drop" #
ovs-ofctl add-flow ovs-br "table=0 in_port=2 tcp nw_dst=10.11.123.0/24
action=set_field:64:d1:54:33:33:33->eth_src,set_field:50:3e:aa:d0:25:ac->eth_dst,set_fiel
d:12221->tcp_src,dec_ttl,set_field:10.11.1.254->nw_src,3" #
```

*Figure 8.17: New TCP rules for NAT*

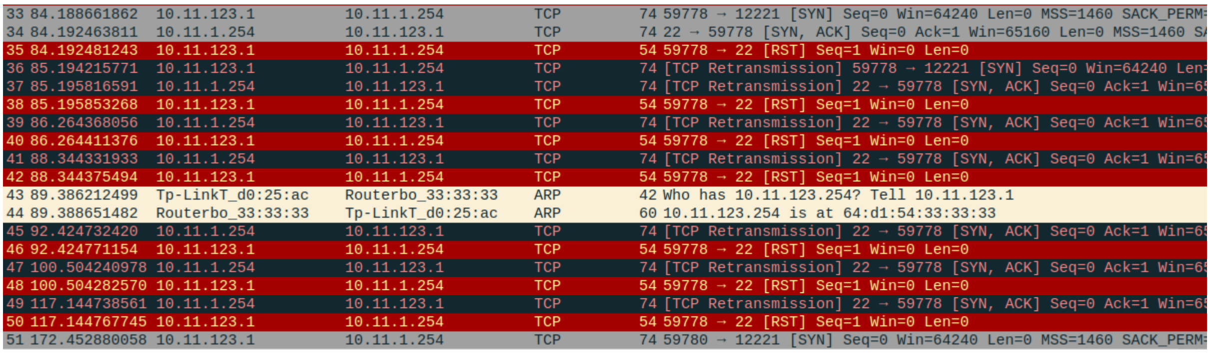
| | | | | | | |
|---|---|---|---|---|---|---|
| 51 | 172.452880058 | 10.11.123.1 | 10.11.1.254 | TCP | 74 | 59780 → 12221 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=… |
| 52 | 172.456325021 | 10.11.1.254 | 10.11.123.1 | TCP | 74 | 12221 → 59780 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460… |
| 53 | 172.456341322 | 10.11.123.1 | 10.11.1.254 | TCP | 66 | 59780 → 12221 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1013073… |
| 54 | 172.456528203 | 10.11.123.1 | 10.11.1.254 | TCP | 107 | 59780 → 12221 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=41 TSval=1… |
| 55 | 172.456985628 | 10.11.1.254 | 10.11.123.1 | TCP | 66 | 12221 → 59780 [ACK] Seq=1 Ack=42 Win=65152 Len=0 TSval=547591… |
| 56 | 172.509468915 | 10.11.1.254 | 10.11.123.1 | TCP | 114 | 12221 → 59780 [PSH, ACK] Seq=1 Ack=42 Win=65152 Len=48 TSval=… |
| 57 | 172.509497392 | 10.11.123.1 | 10.11.1.254 | TCP | 66 | 59780 → 12221 [ACK] Seq=42 Ack=49 Win=64256 Len=0 TSval=10130… |
| 58 | 172.510029277 | 10.11.123.1 | 10.11.1.254 | TCP | 1578 | 59780 → 12221 [PSH, ACK] Seq=42 Ack=49 Win=64256 Len=1512 TSv… |

*Figure 8.18: TCP connection established with the new port and gateway*

```
hwp@hwp-l:~$ ssh pi@10.11.1.254 -p 12221
The authenticity of host '[10.11.1.254]:12221 ([10.11.1.254]:12221)' can't be established.
ECDSA key fingerprint is SHA256:oh80ILNnbcnum7407C2C/kJdQfqivNm7tpJwG1L6ly0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.11.1.254]:12221' (ECDSA) to the list of known hosts.
pi@10.11.1.254's password:
Linux raspberrypi 5.10.17-v7l+ #1414 SMP Fri Apr 30 13:20:47 BST 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed May 18 13:44:43 2022 from 10.11.123.1
pi@raspberrypi:~ $ ^C
pi@raspberrypi:~ $ exit
Abgemeldet
Connection to 10.11.1.254 closed.
hwp@hwp-l:~$ ping 10.11.1.1
PING 10.11.1.1 (10.11.1.1) 56(84) Bytes Daten.
^C
--- 10.11.1.1 ping statistics ---
4 Pakete übertragen, 0 empfangen, 100% Paketverlust, Zeit 3059ms

hwp@hwp-l:~$ ping 10.11.1.3
PING 10.11.1.3 (10.11.1.3) 56(84) Bytes Daten.
^C
--- 10.11.1.3 ping statistics ---
3 Pakete übertragen, 0 empfangen, 100% Paketverlust, Zeit 2051ms

hwp@hwp-l:~$ ping 10.11.1.5
PING 10.11.1.5 (10.11.1.5) 56(84) Bytes Daten.
^C
--- 10.11.1.5 ping statistics ---
3 Pakete übertragen, 0 empfangen, 100% Paketverlust, Zeit 2027ms
```

*Figure 8.19: Pings to different devices and SSH success*

Finally, we also made another rule to be able to access PC B from PC A, on port 10810, in order to read the temperature sensor data. At first it didn't work for us, even when using the same configuration used to connect to the Raspberry Pi, and making sure the IP and MAC addresses were correct. The connection kept resetting.



| | | | | | |
|---|---|---|---|---|---|
| 33 | 84.188661862 | 10.11.123.1 | 10.11.1.254 | TCP | 74 59778 → 12221 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM= |
| 34 | 84.192463811 | 10.11.1.254 | 10.11.123.1 | TCP | 74 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SA |
| 35 | 84.192481243 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 36 | 85.194215771 | 10.11.123.1 | 10.11.1.254 | TCP | 74 [TCP Retransmission] 59778 → 12221 [SYN] Seq=0 Win=64240 Len= |
| 37 | 85.195816591 | 10.11.1.254 | 10.11.123.1 | TCP | 74 [TCP Retransmission] 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65 |
| 38 | 85.195853268 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 39 | 86.264368056 | 10.11.1.254 | 10.11.123.1 | TCP | 74 [TCP Retransmission] 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65 |
| 40 | 86.264411376 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 41 | 88.344331933 | 10.11.1.254 | 10.11.123.1 | TCP | 74 [TCP Retransmission] 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65 |
| 42 | 88.344375494 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 43 | 89.386212499 | Tp-LinkT_d0:25:ac | Routerbo_33:33:33 | ARP | 42 Who has 10.11.123.254? Tell 10.11.123.1 |
| 44 | 89.388651482 | Routerbo_33:33:33 | Tp-LinkT_d0:25:ac | ARP | 60 10.11.123.254 is at 64:d1:54:33:33:33 |
| 45 | 92.424732420 | 10.11.1.254 | 10.11.123.1 | TCP | 74 [TCP Retransmission] 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65 |
| 46 | 92.424771154 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 47 | 100.504240978 | 10.11.1.254 | 10.11.123.1 | TCP | 74 [TCP Retransmission] 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65 |
| 48 | 100.504282570 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 49 | 117.144738561 | 10.11.1.254 | 10.11.123.1 | TCP | 74 [TCP Retransmission] 22 → 59778 [SYN, ACK] Seq=0 Ack=1 Win=65 |
| 50 | 117.144767745 | 10.11.123.1 | 10.11.1.254 | TCP | 54 59778 → 22 [RST] Seq=1 Win=0 Len=0 |
| 51 | 172.452880058 | 10.11.123.1 | 10.11.1.254 | TCP | 74 59780 → 12221 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM= |

*Figure 8.20: Failure to connect to the DASH server*

Figure 8.20 shows the same kind of error we got; however, these were directed to port 22. When it was instead 8050 (DASH server) or 10810, the same errors popped up.

After some help, we fixed it, realizing it was an error coming from the DASH server file, as it didn't permit connections from outside networks. To fix it, we added host = '0.0.0.0' to the app.run_server line. With that, the connection was successful.

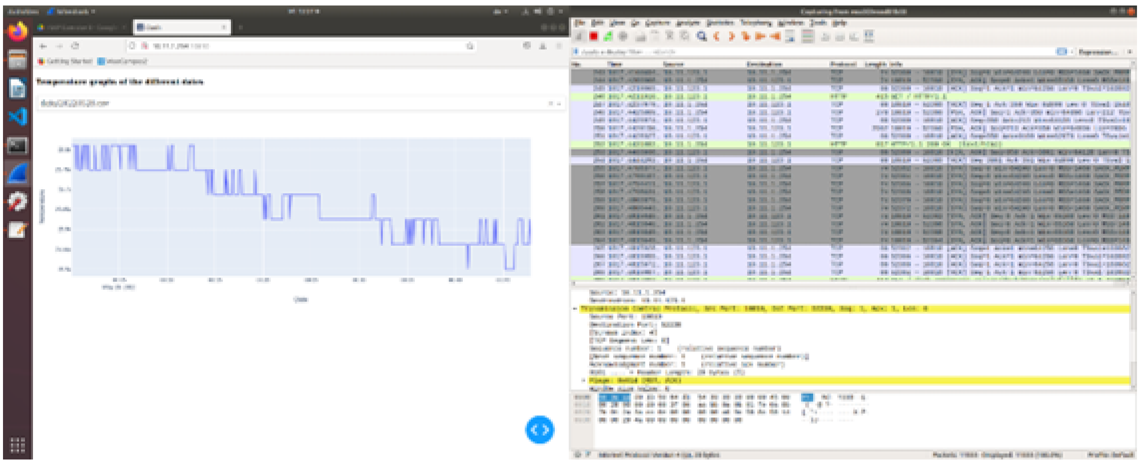


*Figure 8.21: Success on connecting to the DASH server*