

**[PRIN]**

Warsztaty 1-8

**Bartoszłomiej**

14 września 2023

# 1 Instalacja wymaganego oprogramowania

W celu wirtualizacji został wykorzystany hipernadzorca Virtualbox. Zgodnie z rekomendacją Przewodzącego warsztat użyto systemu operacyjnego Ubuntu 20.04 LTS. Następnie został zainstalowany emulator sieci Mininet oraz sterowniki: Ryu oraz ONOS.

## 2 Ryu testing

Po instalacji środowiska zostały przeprowadzone następujące testy w celu weryfikacji poprawności instalacji:

1. Test automatyczny emulatora Mininet
2. Obserwacja ruchu pakietów w programie Wireshark

### 2.1 Test automatyczny emulatora Mininet

W celu użycia sterownika Ryu wraz z prostym switch'em należy uruchomić go za pomocą komendy:

```
ryu-manager ryu.app.simple_switch
```

Następnie należy go przetestować jako sterownik w emulatorze prostej sieci w Mininet. Uruchomienie prostej sieci testowej składającej się z dwóch hostów i switcha OpenFlow między nimi wraz z zewnętrznym sterownikiem odbywa się za pomocą komendy:

```
sudo mn --controller=remote --test pingall
```

Wyniki otrzymane za pomocą powyższej komendy po uprzednim uruchomieniu sterownika Ryu są widoczne na zdjęciu ekranu (Rys. 1). Jak widać na Rys. 1 widnieje komunikacja między switch'em a sterownikiem poprzez protokół OpenFlow. Napływające pakiety są notowane w sterowników z wykorzystaniem pakietów PACKET\_IN, PACKET\_OUT. Dokonując głębszej inspekcji pakietów (Rys. 4), można zauważyć, że w rzeczy samej wewnątrz protokołu OpenFlow znajduje się zenkapsulowany pakiet zmierzający z jednego hosta do drugiego (taki ruch został odnotowany w sterowniku Ryu i jest widoczny w konsoli na Rys. 3).

### 2.2 Obserwacja ruchu pakietów w programie Wireshark

Na maszynie wirtualnej został zainstalowany i uruchomiony program wireshark w celu dokładniejszej inspekcji pakietów. Jak widać na Rys. 3 widnieje komunikacja między switch'em a sterownikiem poprzez protokół OpenFlow. Napływające pakiety są notowane w sterowników z wykorzystaniem pakietów PACKET\_IN, PACKET\_OUT. Dokonując głębszej inspekcji pakietów (Rys. 4), można zauważyć, że w rzeczy samej wewnątrz protokołu OpenFlow znajduje się zenkapsulowany pakiet zmierzający z jednego hosta do drugiego (taki ruch został odnotowany w sterowniku Ryu i jest widoczny w konsoli na Rys. 3).

```

Activities Terminal mar 22 11:41
bartlomiej@bartlomiej-VirtualBox:~$ ./test_ryu.sh
bartlomiej@bartlomiej-VirtualBox:~$ mv test_ryu.sh test_ryu.txt
bartlomiej@bartlomiej-VirtualBox:~$ cat test_ryu.txt
sudo mn --controller=remote --test=pingall
bartlomiej@bartlomiej-VirtualBox:~$ #!/bin/bash
bartlomiej@bartlomiej-VirtualBox:~$ sudo mn --controller=remote --test=pingall
bartlomiej@bartlomiej-VirtualBox:~$ sudo mn --controller=remote --test=pingall
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
...
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.786 seconds
bartlomiej@bartlomiej-VirtualBox:~$ 

```

Rysunek 1: Mininet ping test ze sterownikiem Ryu.

```

Activities Terminal mar 28 15:28
bartlomiej@bartlomiej-VirtualBox:~$ netstat -ap | grep python
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:6633          0.0.0.0:*          LISTEN      5053/python3
tcp        0      0 0.0.0.0:6653          0.0.0.0:*          LISTEN      5053/python3
bartlomiej@bartlomiej-VirtualBox:~$ 

```

Rysunek 2: Sprawdzanie działania sterownika Ryu z wykorzystaniem narzędzia netstat.

No.	Time	Source	Destination	Protocol	Length	Info
10	0.189186716	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
14	0.195273813	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
15	0.195273813	127.0.0.1	127.0.0.1	OpenFlow	120	Type: OFPT_PORT_STATUS
16	0.299862329	127.0.0.1	127.0.0.1	OpenFlow	242	Type: OFPT_FEATURES_REPLY
38	0.602284113	:	:	OpenFlow	170	Type: OFPT_PACKET_IN
39	0.604468285	:	:	OpenFlow	176	Type: OFPT_PACKET_OUT
42	0.922211449	:	:	OpenFlow	174	Type: OFPT_PACKET_IN
43	0.924428712	:	:	OpenFlow	174	Type: OFPT_PACKET_OUT
44	0.924428712	:	:	OpenFlow	174	Type: OFPT_PACKET_IN
46	0.988017497	:	:	OpenFlow	180	Type: OFPT_PACKET_OUT
52	1.290156362	fe80::8880:49ff:feb. ffff:16	ffff:16	OpenFlow	174	Type: OFPT_PACKET_IN
53	1.290177897	fe80::8880:49ff:feb. ffff:2	ffff:2	OpenFlow	154	Type: OFPT_PACKET_IN
55	1.210353970	fe80::8880:49ff:feb. ffff:16	ffff:16	OpenFlow	154	Type: OFPT_PACKET_OUT
57	1.210353970	fe80::8880:49ff:feb. ffff:12	ffff:12	OpenFlow	169	Type: OFPT_PACKET_OUT
59	1.305131372	fe80::8880:49ff:feb. broadcast	broadcast	OpenFlow	146	Type: OFPT_PACKET_IN
68	1.306978283	1e:8a:b2:04:e5:ba	Broadcast	OpenFlow	132	Type: OFPT_PACKET_OUT
62	1.307764129	82:8b:49:b9:8c:3a	1e:8a:b2:04:e5:ba	OpenFlow	126	Type: OFPT_PACKET_IN
63	1.309246989	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
64	1.309246989	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_OUT
65	1.310949751	10:0:0:0:0:0	10:0:0:0:0:0	OpenFlow	152	Type: OFPT_PACKET_IN
67	1.31894578	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_FLOW_MOD
68	1.310949726	10.0.0.1	10.0.0.2	OpenFlow	108	Type: OFPT_PACKET_OUT
74	1.490163446	127.0.0.1	127.0.0.1	OpenFlow	127	Type: OFPT_PORT_STATUS
75	1.411709853	127.0.0.1	127.0.0.1	OpenFlow	154	Type: OFPT_PORT_STATUS
81	1.477252407	127.0.0.1	127.0.0.1	OpenFlow	154	Type: OFPT_FLOW_ADDED
82	1.477526883	127.0.0.1	127.0.0.1	OpenFlow	154	Type: OFPT_FLOW_REMOVED

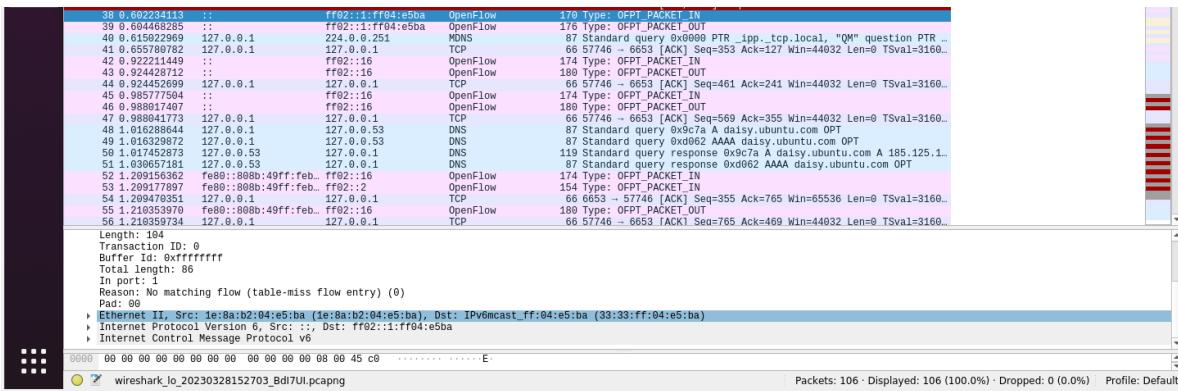
```

Activities Terminal mar 28 15:32
bartlomiej@bartlomiej-VirtualBox:~$ 
packet in 1 1e:8a:b2:04:e5:ba 33:33:ff:04:e5:ba 1
packet in 1 82:8b:49:b9:8c:3a 33:33:00:00:00:16 2
packet in 1 1e:8a:b2:04:e5:ba 33:33:00:00:00:16 1
packet in 1 82:8b:49:b9:8c:3a 33:33:00:00:00:16 2
packet in 1 82:8b:49:b9:8c:3a 33:33:00:00:00:02 2
packet in 1 1e:8a:b2:04:e5:ba ff:ff:ff:ff:ff:ff 1
packet in 1 82:8b:49:b9:8c:3a 1e:8a:b2:04:e5:ba 2
packet in 1 1e:8a:b2:04:e5:ba 82:8b:49:b9:8c:3a 2
port deleted 2
port deleted 1
bartlomiej@bartlomiej-VirtualBox:~$ 

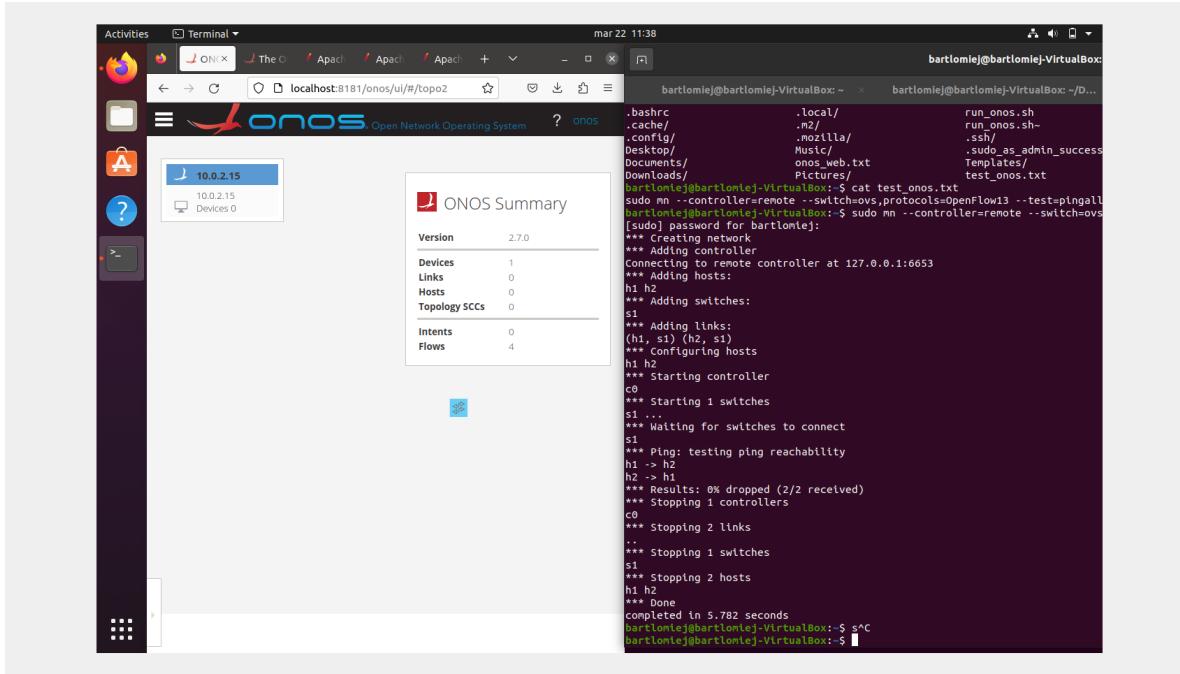
```

Packets: 106 · Displayed: 26 (24.5%) · Selected: 8 (7.5%) · Dropped: 0 (0.0%) · Profile: Default

Rysunek 3: Pakiety openflow zaobserwowane w programie wireshark - Ryu.



Rysunek 4: Głębsza analiza pakietu PACKET\_IN - Ryu.



Rysunek 5: Mininet ping test ze sterownikiem ONOS.

### 3 ONOS testing

Po instalacji środowiska zostały przeprowadzone następujące testy w celu weryfikacji poprawności instalacji:

1. Test automatyczny emulatora Mininet
2. Obserwacja ruchu pakietów w programie Wireshark

#### 3.1 Test automatyczny emulatora Mininet

W celu użycia sterownika ONOS należy najpierw uruchomić service ONOS. Dla lokalizacji plików ONOS na masznie wirtualnej wykorzystanej w tym warsztacie była użyta następująca komenda:

```
/opt/onos/bin/onos.service start
```

Następnie został uruchominy sterownik ONOS w formie graficznej w przeglądarce poprzez wejście na adres:

```
http://localhost:8181/onos/ui/
```

Podobnie jak w wypadku sterownika Ryu należy go przetestować w emulatorze prostej sieci w Mininet. Uruchomienie prostej sieci testowej składającej się z dwóch hostów i switcha OpenFlow między nimi wraz ze sterownikiem ONOS odbywa się za pomocą komendy:

```
sudo mn --controller=remote --switch=ovs,protocols=OpenFlow13 --test pingall
```

Wyniki otrzymane za pomocą powyższej komendy po uprzednim uruchomieniu sterownika ONOS są widoczne na zdjęciu ekranu (Rys. 5). Test w konsoli wskazuje, że pakiety między hostem 1 a hostem 2 zostały prawidłowo przekazane. Jak widać, w aplikacji sterwonika znajduje się obraz switch'a, oraz można zwrócić uwagę na widoczną wartość Flow równą 4.

Otwierając w aplikacji sterownika tablicę przepływów (Rys.6) widać jakie zostały odnotowane protokoły i dostosowane przez sterownik metody ich przetwarzania przez switch.

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	0	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	0	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	0	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	0	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core

Rysunek 6: Flow table dostępna na sterowniku ONOS po uruchomieniu testu.

### 3.2 Obserwacja ruchu pakietów w programie Wireshark

Dokonując inspekcji pakietów w programie wireshark można zauważyc ruch znacznie różniacy się względem testu ze sterownikiem Ryu. Jedną z nich jest wersja protokołu OpenFlow: 1.3 (Rys.7). Na Rys. 8 widać, komunikację między switch'em a sterownikiem w celu poprawnego obsługi pakietów (PACKET\_IN, PACKET\_OUT).

### 3.3 Wyłączanie automatycznego przekazywania pakietów

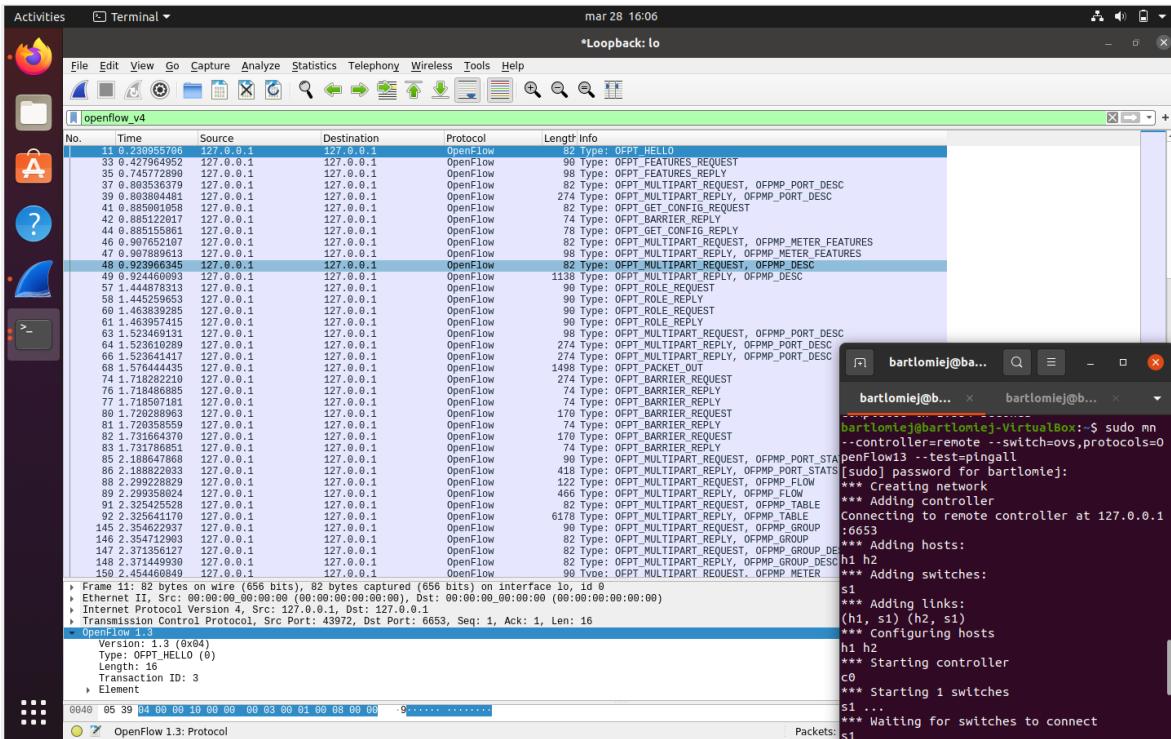
W celu sprawdzenia czy sterowniki ONOS działa, w aplikacji sterownika wyłączono automatyczne przekazywanie pakietów (Rys.9). Jak można zauważyc (Rys. 10) podczas testu w konsoli widać, że pakiety nie przechodzą między hostem 1, a hostem 2. Jednakże, obserwując pakiety widać, że pakiety wpływają do switch'a i są przekazywane do sterownika, lecz ze względu na brak automatycznego przekazywania pakietów, to nie są one przekazywane dalej.

## 4 Budowa sieci

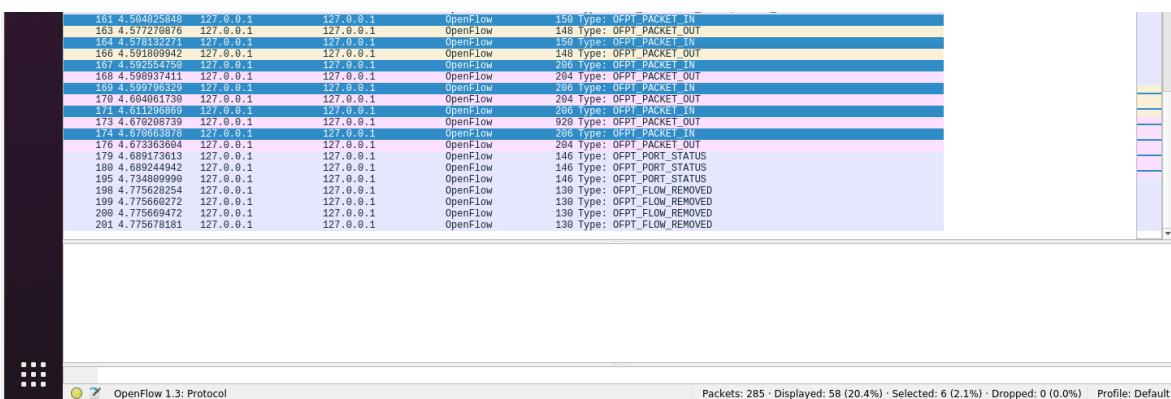
Celem warsztatu było przygotowanie zadanej topologii (Fig. 11) w sieci mininet zautomatyzowanego za pomocą skryptu w języku Python. Następnie przygotowanie jej do działania z wybranym sterownikiem.

Zgodnie ze wskazówką, skrypt był tworzony warstwo. Najpierw zostało utworzone połączenie między dwoma hostami (wraz ze switchem między nimi). Następnie topologia była sukcesywnie rozbudowywana, aż do osiągnięcia pełnego kształtu. Pełny kod można zobaczyć na listingu 1. Istotną sprawą są adresy IP. Zaprezentowane na listingu hosty znajdują się w różnych podsieciach. Dodatkowo został przygotowany prosty skrypt uruchamiający sieć mininet wraz z własną topologią 2.

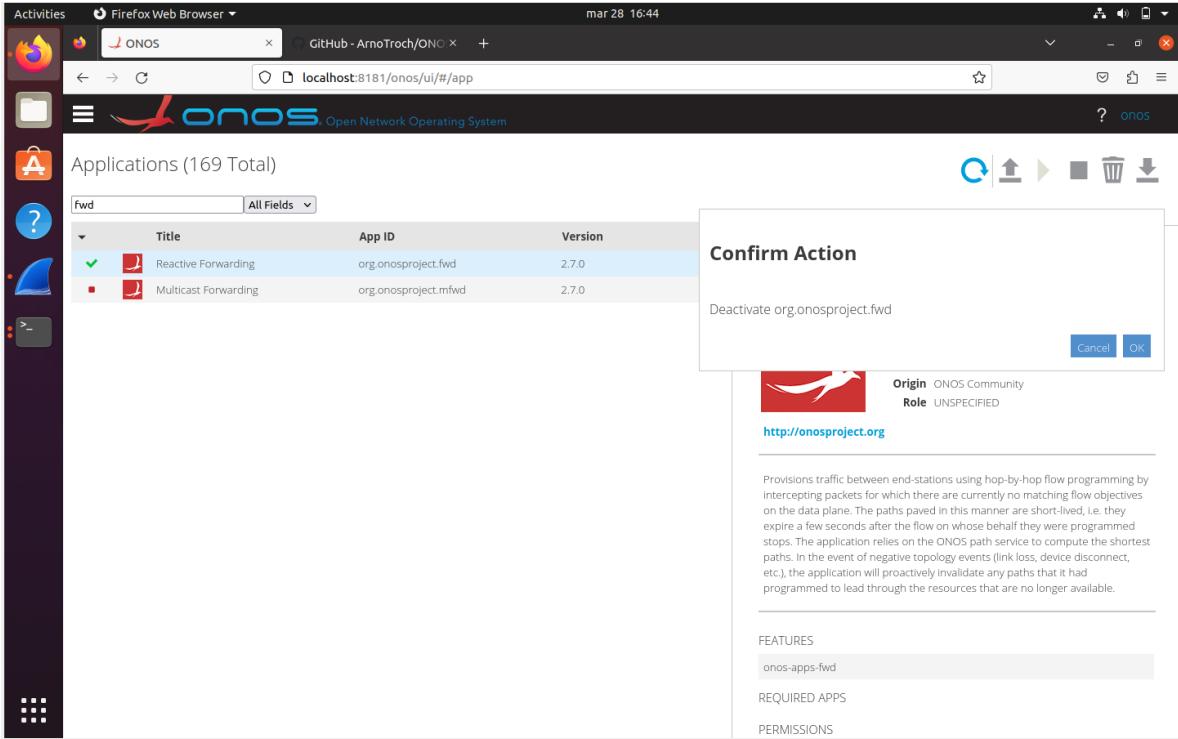
Dla drugiego wariantu zadania konieczne było przypisanie hostów do dwóch podsieci. Przypisane adresy ip i mac do hostów można zobaczyć dla jednej podsieci w Tab. 1, a dla osobnych podsieci w Tab. 2. Po uruchomieniu sieci mininet należy ręcznie zdefiniować trasowanie. Zostało to przedstawione w dwóch ostatnich (wykomentowanych) linijkach Lst. 1. Na rzecz tego warsztatu została podjęta taka decyzja ze względu na debbugning w trybie interaktywnym sieci mininet. Jest to oczywiście możliwe



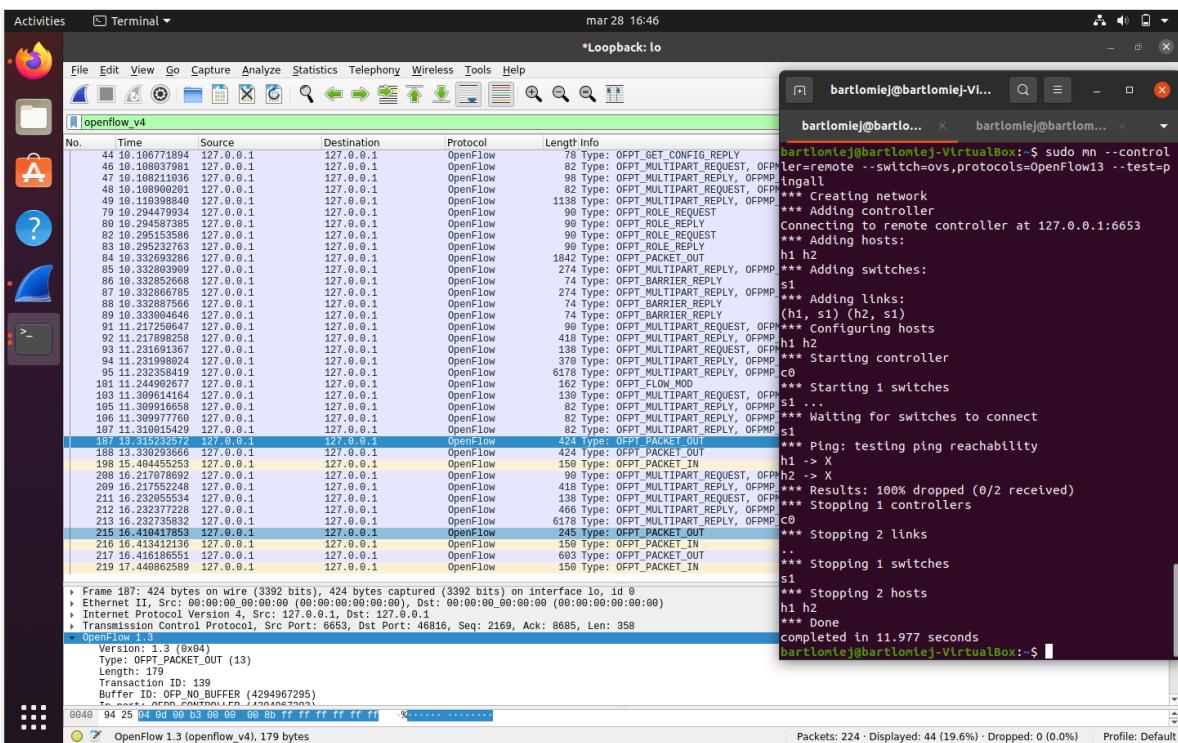
Rysunek 7: Pakiety openflow zaobserwowane w programie wireshark - ONOS.



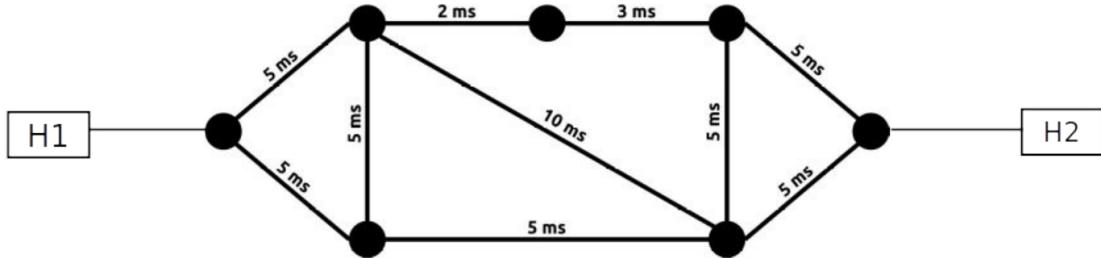
Rysunek 8: Napływ pakietów (PACKET\_IN, PACKET\_OUT) - ONOS.



Rysunek 9: Wyłączenie automatycznego przekazywania pakietów - ONOS.



Rysunek 10: Test braku automatycznego przekazywania pakietów - ONOS.



Rysunek 11: Docelowa topologia sieci.

do pełnego zautomatyzowania, lecz w trakcie rozmowy z Prowadzącym nie było to wskazane jako konieczność.

Kod zaprezentowany na Listingu 1 jest bardzo przejrzysty. Wykorzystując paradigmat programowania obiektowego budowa topologii jest zaskakująco przejrzysta. Każdy element sieci: host, switch i odpowiednie połączenia dodaje się wykorzystując odpowiednie metody odziedziczone po klasie Topo. Połączenia mogą być sparametryzowane. Zostało to wykorzystane w celu ustanowienia wymaganych opóźnień. W ostatniej niewykommentowanej linijce widać wyrażenie lambda służące do stworzenia klasy MyTopo przy uruchomieniu skryptu uruchomieniowego w celu stworzenia danej topologii .

Host	adres ip/maska	mac address
Host1	10.0.0.1/30	d2 : a2 : bf : 68 : a9 : 5f
Host2	10.0.0.2/30	e6 : 49 : 69 : 48 : e4 : c1

Tabela 1: Przypisane adresy ip i MAC do hostów - jedna podsieć.

Host	adres ip/maska	mac address
Host1	10.0.0.1/24	d2 : a2 : bf : 68 : a9 : 5f
Host2	10.0.1.1/24	e6 : 49 : 69 : 48 : e4 : c1

Tabela 2: Przypisane adresy ip i MAC do hostów - dwie podsieci.

Listing 1: Budowa sieci za pomocą skryptu.

```
from mininet.topo import Topo
from mininet.link import TCLink

from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.cli import CLI

class MyTopo(Topo):
    def __init__(self):
        Topo.__init__(self)

        Host1 = self.addHost('h1', ip='10.0.0.1/24', mac='d2:a2:bf:68:a9:5f')
        Host2 = self.addHost('h2', ip='10.0.1.1/24', mac='e6:49:69:48:e4:c1')

        Switch1 = self.addSwitch('s1')
        Switch2 = self.addSwitch('s2')
        Switch3 = self.addSwitch('s3')
        Switch4 = self.addSwitch('s4')
        Switch5 = self.addSwitch('s5')
        Switch6 = self.addSwitch('s6')
        Switch7 = self.addSwitch('s7')

        Host1 -> Switch1
        Host2 -> Switch1
        Host1 -> Switch2
        Host2 -> Switch2
        Host1 -> Switch3
        Host2 -> Switch3
        Host1 -> Switch4
        Host2 -> Switch4
        Host1 -> Switch5
        Host2 -> Switch5
        Host1 -> Switch6
        Host2 -> Switch6
        Host1 -> Switch7
        Host2 -> Switch7
```

```

    self.addLink(Host1, Switch1)
    self.addLink(Switch1, Switch2, delay='5ms')
    self.addLink(Switch1, Switch3, delay='5ms')
    self.addLink(Switch2, Switch3, delay='5ms')
    self.addLink(Switch2, Switch4, delay='2ms')
    self.addLink(Switch2, Switch6, delay='10ms')
    self.addLink(Switch3, Switch6, delay='5ms')
    self.addLink(Switch4, Switch5, delay='3ms')
    self.addLink(Switch5, Switch7, delay='5ms')
    self.addLink(Switch6, Switch7, delay='5ms')
    self.addLink(Switch5, Switch6, delay='5ms')
    self.addLink(Host2, Switch7)

topos = { 'mytopo' : ( lambda: MyTopo() ) }
#h1 ip route add 10.0.1.0/24 via 10.0.0.1 dev h1-eth0
#h2 ip route add 10.0.0.0/24 via 10.0.1.1 dev h2-eth0

```

Listing 2: Skrypt uruchamiający topologię.

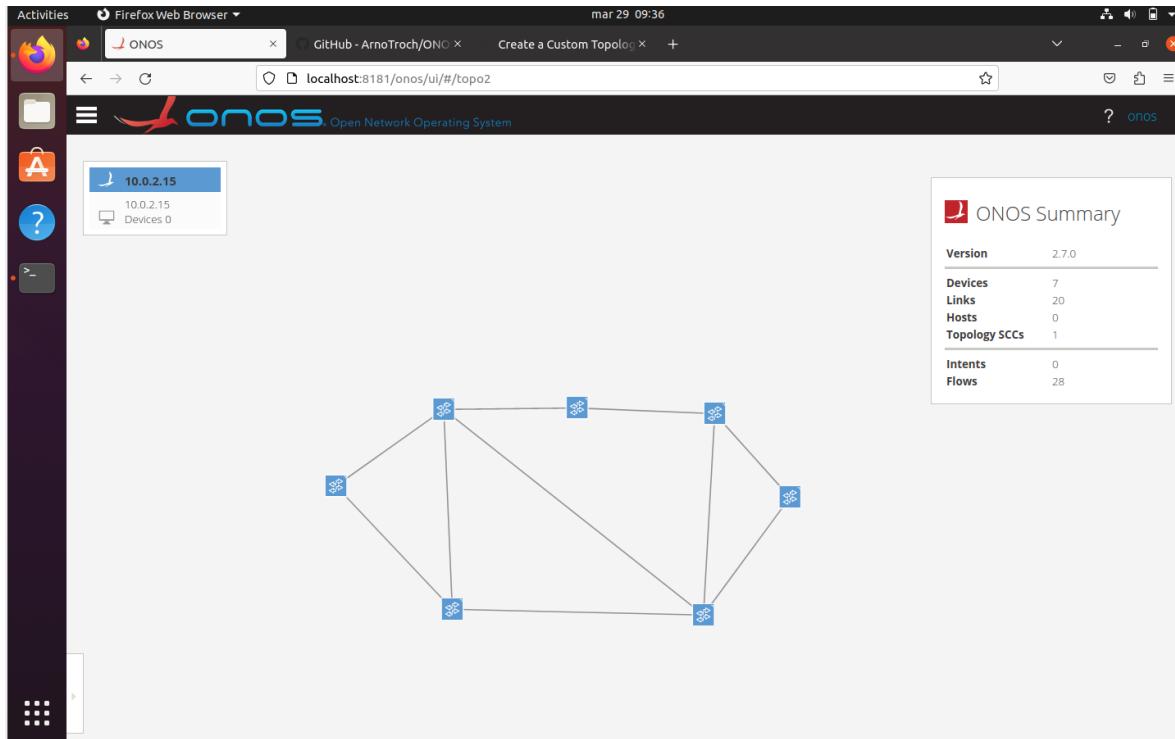
```

#!/bin/bash
path=$(pwd)
sudo mn --custom $path/warsztat3.py --switch=ovs,protocols=OpenFlow13 --topo=mytopo
--controller=remote --link tc

```

## 5 ONOS

W celu sprawdzenia poprawności topologii został wykorzystany sterownik - ONOS. Dzięki możliwości zobaczenia graficznej reprezentacji sieci (Fig. 12) została wstępnie potwierdzona poprawnie zbudowana topologia sieci.



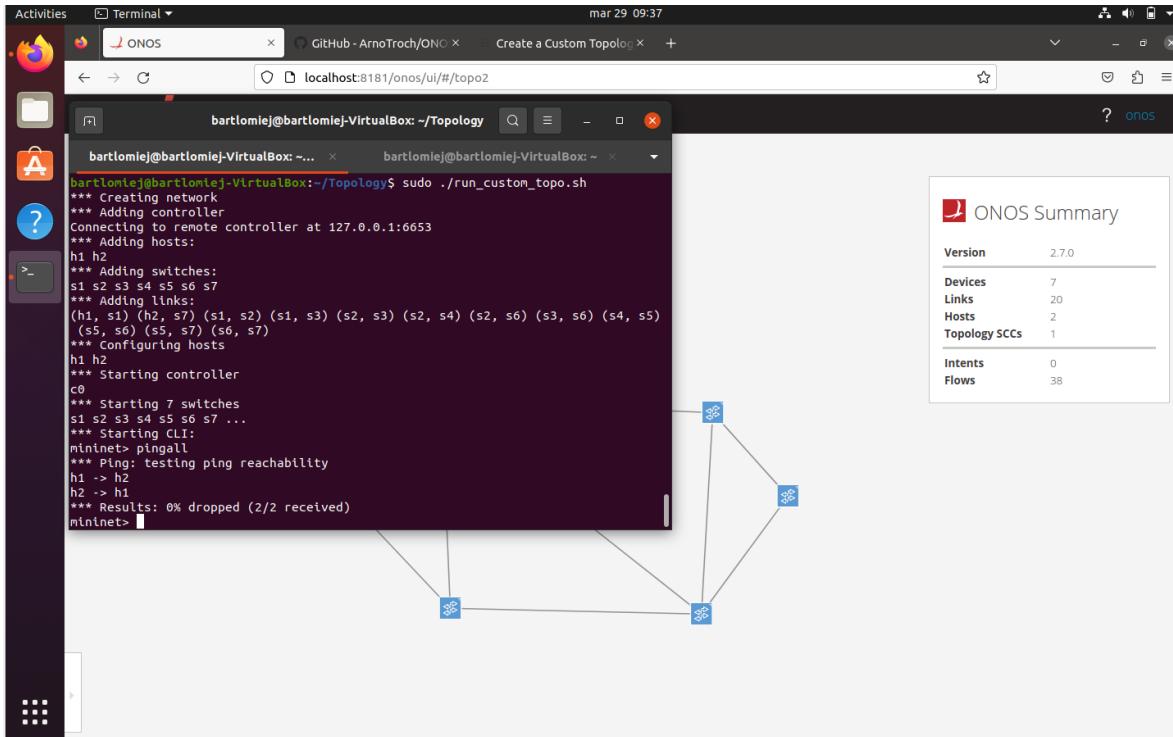
Rysunek 12: Topologia sieci - ONOS.

### 5.1 Wspólna podsieć

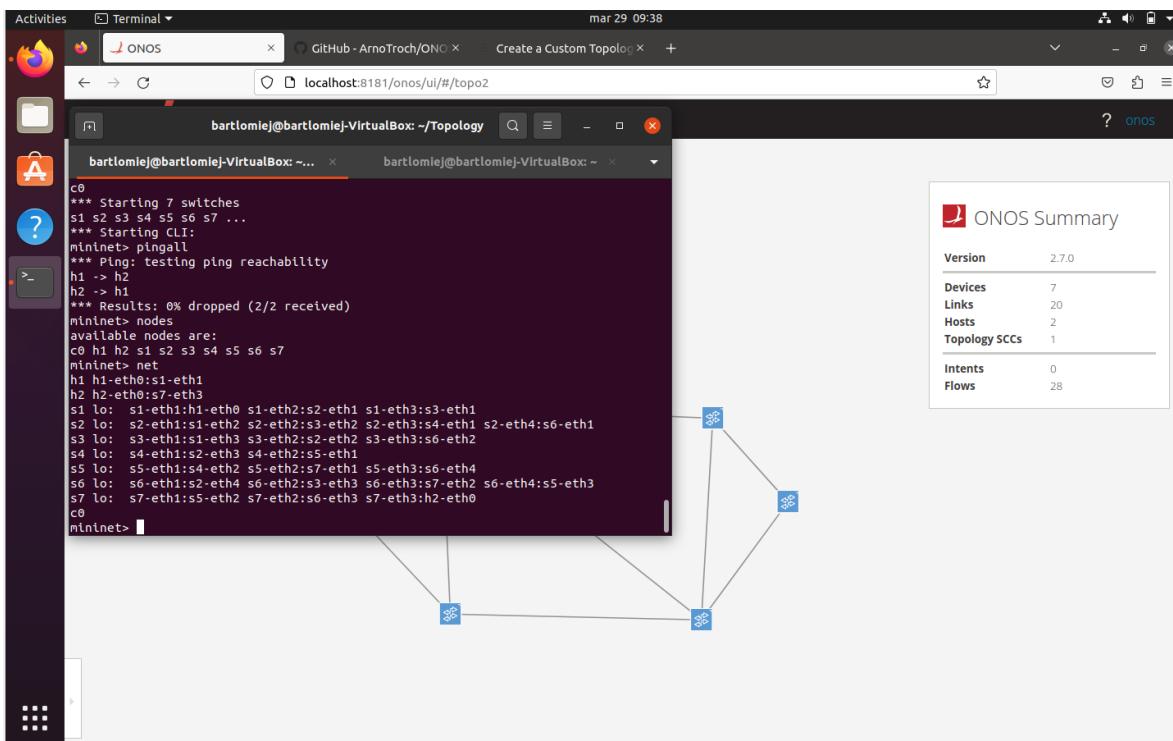
Następnie po uruchomieniu sieć mininet w trybie testowym (*pingall*) uzyskano potwierdzenie poprawności topologii (Fig.13). Co więcej, podczas testu poprawnie odbył się przepływ pakietów między hostami. Podobnież topologia została sprawdzona wewnętrz sieci mininet (Fig. 14). Nie była konieczna instalacja dodatkowych aplikacji względem przedstawionych w poprzednim warsztacie.

### 5.2 Różne podsieci

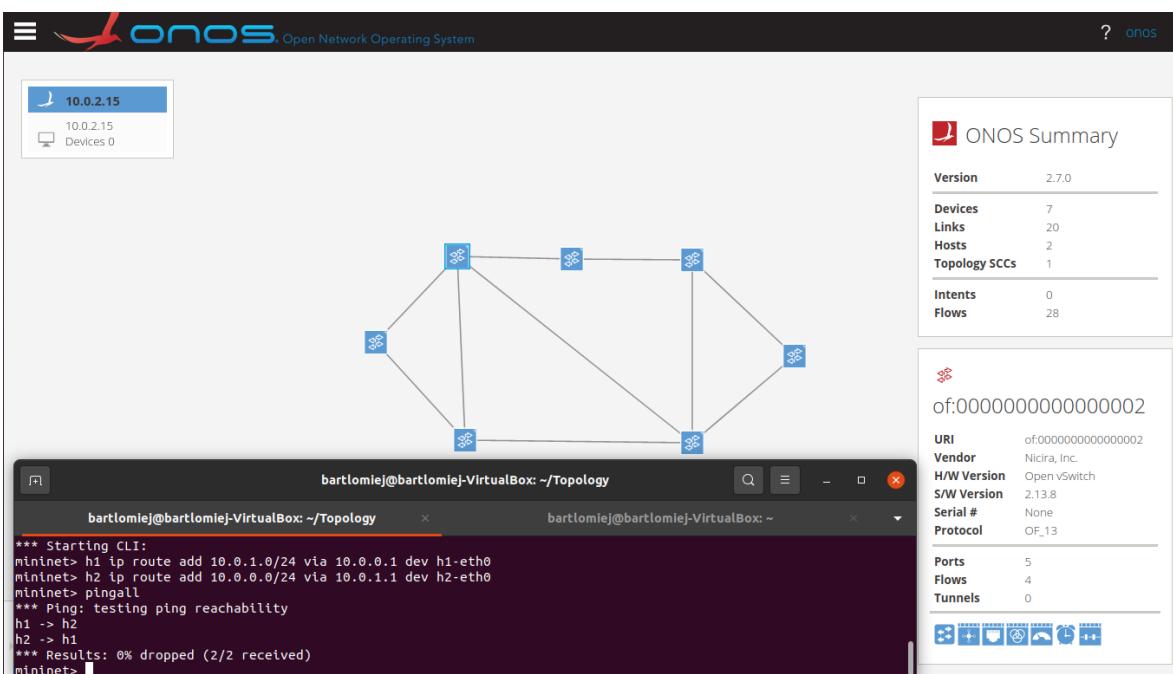
Jak zostało poruszone przy budowie sieci, hosty znajdowały się w osobnych podsieciach. Zatem konieczne było przygotowanie routingu w sieci mininet na hostach, jak zostało wcześniej przedstawione. Wprowadzenie routingu oraz test poprawności działania można zobaczyć na Fig. 15.



Rysunek 13: Test sieci - mininet.



Rysunek 14: Topologia sieci - mininet.



Rysunek 15: Test działania w dwóch podsieciach.

## 6 Ryu

Mimo, że celem zadania było wykonania zadania z użyciem jednego sterownika, w celach edukacyjnych został również przedstawiony test również z użyciem sterownika Ryu. Istotną uwagę przykuł fakt, że wystąpił błąd korzystając z użyciem domyślnej wersji protokołu OpenFlow oraz switcha STP, który uniemożliwił uruchomienie sterownika. Problemy nie występowały natomiast przy wykorzystaniu wersji protokołu OpenFlow 1.3 oraz aplikacji switch'a *stp\_13*.

Konieczność wykorzystania aplikacji switcha wykorzystującej protokół STP (spanning tree protocol) w celu uniknięcia nieskończonych pętli ramek. Uruchomienie sterownika jest widoczne na Fig. 16. Wykorzystanie protokołu STP niesie ze sobą pewne konsekwencję - musi się wykonać co zajmuje czas. Z tego powodu, niezależnie czy hosty są w tej samej podsieci czy też w osobnych to początkowo nie ma między nimi przepływu danych. Zostało to zaprezentowane na Fig.17.

Następnie zostało przetestowane połączenie między hostami po uprzednim oczekaniu chwili 18. Jak widać pakiety docierały bez problemu. Dodatkowo, tablica arp oraz routingu została zaprezentowana na Fig. 19 w celu pokazania poprawności konfiguracji routingu.

```
bartlomiej@bartlomiej-VirtualBox:~$ ryu-manager ryu.app.simple_switch_stp_13
loading app ryu.app.simple_switch_stp_13
loading app ryu.controller.ofp_handler
instantiating app None of Stp
creating context stplib
instantiating app ryu.app.simple_switch_stp_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
[STP][INFO] dpid=0000000000000006: Join as stp bridge.
[STP][INFO] dpid=0000000000000006: [port=1] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000006: [port=4] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000006: [port=2] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000005: Join as stp bridge.
[STP][INFO] dpid=0000000000000005: [port=3] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000005: [port=1] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000005: [port=2] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000007: Join as stp bridge.
[STP][INFO] dpid=0000000000000005: [port=3] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000007: [port=1] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000007: [port=2] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000001: Join as stp bridge.
[STP][INFO] dpid=0000000000000007: [port=3] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT / LISTEN
```

Rysunek 16: Uruchomienie sterownika Ryu.

## 7 Kierowanie na podstawie adresu IP

Zgodnie instrukcją zostało zrealizowane. Zarówno TTL jak i aktualizacja sumy kontrolnej. Dostęp w kodzie źródłowym. Uruchomienie i komplikacja - jak w warsztacie 5.

## 8 Filtrowanie

Filtrowanie pakietów miało odbywać się poprzez

1. Adres docelowy IP
2. Port docelowy warstwy transportowej
3. Protokół warstwy transportowej

niestety w celu zachowania ruchu sieciowego trzeba było ograniczyć się do adresu docelowego IP. Wynikało to bezpośrednio z błędu dopasowania (ternary/exact) z nieznanych przyczyn (ani dopasowanie exact na dokładną wartość, ani ternary 0&&&0 (wszystkie wartości) nie działały). Prawdopodobnie problem jest niezależny i wynika z błędu kompilatora (wartości portów i protokołu były sprawdzane w wireshark'u oraz w tablicach (table\_show <table name>)).

Rysunek 17: Test połączenia między hostami - oczekiwanie na STP.

```
hi h2
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ... (5ms delay) (5ms delay) (5ms delay) (5ms delay) (2ms delay)
(10ms delay) (5ms delay) (5ms delay) (5ms delay) (2ms delay) (3ms delay) (3ms delay)
(5ms delay) (5ms delay) (10ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay)
(5ms delay) (5ms delay)
*** Starting CLI:
mininet> h1 ip route add 10.0.1.0/24 via 10.0.0.1 dev h1-eth0
mininet> h2 ip route add 10.0.0.0/24 via 10.0.1.1 dev h2-eth0
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

Rysunek 18: Test połączenia między hostami.

Testowanie odbywało się poprzez uruchomienie prostego serwera http w pythonie na jednym host'ie, a następnie próba łączenia się z nim z drugiego hosta. Dodatkowo ruch był obserwowany w wireshark'u. Załączam wyniki testów dla filtrowania adresów IP.

Dodatkowo załączam tabele do filtrowania adresów IP (działające) oraz całej reszty (niedziałające):

## 9 Statystyki

W celu zebrania statystyk zostały użyte liczniki (counters). W celu zebrania dany trzeba w trakcie działania programu wpisać następujące komendy (wewnątrz simple\_switch\_CLI:

RuntimeCmd: counter\_read <counter name> <port id>

```

mininet> h1 arp
Address          HWtype  HWaddress          Flags Mask      Iface
10.0.1.1        ether    e6:49:69:48:e4:c1  C          h1-eth0
mininet> h2 arp
Address          HWtype  HWaddress          Flags Mask      Iface
10.0.0.1        ether    d2:a2:bf:68:a9:5f  C          h2-eth0
mininet> h1 netstat -nr
Kernel IP routing table
Destination      Gateway      Genmask      Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0    255.255.255.0 U        0 0          0 h1-eth0
10.0.1.0        10.0.0.1   255.255.255.0 UG       0 0          0 h1-eth0
mininet> h2 netstat -nr
Kernel IP routing table
Destination      Gateway      Genmask      Flags   MSS Window irtt Iface
10.0.0.0        10.0.1.1   255.255.255.0 UG       0 0          0 h2-eth0
10.0.1.0        0.0.0.0    255.255.255.0 U        0 0          0 h2-eth0
mininet>

```

Rysunek 19: Tablica arp oraz routingu.

```

Starting CLI...
mininet> h2 python3 -m SimpleHTTPServer 80 &
mininet> h1 wget -O - h2
--2023-05-17 23:39:20--  http://10.0.1.10/
Connecting to 10.0.1.10:80... failed: Connection refused.
mininet>

```

Rysunek 20: Działania testowe - jest połaczenie (mimo, że "refused").

Wyniki zostały zebrane na trzech licznikach:

1. Odebrane/wysłane pakiety - dla każdego portu - zostało zrealizowane wewnątrz tabeli odpowiadającej za filtrowanie ruchu (jeżeli pakiet został przekazany do wysłania dalej to licznik się zwiększał - Ingress).
2. Przesłane pakiety dalej - dla każdego portu - została stworzona tabela w celu zbierania statystyk na portach wychodzących (Egress).
3. Odrzucone pakiety - zbiorczo dla wszystkich portów - wewnątrz Ingress - jeżeli pakiet jest odrzucony to zwiększały licznik.

## 10 Podsumowanie

Udało zrealizować się podstawową funkcjonalność routera IP. Niestety z nieznanych przyczyn nie udało się zrobić dokładnego dopasowania. Prawdopodobnie błąd wynika z przyczyn niezależnych - nawet korzystając z dopasowania ternary: 0&&&0 (dopasuj wszystko) pakiety były odrzucane. Nie wynikało to z błędu nagłówka, lecz jedynie z dopasowania.

Pozostałe funkcjonalności zostały zrealizowane. Jedyny problem jaki wystąpił był z odczytaniem wartości direct\_counter - nie znalazłem w dokumentacji takich danych.

## 11 Instalacja środowiska

Ze względu na błąd Advanced Packaging Tool (apt) na Ubuntu (wskazywał nieaktualny klucz w repozytorium P4 konieczna była instalacja ze źródeł zgodnie z instrukcją <https://github.com/p4lang/p4c>

1	0.000000000	10.0.0.10	10.0.1.10	TCP	76 42528 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 T...
2	0.000865480	10.0.0.10	10.0.1.10	TCP	76 [TCP Out-Of-Order] 42528 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=...
3	0.000920504	10.0.1.10	10.0.0.10	TCP	56 80 → 42528 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4	0.001508993	10.0.1.10	10.0.0.10	TCP	56 80 → 42528 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Rysunek 21: Działania testowe - ruch między hostami.

```
table_set_default cIngress.traffic_filter drop
table_add cIngress.traffic_filter ipv4_forward 10.0.1.10&&&1.1.1.1 => 00:04:00:00:00:01 2 1
table_add cIngress.traffic_filter ipv4_forward 10.0.0.10&&&1.1.1.1 => 00:04:00:00:00:00 1 2

- : - - cmd_v3          All L1      (Fundamental)
```

Rysunek 22: Tabela do filtrowania adresów IP

```
table_set_default cIngress.traffic_filter drop
table_add cIngress.traffic_filter ipv4_forward 10.0.1.10&&&1.1.1.1 80 6 => 00:04:00:00:00:01 2 2
table_add cIngress.traffic_filter ipv4_forward 10.0.0.10&&&1.1.1.1 80 6 => 00:04:00:00:00:00 1 1
```

Rysunek 23: Tabela do pełnego filtrowania

```
prinv2@prinv2-VirtualBox:~/w6$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read count_parsed 1
count_parsed[1]= (74 bytes, 1 packets)
RuntimeCmd: counter_read count_parsed 2
count_parsed[2]= (54 bytes, 1 packets)
RuntimeCmd: counter_read count_parsed 0
count_parsed[0]= (0 bytes, 0 packets)
RuntimeCmd: counter_read count_dropped 0
count_dropped[0]= (0 bytes, 0 packets)
RuntimeCmd: █
```

Rysunek 24: Odebrane/wysłane pakiety

```
prinv2@prinv2-VirtualBox:~/w6$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read count_sent
Error: Wrong number of args, expected 2 but got 1
RuntimeCmd: counter_read count_sent 0
count_sent[0]= (128 bytes, 2 packets)
RuntimeCmd: counter_read count_sent 1
count_sent[1]= (0 bytes, 0 packets)
RuntimeCmd: counter_read count_sent 2
count_sent[2]= (0 bytes, 0 packets)
RuntimeCmd: counter_read count_sent 3
count_sent[3]= (0 bytes, 0 packets)
RuntimeCmd:
```

Rysunek 25: Przesłane pakiety dalej

```
prinv2@prinv2-VirtualBox:~/w6$ simple_switch_CLI
Obtaining JSON from switch...
Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read count_dropped 0
this is the direct counter for table cIngress.traffic_filter
Invalid table operation (INVALID_HANDLE)
RuntimeCmd: █
```

Rysunek 26: Odrzucone pakiety

```
prinv2@prinv2-VirtualBox:~/w5$ cat compile.sh
#!/bin/bash
p4c-bm2-ss --std p4-16 --target bmv2 --arch v1model -o ./build/solution.json ./src/solution.p4
```

Rysunek 27: Kompilacja programu P4 pod behavioral model 2.

```
prinv2@prinv2-VirtualBox:~/w5$ cat run.sh
#!/bin/bash
sudo python3 mininet_model.py --behavioral-exe /usr/bin/simple_switch --json build/solution.json
```

Rysunek 28: Uruchomienie środowiska mininet z wgraną konfiguracją na switch'a.

- rozdział "Installing p4c from source". W celu uruchomienia switch'a w programie mininet koniecznym było pobranie skryptów: *mininet\_model.py* oraz *netstat.py*.

## 12 Uruchomienie programu P4

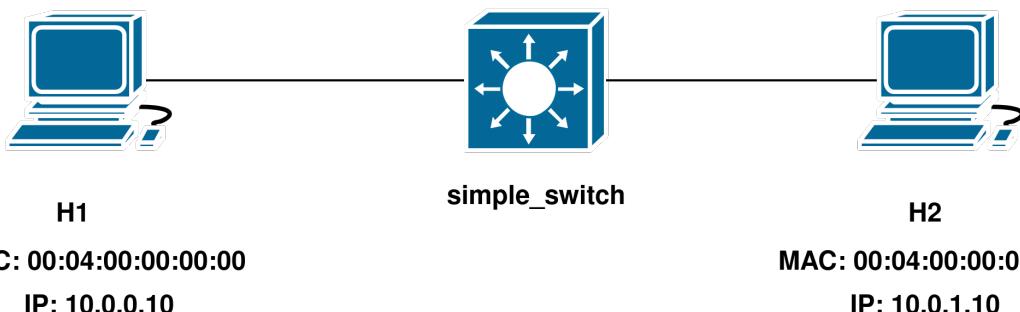
W celu skompilowania programu P4 na model switch'a typu behavioral model należy to wskazać w lini poleceń. Podobnie wersję języka P4 jak i architekturę (v1model). W celu zwiększenia automatyzacji został stworzony skrypt *compile.sh* (Fig. 27). Następnie w celu uruchomienia programu został napisany skrypt *run.sh*, który uruchamia skrypt *mininet\_model.py* podając w parametrach behavioral model switch jak i plik .json posiadający skompilowaną konfigurację (Fig.28). Skrypt *mininet\_model.py* tworzy następującą topologię w sieci mininet (Fig. 29).

Behavioral Model v2 umożliwia dostęp do switch'a w czasie działania programu. Można go uruchomić poprzez komendę *simple\_switch\_CLI*. W celu prostszego wgrywania tablic do switch'a został napisany prosty skrypt *table\_rules.sh* (Fig. 30).

## 13 Przekazywanie pakietów między interface'ami

Podstawowym zadaniem było przekazywanie pakietów między interfejsami. W tym celu została dodana tablica w bloku ingress Fig. 31. Nazwa tablicy: *vlan\_parse\_exact* wynika z dalszej pracy nad programem, niemniej nie zmienia funkcjonalności. W celu zadziałania przekazywania pakietów, po uruchomieniu programu (Fig. 28) należy wgrać konkretne wpisy do tablicy przekazywania pakietów. Zasady mające na celu zapewnienie podstawową funkcjonalność - przekazywanie pakietów między interfejsami są widoczne na Fig. 32. Jak można zauważyć podawany jest MAC adres hosta wysyłającego pakiet (odpowiada to polu klucza w tablicy w programie P4), a następnie MAC adres hosta odbierającego oraz port, na którym należy go przekazać (parametry akcji *vlan\_parse\_exact*).

Poprawność działania została sprawdzona poprzez testowanie połączenia za pomocą komendy ping. Wyniki były obserwowane również w programie wireshark (Fig. 33). Została uzyskana poprawna wymiana pakietów między hostami.



Rysunek 29: Utworzona topologia w sieci mininet.

```
prinv2@prinv2-VirtualBox:~/w5$ cat table_rules.sh
#!/bin/bash
simple_switch_CLI --thrift-port 9090 < cmd
```

Rysunek 30: Automatyzacja wpisywania zasad do tablic simple\_switch \_CLI.

```
- action drop(){
    mark_to_drop(stdmeta);
}

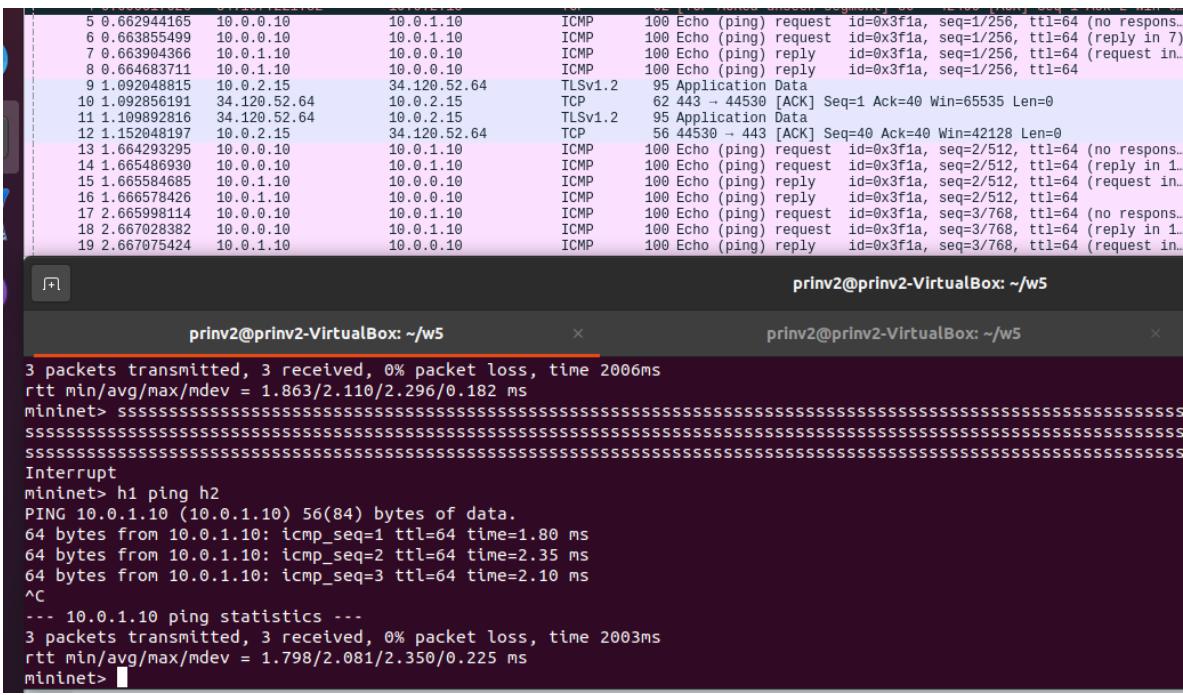
action mac_forward(macAddr_t dstAddr, egressSpec_t port){
    stdmeta.egress_spec = port;
    meta.port = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
}

table vlan_parse_exact{
    key = {
        hdr.ethernet.srcAddr: exact;
    }
    actions = {
        mac_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

Rysunek 31: Implementacja przekazywania pakietów między interfejsami.

```
table_set_default cIngress.vlan_parse_exact drop
table_add cIngress.vlan_parse_exact mac_forward 00:04:00:00:00:00 => 00:04:00:00:00:01 2
table_add cIngress.vlan_parse_exact mac_forward 00:04:00:00:00:01 => 00:04:00:00:00:00 1
```

Rysunek 32: Wpisy do tablicy przekazywania pakietów.



Rysunek 33: Test poprawnego przekazywania pakietów.

## 14 Dodawanie/usuwanie VLAN tag

Dodawanie/usuwanie VLAN tag'u odbywa się na konkretnych portach. Idea była taka:

1. INGRESS: Jeśli nie ma VLAN tag'u → dodaj VLAN tag.
2. INGRESS: Jeśli jest - przekaż pakiet na port egress.
3. EGRESS: Usuń VLAN tag na danym porcie lub nic nie rób.

Z tego powodu zostały stworzone trzy tablice reguł - dwie w bloku ingress i jedna w egress.

Przekazywanie pakietu jest takie samo jak w bloku wcześniejszym, natomiast dodawanie VLAN tag'u jest widoczne w tablicy *add\_vlanTag\_exact* (Fig. 34). Koniecznym było dodanie wyboru tablicy w zależności od posiadania VLAN tag (Fig. 35). W bloku egress VLAN tag jest usuwany na wskazanym porcie o ile występuje (Fig. 36).

Zostały wykonane następujące testy:

1. Usuwanie VLAN tag'u na jednym porcie (Fig. 37, Fig. 38).
2. Usuwanie VLAN tag'u na wszystkich portach (Fig. 39, Fig. 40).

Jak można zauważyć zadanie zostało wykonane poprawnie - na wskazanych portach jest dodawany jak i usuwany VLAN tag.

## 15 Ograniczenia przepływności

W celu zapewnienia funkcjonalności ograniczenia przepływności został wykorzystany extern meter. W celu zwiększenia przejrzystości raportu cały kod znajduje się w załączniku, natomiast w raporcie zostaną pokazane jedynie najważniejsze zmiany.

Meter'y działają na zasadzie Dual Token Bucket (RFC 2698), a zatem oznaczają pakiety na jeden z trzech kolorów: zielony, żółty lub czerwony. W implementacji koniecznym jest stworzenie zmiennej o wielkości conajmniej 2 bity (3 kolory), która będzie trzymała oznaczenie danego koloru. Zostało to zaimplementowane poprzez zdefiniowanie nowego typu 2 bitowego i dołączenie go do bloku metadata (Fig. 41).

```

control cIngress(inout headers hdr, inout metadata meta, inout standard_metadata_t stdmeta){
    action drop(){
        mark_to_drop(stdmeta);
    }

    action mac_forward(macAddr_t dstAddr, egressSpec_t port){
        stdmeta.egress_spec = port;
        meta.port = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
    }

    table vlan_parse_exact{
        key = {
            hdr.ethernet.srcAddr: exact;
        }
        actions = {
            mac_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
}

action add_vlanTag(macAddr_t dstAddr, egressSpec_t port, bit<12> vid){
    stdmeta.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.vlanTag.tpid = TYPE_8021Q;
    hdr.vlanTag.vid = vid;
    hdr.vlanTag.setValid();
    hdr.ethernet.etherType = TYPE_8021Q;
}

table add_vlanTag_exact{
    key = {
        hdr.ethernet.srcAddr: exact;
    }

    actions = {
        add_vlanTag;
        drop;
        NoAction;
    }
    default_action = NoAction();
}

```

Rysunek 34: Tablice wewnętrz bloku ingress.

```

apply{
    if(hdr.ipv4.isValid() && !hdr.vlanTag.isValid()){
        add_vlanTag_exact.apply();

    }
    else if(hdr.vlanTag.isValid()){
        vlan_parse_exact.apply();
    }
}

```

Rysunek 35: Wybór tablicy.

```

control cEgress(inout headers hdr, inout metadata meta, inout standard_metadata_t stdmeta){
    action remove_vlanTag(){
        hdr.vlanTag.setInvalid();
        hdr.etherType = TYPE_IPV4;
    }

    action remove(){
        hdr.vlanTag.setInvalid();
        hdr.etherType = TYPE_IPV4;
    }

    table remove_vlanTag_exact{
        key = {
            meta.port: exact;
        }

        actions = {
            remove_vlanTag;
            remove;
            NoAction;
        }
        default_action = NoAction();
    }
    apply{
        if(hdr.vlanTag.isValid()){
            remove_vlanTag_exact.apply();
        }
    }
}

```

Rysunek 36: Blok egress.

	Time	Source	Destination	Protocol	Length	Details
1	0.000000000	10.0.1.10	10.0.1.10	ICMP	100	Echo (ping) request id=0x4040, seq=1/256, ttl=64 (no respons...
2	0.000752501	00:aa:bb:00:00:00		0x4000	96	PRI: 4 DEI: 1 ID: 2383
3	1.016504078	10.0.0.10	10.0.1.10	ICMP	100	Echo (ping) request id=0x4040, seq=2/512, ttl=64 (no respons...
4	1.010760037	00:aa:bb:00:00:00		0x4000	96	PRI: 4 DEI: 1 ID: 2576
5	2.034796986	10.0.0.10	10.0.1.10	ICMP	100	Echo (ping) request id=0x4040, seq=3/768, ttl=64 (no respons...
6	2.035784665	00:aa:bb:00:00:00		0x4000	96	PRI: 4 DEI: 1 ID: 2586
7	2.066725067	fe80::40df:89ff:feb..ff02::2		ICMPv6	72	Router Solicitation from 42:df:89:b5:5d:55
8	2.066769679	fe80::820:c6ff:fe5b..ff02::2		ICMPv6	72	Router Solicitation from 0a:20:c6:5b:0f:58
9	3.058821119	10.0.0.10	10.0.1.10	ICMP	100	Echo (ping) request id=0x4040, seq=4/1024, ttl=64 (no respons...
10	3.059577647	00:aa:bb:00:00:00		0x4000	96	PRI: 4 DEI: 1 ID: 2822

[Time shift for this packet: 0.000000000 seconds]  
Epoch Time: 1684682759.407170716 seconds  
[Time delta from previous captured frame: 0.000752501 seconds]  
[Time delta from previous displayed frame: 0.000752501 seconds]  
[Time since reference or first frame: 0.000752501 seconds]  
Frame Number: 2  
Frame Length: 96 bytes (768 bits)  
Capture Length: 96 bytes (768 bits)  
[Frame is marked: False]  
[Frame is ignored: False]  
[Protocols in frame: sll:ethertype:vlan:ethertype:data]

- ▼ Linux cooked capture
  - Packet type: Sent by us (4)
  - Link-layer address type: 1
  - Link-layer address length: 6
  - Source: 00:aa:bb:00:00:00 (00:aa:bb:00:00:00)
  - Unused: 0000

Protocol: 802.1Q Virtual LAN (0x8100)

- 802.1Q Virtual LAN, PRI: 4, DEI: 1, ID: 2383
- Data (76 bytes)

Hex	Dec	Source	Dest	Length
0000	00 04 00 01 00 06 aa bb 00 00 00 00 00 01 00			96
0010	99 4f 40 00 40 01 8c 46 0a 00 00 0a 0a 00 01 0a	00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00	00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00	96
0020	08 00 1f 1c 40 40 00 01 07 38 6a 64 00 00 00 00			96
0030	62 33 06 00 00 00 00 00 10 11 12 13 14 15 16 17	b3.....		96
0040	18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27	..... !#%&'		96
0050	28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37	()*+, -./ 01234567		96

Rysunek 37: Test - widoczny VLAN tag.

```

table_set_default cIngress.vlan_parse_exact drop

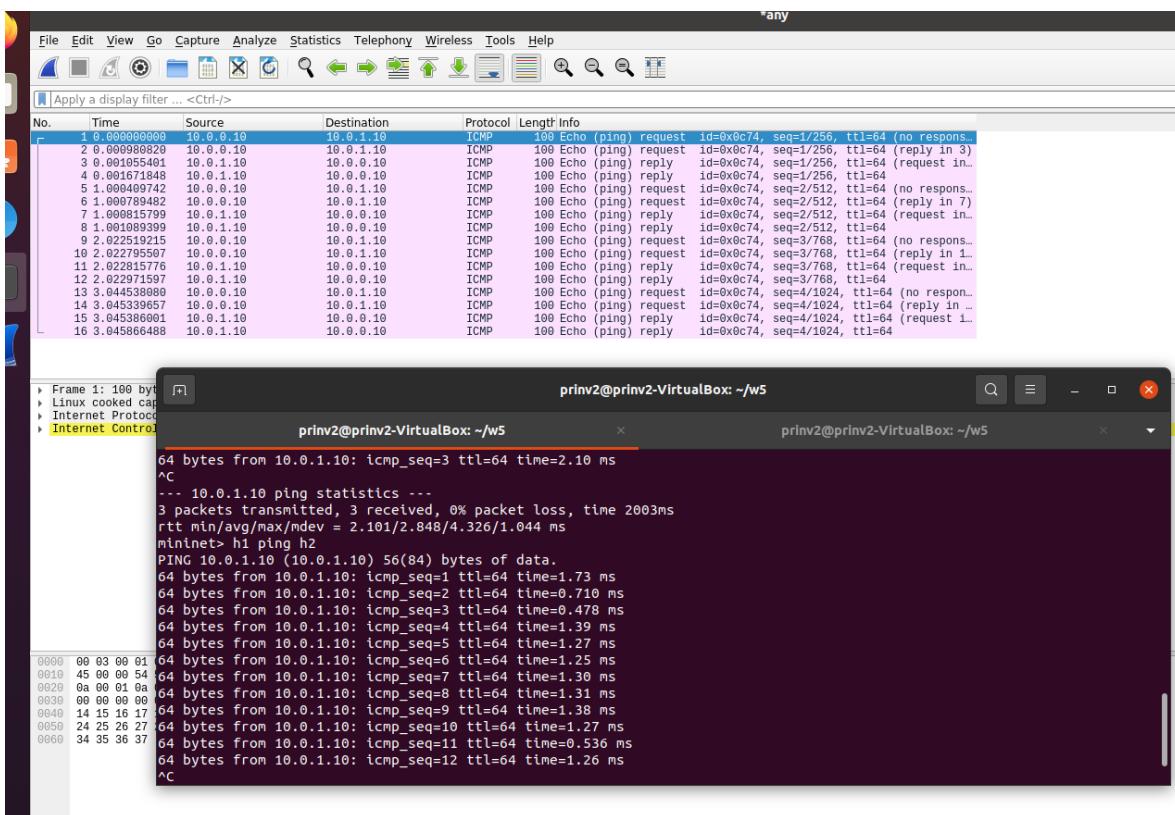
table_add cIngress.vlan_parse_exact mac_forward 00:04:00:00:00:00 => 00:04:00:00:00:01 2
table_add cIngress.vlan_parse_exact mac_forward 00:04:00:00:00:01 => 00:04:00:00:00:00 1

table_add cIngress.add_vlanTag_exact add_vlanTag 00:04:00:00:00:00 => 00:04:00:00:00:01 2 255
table_add cIngress.add_vlanTag_exact add_vlanTag 00:04:00:00:00:01 => 00:04:00:00:00:00 1 255

table_add cEgress.remove_vlanTag_exact remove_vlanTag 1 =>

```

Rysunek 38: Zasady tablic - usuwanie VLAN tag'u na maszynie odpowiadającej.



Rysunek 39: Test - usunięty VLAN tag.

```

table_set_default cIngress.vlan_parse_exact drop

table_add cIngress.vlan_parse_exact mac_forward 00:04:00:00:00:00 => 00:04:00:00:00:01 2
table_add cIngress.vlan_parse_exact mac_forward 00:04:00:00:00:01 => 00:04:00:00:00:00 1

table_add cIngress.add_vlanTag_exact add_vlanTag 00:04:00:00:00:00 => 00:04:00:00:00:01 2 255
table_add cIngress.add_vlanTag_exact add_vlanTag 00:04:00:00:00:01 => 00:04:00:00:00:00 1 255

table_add cEgress.remove_vlanTag_exact remove_vlanTag 0 =>
table_add cEgress.remove_vlanTag_exact remove_vlanTag 1 =>

```

Rysunek 40: Zasady tablic - usuwanie obu VLAN tag'ów.

```

typedef bit<2> color_t;
struct metadata{
    bit<16> port;
    color_t pkt_color;
}

```

Rysunek 41: Stworzenie zmiennej koloru w metadata.

W celu ograniczenia natężeń ruchu meter'y zostały zastosowane w bloku ingress. Następnie w tablicy decydujemy się na obsłużenie danego koloru określoną akcją. W projekcie zostały wykorzystane następujące funkcje → drop/NoAction (NoAction sprawia, że pakiet jest dalej obsługiwany, a przez to w konsekwencji przekazywany dalej). Implementacja ograniczenia ruchu jest widoczna na Fig. 42.

```

direct_meter<color_t>(MeterType.bytes) dmeter_instance;
action drop(){
    mark_to_drop(stdmeta);
}

action mark_packet(){
    dmeter_instance.read(meta(pkt_color));
}

table mark_packet_any{
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        mark_packet;
        drop;
    }
    meters = dmeter_instance;
    default_action = mark_packet;
}

table filter_meters_exact{
    key = {
        meta(pkt_color: exact;
    }
    actions = {
        NoAction;
        drop;
    }
    default_action = drop();
}

```

Rysunek 42: Ingress - implementacja meter'a w ograniczeniu ruchu.

W celu zapewnienia poprawnej funkcjonalności konieczna była edycja komend podawanych *simple\_switch\_CLI*. Podobnie jak w poprzednich częściach raportu zostało to zautomatyzowane. Najpierw zaznaczamy, że dla wszystkich hostów w naszej sieci chcemy ją stosować. Następnie określamy, że kolor zielony (0) oraz żółty (1) są stosowane do akcji NoAction - pakiet oznaczony jako czerwony zostanie odrzucony. Ostatnim parametrem są dane dla meter'a - pierwsza para mówi, aby oznaczyć na czerwono pakiety, jeśli przepływność jest większa niż 0.1 bita na 1 mikrosekundę ( $100Kbit/s$ ) lub MTU przekracza 2000. Druga para określa parametry oznaczenia na kolor żółty (jeśli pakiet nie jest czerwony).

```
table_add cIngress.mark_packet_any mark_packet 10.0.0.10/12 =>
table_add cIngress.filter_meters_exact NoAction 0 =>
table_add cIngress.filter_meters_exact NoAction 1 =>
meter_array_set_rates dmeter_instance 0.1:2000 0.1:2000
```

Rysunek 43: Zaktualizowana tablica dla *simple\_switch\_CLI*.

### 15.1 Testowanie

W celu przeprowadzenia testów został wykorzystany iperf3. Jego uruchomienie w środowisku mininet odbywa się za pomocą następujących komend: Najpierw został przeprowadzony test prędkości przy wyłączonym ograniczeniu ruchu: Fig. 45. Następnie zostały sprawdzone możliwości ograniczenia ruchu do  $100Kbit/s$  - Fig 46,  $200Kbit/s$  - Fig. 47,  $1Mbit/s$  - Fig. 48. Jak można zaobserwować wartości otrzymane w testach nieznacznie się różnią - wynika to z opóźnień w kanale i emulacji. Jednakże widać, że otrzymane ograniczenie jest w wystarczającym stopniu dokładne, a przede wszystkim wyraźnie widać zależność.

```
mininet> h2 iperf3 -s -p 80 &
-----
Server listening on 80
-----
mininet> h1 iperf3 -c h2 -p 80
```

Rysunek 44: Uruchomienie iperf3 w mininet.

```
prinv2@prinv2-VirtualBox: ~/w7          prinv2@prinv2-VirtualBox: ~/w7
default interface: eth0 10.0.0.10      00:04:00:00:00:00
*****
*****
h2
default interface: eth0 10.0.1.10      00:04:00:00:00:01
*****
Ready !
*** Starting CLI:
mininet> h2 iperf3 -s -p 80 &
-----
Server listening on 80
-----
mininet> h1 iperf3 -c h2 -p 80
Connecting to host 10.0.1.10, port 80
[ 5] local 10.0.0.10 port 56408 connected to 10.0.1.10 port 80
[ ID] Interval           Transfer     Bitrate    Retr  Cwnd
[ 5]  0.00-1.00   sec   14.8 MBytes   125 Mbits/sec   0   655 KBytes
[ 5]  1.00-2.00   sec   11.2 MBytes   94.4 Mbits/sec  146  689 KBytes
[ 5]  2.00-3.00   sec   11.2 MBytes   94.4 Mbits/sec   36  536 KBytes
[ 5]  3.00-4.00   sec   8.75 MBytes   73.3 Mbits/sec  147  103 KBytes
[ 5]  4.00-5.00   sec   6.25 MBytes   52.5 Mbits/sec   41  59.4 KBytes
[ 5]  5.00-6.00   sec   7.50 MBytes   62.9 Mbits/sec   20  99.0 KBytes
[ 5]  6.00-7.00   sec   10.0 MBytes   83.9 Mbits/sec   11  116 KBytes
[ 5]  7.00-8.00   sec   10.0 MBytes   83.9 Mbits/sec   50  94.7 KBytes
[ 5]  8.00-9.00   sec   13.8 MBytes   115 Mbits/sec   2   141 KBytes
[ 5]  9.00-10.00  sec   11.2 MBytes   94.4 Mbits/sec  71  115 KBytes
[ ID] Interval           Transfer     Bitrate    Retr
[ 5]  0.00-10.00  sec   105 MBytes   87.9 Mbits/sec  524             sender
[ 5]  0.00-10.05  sec   100 MBytes   83.7 Mbits/sec                   receiver

iperf Done.
mininet>
```

Rysunek 45: Test prędkości bez ograniczeń.

Rysunek 46: Test prędkości - ograniczenie do  $100\text{kbit/s}$

Rysunek 47: Test prędkości - ograniczenie do  $200Kbit/s$

```
mininet> h2 iperf3 -s -p 80 &
mininet> h1 iperf3 -c h2 -p 80
Connecting to host 10.0.1.10, port 80
[ 5] local 10.0.0.10 port 51020 connected to 10.0.1.10 port 80
[ ID] Interval          Transfer     Bitrate      Retr  Cwnd
[ 5]  0.00-1.04  sec   344 KBytes  2.72 Mbits/sec  46  2.83 KBytes
[ 5]  1.04-2.00  sec   184 KBytes  1.56 Mbits/sec  23  1.41 KBytes
[ 5]  2.00-3.00  sec   178 KBytes  1.46 Mbits/sec  21  2.83 KBytes
[ 5]  3.00-4.00  sec   116 KBytes  952 Kbits/sec  21  2.83 KBytes
[ 5]  4.00-5.07  sec   117 KBytes  902 Kbits/sec  14  2.83 KBytes
[ 5]  5.07-6.03  sec   59.4 KBytes 503 Kbits/sec  15  2.83 KBytes
[ 5]  6.03-7.00  sec   120 KBytes  1.02 Mbits/sec  27  2.83 KBytes
[ 5]  7.00-8.00  sec   120 KBytes  987 Kbits/sec  24  2.83 KBytes
[ 5]  8.00-9.03  sec   117 KBytes  938 Kbits/sec  19  2.83 KBytes
[ 5]  9.03-10.01 sec   116 KBytes  962 Kbits/sec  16  2.83 KBytes
[ ID] Interval          Transfer     Bitrate      Retr
[ 5]  0.00-10.01 sec  1.44 MBytes  1.20 Mbits/sec 226
[ 5]  0.00-10.04 sec  1.36 MBytes  1.14 Mbits/sec
                                         sender
                                         receiver

iperf Done.
mininet>
```

Rysunek 48: Test prędkości - ograniczenie do 1Mbit/s

## 16 Schowek sieciowy

Zgodnie z instrukcją schowek miał na celu:

1. Przechowywanie i współdzielenie małych porcji informacji
2. Wykrywanie specjalnego rodzaju pakietów
3. Zapisywanie informacji z pakietu na switchu
4. Odesłanie informacji do nadawcy

Aby sprawdzić ogólną nożliwość wykorzystania schowka zostały przyjęte następujące założenia wiadoczne w Tab. 3. Rozmiar trzymanych danych można z łatwością przeskalować do większych rozmiarów, jednakże w projekcie został wykorzystany tylko jako proof of concept. Dla wartości EtherType zostały przyjęte, gdyż nie są wykorzystywane w żadnym powszechnie używanym protokole. Zostało również przyjęte, że specjalny rodzaj pakietu składa się z ramki ethernet oraz nagłówka data (Fig. 49).

Rozmiar trzymanych danych:	128 bity
EtherType nagłówka z danymi do zapisania:	0x690
EtherType nagłówka do odebrania dancyh:	0x700

Tabela 3: Przyjęte założenia

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

header data_t{
    bit<128> field_1;
}
```

Rysunek 49: Nagłówki używanych pakietów przez schowek.

Ogólny schemat działania schowka został przedstawiony na Fig. 50. Dodatkowo, docelowy adres MAC urządzenia nie ma znaczenia, ponieważ switch przy otrzymania pakietu prosiącego o zwrot danych wykorzystuje adres MAC urządzenia dokonującego tejże prośby.

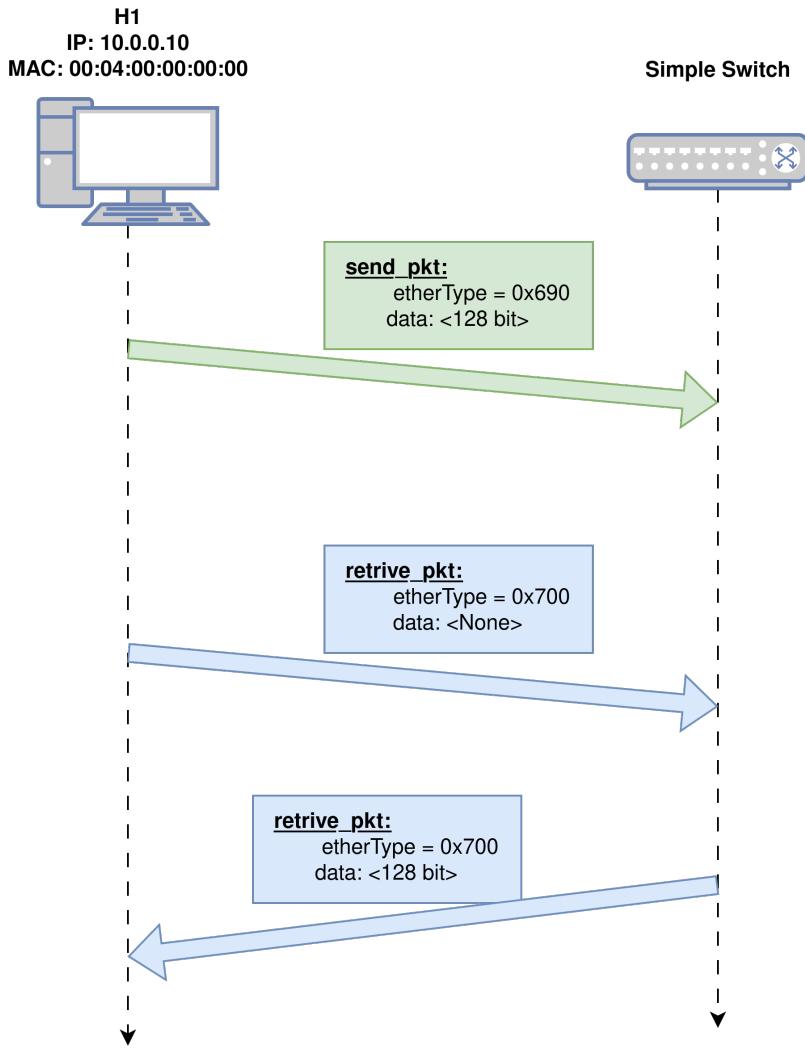
### 16.1 Program P4

Jakoże został przyjęty specjalny rodzaj pakietów, posiadający szczególny EtherType, w związku z tym konieczna było jego odpowiednie parsowanie (Fig. 51).

Po parsowaniu koniecznym było przeście przez blok ingress, w którym dochodzi do zapisania/odczytania pakietu (Fig. 52, Fig. 53, Fig. 54). Warto zauważyć, że blok wykonuje się zawsze po otrzymaniu danego typu pakietu, z tego powodu wartość klucza nie jest podana.

### 16.2 Skrypt testowy

W celu testowania schowka zostały stworzone dwa proste skrypty w pythonie. Wykorzystują one bibliotekę scapy do wysłania specjalnego typu pakietów. Jeden tworzy pakiet z danymi do zapisania, natomiast drugi służy do zgłoszenia prośby o odesłanie wcześniej zapisanych danych. Kod obu skryptów jest dostępny zarówno na Fig. 55, Fig. 56 jak i w załączniku.



Rysunek 50: Schemat działania schowka.

### 16.3 Tesowanie

Testowanie odbyło się w sposób następujący: w środowisku mininet została nadana wiadomość przez hosta do switcha z danymi, a następnie została zgłoszona prośba przez tego hosta o odesłanie tychże danych (Fig. 57). Cała transmisja jest monitorowana w programie wireshark. Dane będące wysłane to liczba 11 (0x0b w systemie szesnastkowym) (Fig. 58). Sukcesem jest odesłany pakiet do hosta z poprawnymi danymi (Fig. 60) - funkcjonalność schowka została zaimplementowana.

## 17 Instalacja p4runtime i przygotowanie środowiska

-opisać instalację i komplikację -zmiany w skryptach Pierwszą czynnością w celu realizacji zadania była instalacja zależności. W tym celu została użyta komenda:

```
pip3 install --upgrade \
git+https://github.com/p4lang/p4runtime-shell.git#egg=httpie
```

Następnie zgodnie z instrukcją warsztatu została dodana modyfikacja do pliku *p4\_mininet.py* (plik dostępny w załączniku). Podobnież konieczna była zmiana w pliku *run.sh* w celu uruchomienia serwera p4runtime na adresie 0.0.0.0:9559. Tak samo konieczna była zmiana w skrypcie komplikacyjnym - należało dodać odpowiednią flagę w celu wygenerowania pliku konfiguracyjnego serwera.

```

state parse_ethernet {
    pkt.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType){
        TYPE_IPV4: parse_ipv4;
        TYPE_SAVE_PKT: parse_save_pkt;
        TYPE_READ_PKT: parse_read_pkt;
        default: accept;
    }
}

state parse_read_pkt{
    pkt.extract(hdr.data);
    meta.eth_hdr = hdr.ethernet;
    transition accept;
}

state parse_save_pkt{
    pkt.extract(hdr.data);
    meta.data = hdr.data.field_1;
    transition accept;
}

```

Rysunek 51: Parsowanie specjalnych typów pakietów.

```

action read_packet(){
    clipboard.read(meta.data, 0);
    hdr.data.field_1 = meta.data;
    hdr.ethernet.dstAddr = meta.eth_hdr.srcAddr;
    hdr.ethernet.srcAddr = meta.eth_hdr.dstAddr;
    stdmeta.egress_spec = stdmeta.ingress_port;
}

table read_pkt{
    key = {
    }
    actions = {
        read_packet;
        drop;
    }
    default_action = read_packet;
}

```

Rysunek 52: Ingress - zapisanie danych z pakietu.

## 18 Wstępne rozeznanie w p4runtime

Został przygotowany skrypt *run\_p4runtime.sh* w celu szybkiego uruchomiania środowiska p4runtime. Po skompilowaniu programu w języku P4 i następnie uruchomieniu środowiska mininet można połączyć się z serwerem p4runtime po przez wyżej wymieniony skrypt. Wówczas jest uruchomiony interpreter pythona przez który można oddziaływać na serwer p4runtime. Przykładowo, można zobaczyć listę akcji dla danej tabeli (Rys. 61).

```

register<bit<128>>(1) clipboard;

action drop(){
    mark_to_drop(stdmeta);
}

action save_packet(){
    clipboard.write(0, meta.data);
    hdr.ethernet.dstAddr = meta.eth_hdr.srcAddr;
    hdr.ethernet.srcAddr = meta.eth_hdr.dstAddr;
}

table save_pkt{
    key = {
    }
    actions = {
        save_packet;
        drop;
    }
    default_action = save_packet;
}

```

Rysunek 53: Ingress - wczytanie danych z pamięci.

```

apply{
    if(hdr.ethernet.etherType == TYPE_READ_PKT){
        read_pkt.apply();
    }
    else if(hdr.ethernet.etherType == TYPE_SAVE_PKT){
        save_pkt.apply();
    }
}

```

Rysunek 54: Ingress - blok apply.

## 19 MAC learning

Celem warsztatu było dodanie funkcjonalności MAC learning. Schemat działania miał być następujący:

1. Powiadomienie płaszczyzny sterowania o nieznanym adresie MAC,
2. Podmiana adresu docelowego na podstawie adresu IP,
3. Brak dodatkowego filtrowania,
4. Bez tworzenia aplikacji płaszczyzny sterowania (tylko kod P4)

W tym celu zostało stworzone następujące rozwiązanie. Zostały przygotowane dwie tabele. Pierwsza służąca do zapisywania (zapamiętowania) adresu MAC oraz portu na jakim znajduje się nowe urządzenie. Druga natomiast miała służyć do przekazywania pakietów na odpowiedni port docelowy jeśli takowy został zapamiętany. W celu powiadomienia płaszczyzny sterowania o nieznanym adresie MAC została wykorzystana funkcja *digest*.

Funkcja *digest* może być wykorzystana tylko i wyłącznie w bloku Ingress. Przyjmuje ona dwa parametry: adres ip odbiorcy wiadomości oraz dane, które mają zostać wysłane. W tym zadaniu adres ip nie był istotny, dlatego została podana wartość 1. Natomiast dane, które są przekazywane zostały zapisane w przygotowanej do tego strukturze i umieszczone w zmiennej metadata. Utworzona zmienna jest widoczna na Rys. 62.

Pierwsza utworzona tabela odpowiadała za dodawanie adresów MAC oraz portów na jakich zostały odnotowane, jeśli wcześniej nie wystąpiły. Zatem jeśli zostały już wcześniej odnotowane to nie powinno się nic dziać. Natomiast w przypadku, gdy są nowe to powinien zostać wysłany digest do płaszczyzny sterowania ze stosownymi danymi. Implementacja tejże tablicy jest widoczna na Rys. 63.

```

#!/usr/bin/python3

from scapy.all import *
import sys

class DataField(Packet):
    name = "DataField"
    fields_desc = [BitField(name="field_1", default=11, size=128)]


def createPacket(src, dst, payload):
    pkt = Ether()
    pkt.src = src
    pkt.dst = dst
    pkt.type = 0x690
    #   d = DataField(field_1=payload)
    d = DataField()
    packet = pkt/d
    return packet

def main():
    _, src, dst, payload = sys.argv
    print(src, dst, payload)
    packet = createPacket(src, dst, payload)
    sendp(packet, iface='eth0')
    packet.show()

if __name__=="__main__":
    main()

```

Rysunek 55: Tworzenie pakietu z danymi do zapisania.

```

#!/usr/bin/python3

from scapy.all import *
import sys

class DataField(Packet):
    name = "DataField"
    fields_desc = [BitField(name="field_1", default=1, size=128)]


def retrievePacket(src, dst, payload):
    pkt = Ether()
    pkt.src = src
    pkt.dst = dst
    pkt.type = 0x700
    d = DataField()

    packet = pkt/d
    return packet

def main():
    _, src, dst, payload = sys.argv
    print(src, dst, payload)
    packet = retrievePacket(src, dst, payload)
    sendp(packet, iface='eth0')
    packet.show()

if __name__=="__main__":
    main()

```

Rysunek 56: Tworzenie pakietu proszącego o pobranie danych.

Następna tabela odpowiada za przekazywanie pakietów na dany port. Jest to funkcjonalność znana na przykład z warsztatu 5. Implementacja tabeli do przekazywania pakietów jest widoczna na Rys. 64.

```

h1
default interface: eth0 10.0.0.10      00:04:00:00:00:00
*****
*****
h2
default interface: eth0 10.0.1.10      00:04:00:00:00:01
*****
Ready !
*** Starting CLI:
mininet> h1 ./send_pkt.py 00:04:00:00:00:00 00:04:00:00:00:02 1
00:04:00:00:00:00 00:04:00:00:00:02 1
.
Sent 1 packets.
###[ Ethernet ]###
  dst      = 00:04:00:00:00:02
  src      = 00:04:00:00:00:00
  type     = 0x690
###[ DataField ]###
  field_1  = 11

mininet> h1 ./retrive_pkt.py 00:04:00:00:00:00 00:04:00:00:00:02 1
00:04:00:00:00:00 00:04:00:00:00:02 1
.
Sent 1 packets.
###[ Ethernet ]###
  dst      = 00:04:00:00:00:02
  src      = 00:04:00:00:00:00
  type     = 0x700
###[ DataField ]###
  field_1  = 1

mininet>

```

Rysunek 57: Uruchomienie pakietów testowych w środowisku mininet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::c04e:3fff:fee.. ff02::fb		MDNS	127 Standard query 0x0000 PTR _pgpkey-hkp._tcp.local, "QM" questi...	
2	0.228596789	fe80::fc81:8dff:fea.. ff02::fb		MDNS	127 Standard query 0x0000 PTR _pgpkey-hkp._tcp.local, "QM" questi...	
3	0.040400079	LexmarkP 00:00:00		0x0690	32 Unicast to another host	
4	4.607745228	fe80::fc81:8dff:fea.. ff02::2		ICMPv6	72 Router Solicitation from fe81:8b:a9:91:64	
5	5.092239698	LexmarkP 00:00:00		0x0700	32 Unicast to another host	
6	5.092554777	LexmarkP 00:00:02		0x0700	32 Sent by us	
7	5.117657532	fe80::c04e:3fff:fee.. ff02::2		ICMPv6	72 Router Solicitation from c2:4e:3f:e6:af:45	

Epoch Time: 1685466031.864690678 seconds
[Time delta from previous captured frame: 1.811803290 seconds]
[Time delta from previous displayed frame: 1.811803290 seconds]
[Time since reference or first frame: 2.040400079 seconds]
Frame Number: 3
Frame Length: 32 bytes (256 bits)
Capture Length: 32 bytes (256 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: sll:ethertype:data]
Linux cooked capture
Packet type: Unicast to another host (3)
Link-layer address type: 1
Link-layer address length: 6
Source: LexmarkP _00:00:00 (00:04:00:00:00:00)
Unused: 0000
Protocol: Unknown (0x0690)
Data (16 bytes)
Data: 00000000000000000000000000000000
[Length: 16]
0000 00 03 00 01 00 06 00 04 00 00 00 00 00 00 06 90 .....
0010 00 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Rysunek 58: Monitorowanie - oględziny pakietu z danymi hosta.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::c04e:3fff:fee... ff02::fb		MDNS	127	Standard query 0x0000 PTR _pgpkey-hkp._tcp.local, "QM" questi...
2	0.228596789	fe80::fc81:8bff:fea... ff02::fb		MDNS	127	Standard query 0x0000 PTR _pgpkey-hkp._tcp.local, "QM" questi...
3	0.040400079	LexmarkP_00:00:00			0x0690	32 Unicast to another host
4	4.607745228	fe80::fc81:8bff:fea... ff02::2		ICMPv6	72	Router Solicitation from fe:81:8b:a9:91:64
5	5.092239698	LexmarkP_00:00:00			0x0700	32 Unicast to another host
6	5.092554777	LexmarkP_00:00:02			0x0700	32 Sent by us
7	5.117657532	fe80::c04e:3fff:fee... ff02::2		ICMPv6	72	Router Solicitation from c2:4e:3f:e6:af:45

Epoch Time: 1685466034.916530297 seconds  
[Time delta from previous captured frame: 0.484494470 seconds]  
[Time delta from previous displayed frame: 0.484494470 seconds]  
[Time since reference or first frame: 5.092239698 seconds]  
Frame Number: 5  
Frame Length: 32 bytes (256 bits)  
Capture Length: 32 bytes (256 bits)  
[Frame is marked: False]  
[Frame is ignored: False]  
[Protocols in frame: sll:ethertype:data]

- Linux cooked capture
  - Packet type: Unicast to another host (3)
  - Link-layer address type: 1
  - Link-layer address length: 6
  - Source: LexmarkP\_00:00:00 (00:04:00:00:00:00)
  - Unused: 0000
  - Protocol: Unknown (0x0700)
- Data (16 bytes)
  - Data: 00000000000000000000000000000001
  - [Length: 16]

0000	00	03	00	01	00	06	00	04	00	00	00	00	00	00	07	00	.....	.....
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	.....	.....

Rysunek 59: Monitorowanie - oględziny pakietu proszącego o dane.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::c04e:3fff:fee... ff02::fb		MDNS	127	Standard query 0x0000 PTR _pgpkey-hkp._tcp.local, "QM" questi...
2	0.228596789	fe80::fc81:8bff:fea... ff02::fb		MDNS	127	Standard query 0x0000 PTR _pgpkey-hkp._tcp.local, "QM" questi...
3	0.040400079	LexmarkP_00:00:00			0x0690	32 Unicast to another host
4	4.607745228	fe80::fc81:8bff:fea... ff02::2		ICMPv6	72	Router Solicitation from fe:81:8b:a9:91:64
5	5.092239698	LexmarkP_00:00:00			0x0700	32 Unicast to another host
6	5.092554777	LexmarkP_00:00:02			0x0700	32 Sent by us
7	5.117657532	fe80::c04e:3fff:fee... ff02::2		ICMPv6	72	Router Solicitation from c2:4e:3f:e6:af:45

Epoch Time: 1685466034.916845376 seconds  
[Time delta from previous captured frame: 0.000315079 seconds]  
[Time delta from previous displayed frame: 0.000315079 seconds]  
[Time since reference or first frame: 5.092554777 seconds]  
Frame Number: 6  
Frame Length: 32 bytes (256 bits)  
Capture Length: 32 bytes (256 bits)  
[Frame is marked: False]  
[Frame is ignored: False]  
[Protocols in frame: sll:ethertype:data]

- Linux cooked capture
  - Packet type: Sent by us (4)
  - Link-layer address type: 1
  - Link-layer address length: 6
  - Source: LexmarkP\_00:00:02 (00:04:00:00:00:02)
  - Unused: 0000
  - Protocol: Unknown (0x0700)
- Data (16 bytes)
  - Data: 0000000000000000000000000000000b
  - [Length: 16]

0000	00	04	00	01	00	06	00	04	00	00	00	02	00	00	07	00	.....	.....
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0b	.....	.....

Rysunek 60: Monitorowanie - oględziny odesłanego pakietu z danymi przez switch.

```

P4Runtime sh >>> l = table_entry["cIngress.l2_table"](action="mac_forward")

P4Runtime sh >>> l
Out[51]:
table_id: 46424487 ("cIngress.l2_table")
action {
    action {
        action_id: 30943298 ("cIngress.mac_forward")
    }
}

P4Runtime sh >>> l.actions?
Object 'l.actions' not found.

P4Runtime sh >>> l.action?
Type: Action
String form: action_id: 30943298
File:      ~/.local/lib/python3.8/site-packages/p4runtime_sh/shell.py
Docstring:
Action parameters for action 'mac_forward':

id: 1
name: "dstAddr"
bitwidth: 48
id: 2
name: "port"
bitwidth: 9

Set a param value with <self>['<param_name>'] = '<value>'
You may also use <self>.set(<param_name>='<value>')

```

Rysunek 61: Lista akcji dla danej tabeli

```

struct mac_learn_t{
    bit<48> srcAddr;
    bit<9> ingress_port;
    bit<32> srcIpAddr;
}

struct metadata{
    egressSpec_t port;
    mac_learn_t mac_learn_msg;
}

```

Rysunek 62: Tworzenie danych do wysłania przez *digest*.

```

action unknown_source(){
    meta.mac_learn_msg.srcAddr = hdr.ethernet.srcAddr;
    meta.mac_learn_msg.ingress_port = stdmeta.ingress_port;
    meta.mac_learn_msg.srcIpAddr = hdr.ipv4.srcAddr;
    digest<mac_learn_t>(0, meta.mac_learn_msg);
}

table learned_MAC{
    key = {
        hdr.ethernet.srcAddr: exact;
    }
    actions = {
        unknown_source;
        NoAction;
    }
    default_action = unknown_source();
}

```

Rysunek 63: Tabela wysyłająca do płaszczyzny sterowania nieznane adresy MAC.

```

action mac_forward(macAddr_t dstAddr, egressSpec_t port){
    stdmeta.egress_spec = port;
    //      hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
}

table l2_table{
    key = {
        hdr.ipv4.dstAddr: exact;
    }
    actions = {
        mac_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}

apply{
    if(hdr.ipv4.isValid()){
        learned_MAC.apply();
        l2_table.apply();
    }
}

```

Rysunek 64: Tabela do przekazywania pakietów.

## 20 Konfiguracja p4runtime

Mając przygotowane tabele koniecznym było przygotowanie konfiguracji w p4runtime. W tym celu koniecznym było stworzenie listy digest'ów. Następnie w kółko powinien być odczytywany digest. Z niego powinny być wczytywane dane do zmiennych (mac adres źródł, port wejściowy oraz ip źródła). Następnie na ich podstawie powinien być dodany wpis do tabeli ze znymi adresami MAC. W dalszej kolejności do tabeli przekazywania pakietów powinien być dodany nowy wpis zawierający jako docelowy adres ip wcześniej wczytany adres ip, jako docelowy adres mac wczytany adres mac, a jako port docelowy również wcześniej wczytany port.

Implementacja tego rozwiązania okazała się nie taka oczywista - wymagała sparsowania adresu MAC z bytestream'u do postaci "aa:bb:cc:dd:ee:ff". Problemem jaki wystąpił było nie przekazywanie adresu ip do płaszczyzny sterowania - za każdym razem wartość adresu ip była równa 0. Z tego powodu zostało dodane uproszczenie (adres ip jest podstawiany). Pełna implementacja jest widoczna na Rys. 65 i Rys. 66 (również w pliku *p4runtime\_script.py*). Niestety z przyczyn błędu bibliotek skrypt trzeba kopiować do środowiska p4runtime; nie można go uruchomić osobno.

```
#digest list creation
d = digest_entry['mac_learn_t']
d.ack_timeout_ns=1000000000
d.max_timeout_ns=1000000000
d.max_list_size=100
d.insert()

while True:
    try:
        de = digest_list.digest_list_queue.get() #proceed if digest is found
        a = de.digest.data[0]
        b = a.struct
        c = b.members[0].bitstring
        mac = clear_mac_addr(c)
        port = b.members[1].bitstring[0]
        ipv4 = b.members[2].bitstring

        te = TableEntry("cIngress.learned_MAC")(action="NoAction")
        te.match["srcAddr"] = mac
        te.insert()

        tab2 = TableEntry("cIngress.l2_table")(action="mac_forward")
        if mac == "00:04:00:00:00:00":
            tab2.match["dstAddr"] = "10.0.0.10"
        elif mac == "00:04:00:00:00:01":
            tab2.match["dstAddr"] = "10.0.1.10"
        tab2.action["dstAddr"] = mac
        tab2.action["port"] = str(port)

        tab2.insert()
    except:
        print("entry was duplicated")
```

Rysunek 65: Skrypt odczytujący digest i zapisujący rekordy do tabeli.

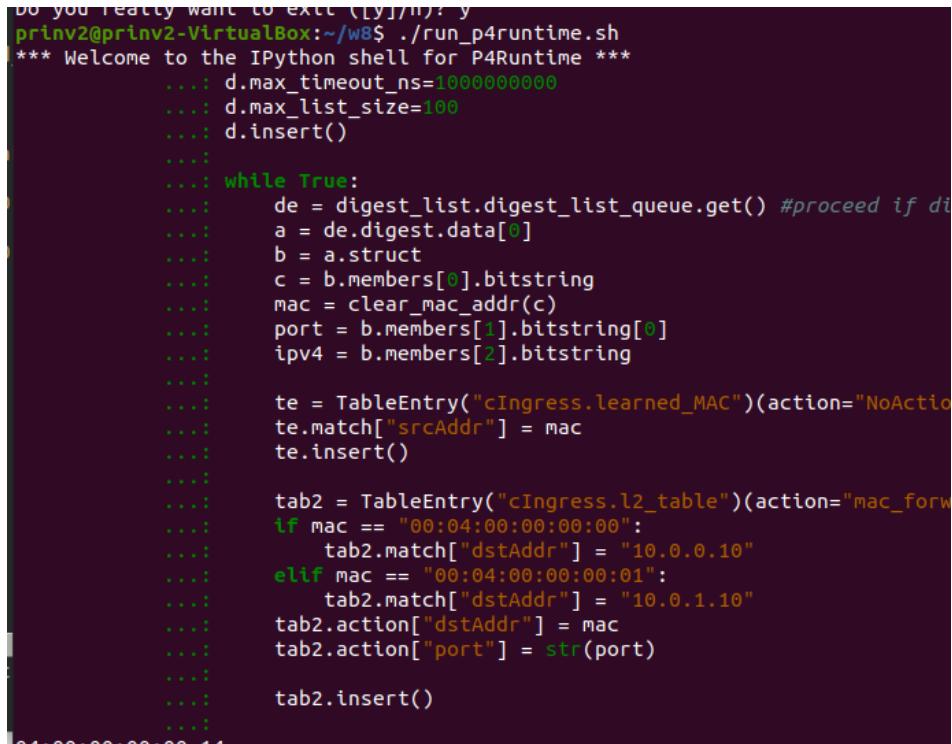
```
def clear_mac_addr(c):
    s = c.hex()
    mac = ""
    for i in range(0, len(s)-1, 2):
        for j in range(i,i+2):
            mac+=s[j]
        mac += ":"
    if mac[-1] == ":":
        mac = mac[:len(mac)-1]
    octets = len(mac)
    print(mac, octets)
    while octets < 17:
        octets += 3
        mac = "00:" + mac

    return mac
```

Rysunek 66: Parsowanie adresu MAC do poprawnej formy.

## 21 Uruchomienie skryptu

Po uruchomieniu sieci mininet, a następnie skryptu *run\_p4runtime.sh* należało przekleić wcześniej przygotowaną konfigurację środowiska (Rys.67).



```
Do you really want to exit ([y]/n)? y
prinv2@prinv2-VirtualBox:~/wb$ ./run_p4runtime.sh
*** Welcome to the IPython shell for P4Runtime ***
.... d.max_timeout_ns=1000000000
.... d.max_list_size=100
.... d.insert()
.... 
.... while True:
....     de = digest_list.digest_list_queue.get() #proceed if digest
....     a = de.digest.data[0]
....     b = a.struct
....     c = b.members[0].bitstring
....     mac = clear_mac_addr(c)
....     port = b.members[1].bitstring[0]
....     ipv4 = b.members[2].bitstring
.... 
....     te = TableEntry("cIngress.learned_MAC")(action="NoAction")
....     te.match["srcAddr"] = mac
....     te.insert()
.... 
....     tab2 = TableEntry("cIngress.l2_table")(action="mac_forward")
....     if mac == "00:04:00:00:00:00":
....         tab2.match["dstAddr"] = "10.0.0.10"
....     elif mac == "00:04:00:00:00:01":
....         tab2.match["dstAddr"] = "10.0.1.10"
....     tab2.action["dstAddr"] = mac
....     tab2.action["port"] = str(port)
.... 
....     tab2.insert()
.... 
```

Rysunek 67: Konfiguracja p4runtime w trakcie działania.

Kolejnym krokiem był test działania - polecenia ping. Zgodnie z przewidywaniem, najpierw pierwsza strona probuje pingować i kończy się to niepowodzeniem. W tym czasie jej adres MAC oraz port zostają wpisane do tabeli. Następnie, druga strona pinguje pierwszą. Pierwszy ping również się nie udaje, gdyż dopiero zostanie wpisany nowy adres MAC i port do tabeli. Kolejne próby kończą się powodzeniem. Całość jest widoczna na Rys. 68. Dodatkowo, można dostrzec, że nowe wpisy pojawiły się wewnętrz p4runtime - są to odczytane digest'y (Rys. 69). Po całej procedurze został ponownie sprawdzone pingowanie w drugą stronę - jak można zauważyć ping odbywa się poprawnie (Rys. 70).

Dodatkowo, zostało sprawdzone działanie w środowisku wireshark - Rys. 71. Ponownie można zobaczyć, że mimo pierwotnego braku możliwości pingowania, jest ona możliwa po "nauczeniu się"MAC adresów przy użyciu płaszczyzny sterowania.

## 22 Podsumowanie

W tym warsztacie została dodana funkcjonalność "uczenia się"adresów MAC z użyciem płaszczyzny sterowania. W tym celu została wykorzystana funkcja *digest* umożliwiająca przekazywanie danych ze switch'a do płaszczyzny sterowania. Wymagania projektu zostały w pełni spełnione, a dostarczone rozwiązanie działa w pełni poprawnie.

```
P4 switch s1 has been started.

*****
h1
default interface: eth0 10.0.0.10      00:04:00:00:00:00
*****
h2
default interface: eth0 10.0.1.10      00:04:00:00:00:01
*****
Ready !
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
^C
--- 10.0.1.10 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1014ms

mininet> h2 ping h1
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=4 ttl=64 time=2.41 ms
64 bytes from 10.0.0.10: icmp_seq=5 ttl=64 time=0.703 ms
64 bytes from 10.0.0.10: icmp_seq=6 ttl=64 time=1.69 ms
64 bytes from 10.0.0.10: icmp_seq=7 ttl=64 time=0.648 ms
64 bytes from 10.0.0.10: icmp_seq=8 ttl=64 time=2.14 ms
64 bytes from 10.0.0.10: icmp_seq=9 ttl=64 time=0.653 ms
64 bytes from 10.0.0.10: icmp_seq=10 ttl=64 time=1.86 ms
64 bytes from 10.0.0.10: icmp_seq=11 ttl=64 time=2.08 ms
^C
--- 10.0.0.10 ping statistics ---
11 packets transmitted, 8 received, 27.2727% packet loss, time 10102ms
rtt min/avg/max/mdev = 0.648/1.522/2.412/0.690 ms
```

Rysunek 68: Pingowanie obu hostów.

```
04:00:00:00:00 14
] field_id: 1
exact {
    value: "\004\000\000\000\000"
ed}

    field_id: 1
exact {
ta  value: "\n\000\000\n"
}
.0
: param_id: 1
.0 value: "\004\000\000\000\000"

param_id: 2
value: "\001"

04:00:00:00:01 14
field_id: 1
exact {
    value: "\004\000\000\000\001"
}

    field_id: 1
exact {
    value: "\n\000\001\n"
}

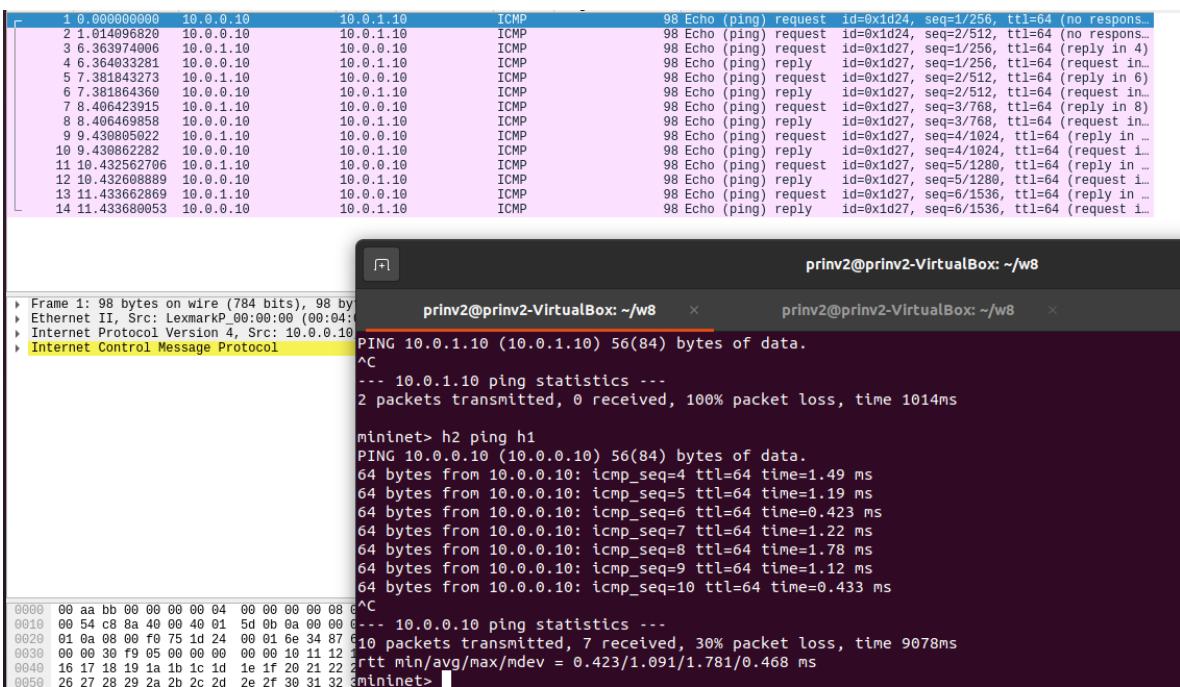
5 param_id: 1
(c value: "\004\000\000\000\001"

param_id: 2
value: "\002"
```

Rysunek 69: Odczytane digest'y.

```
mininet> h1 ping h2
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
') 64 bytes from 10.0.1.10: icmp_seq=1 ttl=64 time=1.16 ms
 64 bytes from 10.0.1.10: icmp_seq=2 ttl=64 time=1.23 ms
 64 bytes from 10.0.1.10: icmp_seq=3 ttl=64 time=1.18 ms
 64 bytes from 10.0.1.10: icmp_seq=4 ttl=64 time=1.17 ms
 64 bytes from 10.0.1.10: icmp_seq=5 ttl=64 time=1.25 ms
```

Rysunek 70: Ping w drugą stronę.



Rysunek 71: Sprawdzenie działania - wykorzystanie narzędzia wireshark.