



WYDZIAŁ ELEKTRONIKI I TECHNIK INFORMACYJNYCH

SIECI PROGRAMOWALNE I WIRTUALIZACJA FUNKCJI SIECIOWYCH

SPIW Projekt 1 - QoS routing with SDN

Eryk Głęb 318769
Patryk Kosiński 318791
Michał Drętkiewicz 298861

21 kwietnia 2024

Spis treści

1 Sposób uruchomienia	2
1.1 Sposób uruchomienia - test1	2
1.2 Sposób uruchomienia - test2	2
1.3 Sposób uruchomienia - test3	2
2 Część eksperimentalna	3
2.1 Test1 - manipulacja ilością łączy dla load-balancigu	3
2.2 Test2 - implementacja intentu dla przepływu określonej ilości bajtów w ciągu sekundy	6
2.3 Test3 - implementacja intentu dla określonej wartości opóźnienia na łączu	8
3 Podsumowanie	10

1 Sposób uruchomienia

Po każdorazowym uruchomieniu VM należy uruchomić OVS:

```
service openvswitch-switch start
```

1.1 Sposób uruchomienia - test1

1. W ścieżce `~/pox/` uruchamiamy kontroler pox z plikiem odpowiadającym za domyślny routing:

```
sudo python ./pox.py routing_controller &
```

2. W ścieżce `~/mininet/zsut/` uruchamiamy program monitorujący przepływy, program badający liczbę bajtów przesłanych na łączu, skrypt topologii Mininet:

```
sudo chmod +x limited_links_load_balancer.sh  
./limited_links_load_balancer.sh  
sudo chmod +x read_bytes.sh  
./read_bytes.sh  
sudo python routing_net.py
```

1.2 Sposób uruchomienia - test2

1. W ścieżce `~/pox/` uruchamiamy kontroler pox z plikiem odpowiadającym za domyślny routing:

```
sudo python ./pox.py routing_controller &
```

2. W ścieżce `~/mininet/zsut/` uruchamiamy program monitorujący przepływy, program badający liczbę bajtów przesłanych na łączu, skrypt topologii Mininet:

```
sudo chmod +x packet_bytes_constraint_load_balancer.sh  
./packet_bytes_constraint_load_balancer.sh  
sudo chmod +x read_bytes.sh  
./read_bytes.sh  
sudo python routing_net.py
```

1.3 Sposób uruchomienia - test3

1. W ścieżce `~/pox/` uruchamiamy kontroler pox z plikiem odpowiadającym za domyślny routing:

```
sudo python ./pox.py routing_controller &
```

2. W ścieżce `~/mininet/zsut/` uruchamiamy program monitorujący przepływy, program badający liczbę bajtów przesłanych na łączu, program sprawdzający aktualne opóźnienie na łączach wychodzących z switcha s1 oraz skrypt topologii Mininet:

```
sudo chmod +x packet_delay_constraint_load_balancer.sh
./packet_delay_constraint_load_balancer.sh
sudo chmod +x read_bytes.sh
./read_bytes.sh
sudo chmod +x getAllCurrentDelays.sh
./getAllCurrentDelays.sh
sudo python routing_net_delay.py
```

2 Część eksperymentalna

2.1 Test1 - manipulacja ilością łączy dla load-balancigu

Pierwszy eksperyment miał na celu elastyczne dostosowanie przez nas tych łącz, których chcemy używać do równoważenia obciążenia. Test obejmował kierowanie routingu jednego przepływu hosta docelowego o adresie ip = 10.0.0.5, do którego ruch był kierowany domyślnie na port s1-eth6 - domyślana konfiguracja tras dla poszczególnych przepływów została zmodyfikowana w pliku `routing_controller.py`. Ścieżki domyślne zostały dobrane dla danych przepływów na podstawie rysunku topologii.

```
msg = of.ofp_flow_mod()
msg.priority =100
msg.idle_timeout = 0
msg.hard_timeout = 1
msg.match.dl_type = 0x0800
msg.match.nw_dst = "10.0.0.4"
msg.actions.append(of.ofp_action_output(port = 4))
event.connection.send(msg)

msg = of.ofp_flow_mod()
msg.priority =100
msg.idle_timeout = 0
msg.hard_timeout = 0
msg.match.dl_type = 0x0800
msg.match.nw_dst = "10.0.0.5"
msg.actions.append(of.ofp_action_output(port = 6))
event.connection.send(msg)

msg = of.ofp_flow_mod()
msg.priority =100
msg.idle_timeout = 0
msg.hard_timeout = 0
msg.match.dl_type = 0x0800
msg.match.nw_dst = "10.0.0.6"
msg.actions.append(of.ofp_action_output(port = 5))
event.connection.send(msg)
```

Rysunek 1: Modyfikacja pliku `routing_controller.py`

Plik `limited_links_load_balancer.sh` ma na celu monitorowanie stanu przepływów z tablicy przepływów dla określonego hosta docelowego. Jeśli wykonujemy ping do hosta o adresie ip 10.0.0.5, wtedy po wywołaniu wspomnianego pliku monitorującego należy wybrać opcję "2". Po ustaleniu jaki przepływ chcemy monitorować, następnie podajemy porty wyjściowe, które będą uczestniczyć w procesie load-balancingu. Na rysunku

3 podano porty 4 oraz 6, z kolei na rysunku 4 podano porty 5 oraz 6. W przypadku rozważanego przepływu do adresu 10.0.0.5, ścieżka domyślna opejmuje wyjściowy port 6, dlatego też w tym przypadku logiczne jest podawanie portu wyjściowego dla domyślnej ścieżki jako jeden z parametrów pliku monitorującego. Ostatni minitest działania dotyczył sprawdzenia load-balancingu obejmującego wszystkie łącza wyjściowe: s1-s2 (outport=s1-eth4), s1-s3(outport=s1-eth5),s1-s4(outport=s1-eth6) .

```
student@openflow:~/mininet/zsut
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=2
592 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=1
558 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=5

student@openflow:~/mininet/zsut
student@openflow:~/mininet/zsut$ bash limited_links_load_balancer.sh
What ip address you want to load balance?
1. 10.0.0.4
2. 10.0.0.5
3. 10.0.0.6

Select 1,2 or 3: 2

Link: s1-s2 <=> output_port: 4
Link: s1-s3 <=> output_port: 5
Link: s1-s4 <=> output_port: 6

Select 4,5,6 output numbers that refer to links used for load-balancing: 5

student@openflow:~/mininet/zsut
Link s1-s3:0
Link s1-s4:1442

Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:0
Link s1-s4:1540

Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:0
Link s1-s4:1638

Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:0
Link s1-s4:1736

student@openflow:~/mininet/zsut
[2024-4-21]08.27.58 s1 p4(Sent): 0 s2 p1(Received): 0
[2024-4-21]08.27.58 s1 p6(Sent): 1 s4 p1(Received): 1
[2024-4-21]08.27.58 s1 p5(Sent): 0 s3 p1(Received): 0
[2024-4-21]08.27.59 s1 p4(Sent): 0 s2 p1(Received): 0
[2024-4-21]08.27.59 s1 p6(Sent): 1 s4 p1(Received): 1
[2024-4-21]08.27.59 s1 p5(Sent): 0 s3 p1(Received): 0
[2024-4-21]08.28.00 s1 p4(Sent): 0 s2 p1(Received): 0
[2024-4-21]08.28.00 s1 p6(Sent): 1 s4 p1(Received): 1
[2024-4-21]08.28.00 s1 p5(Sent): 0 s3 p1(Received): 0
[2024-4-21]08.28.01 s1 p4(Sent): 0 s2 p1(Received): 0
[2024-4-21]08.28.01 s1 p6(Sent): 1 s4 p1(Received): 1
[2024-4-21]08.28.01 s1 p5(Sent): 0 s3 p1(Received): 0
[2024-4-21]08.28.02 s1 p4(Sent): 0 s2 p1(Received): 0
[2024-4-21]08.28.02 s1 p6(Sent): 1 s4 p1(Received): 1
[2024-4-21]08.28.02 s1 p5(Sent): 0 s3 p1(Received): 0
[2024-4-21]08.28.03 s1 p4(Sent): 0 s2 p1(Received): 0
[2024-4-21]08.28.03 s1 p6(Sent): 1 s4 p1(Received): 1
[2024-4-21]08.28.03 s1 p5(Sent): 0 s3 p1(Received): 0
```

Rysunek 2: Początkowy ruch kierowany przez domyślne łącze.

```
student@openflow: ~/mininet/zsut
mininet> h1 ping 10.0.0.5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=2
657 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=1
638 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=6

student@openflow: ~/mininet/zsut
Select 1, 2 or 3: 2

Link: s1-s2 <=> output_port: 4
Link: s1-s3 <=> output port: 5
Link: s1-s4 <=> output_port: 6

Select 4,5,6 output numbers that refer to links used for load-balancing: 4 6
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=3.601s, table=0, n_packets=6, n_bytes=588, idle_age=0,
  hard_age=0, priority=100,ip,nw dst=10.0.0.5 actions=output:6
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=4.623s, table=0, n_packets=7, n_bytes=686, idle_age=0,
  hard_age=0, priority=100,ip,nw dst=10.0.0.5 actions=output:4
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=0.066s, table=0, n_packets=0, n_bytes=0, idle_age=0, pr
  iority=100,ip,nw dst=10.0.0.5 actions=output:6
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=1.126s, table=0, n_packets=1, n_bytes=98, idle_age=1, h
  ard_age=0, priority=100,ip,nw dst=10.0.0.5 actions=output:4
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=2.154s, table=0, n_packets=3, n_bytes=294, idle_age=0,
  hard_age=0, priority=100,ip,nw dst=10.0.0.5 actions=output:6
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=3.188s, table=0, n_packets=4, n_bytes=392, idle_age=0,
  hard_age=0, priority=100,ip,nw dst=10.0.0.5 actions=output:4
Dumping flow entry for destination address 10.0.0.5:
  cookie=0x0, duration=4.224s, table=0, n_packets=5, n_bytes=490, idle_age=0,
  hard_age=0, priority=100,ip,nw dst=10.0.0.5 actions=output:6
Dumping flow entry for destination address 10.0.0.5:

student@openflow: ~/mininet/zsut
Link s1-s3:0
Link s1-s4:2324

Traffic transition per second in bytes:
Link s1-s2:1568
Link s1-s3:0
Link s1-s4:2324

Traffic transition per second in bytes:
Link s1-s2:1666
Link s1-s3:0
Link s1-s4:2324

Traffic transition per second in bytes:
Link s1-s2:1764
Link s1-s3:0
Link s1-s4:2422
[]

student@openflow: ~/pxos
[2024-4-21]08.30.52 s1_p6(Sent): 0 s4_p1(Received): 0
[2024-4-21]08.30.52 s1_p4(Sent): 0 s2_p1(Received): 1
[2024-4-21]08.30.52 s1_p4(Sent): 0 s2_p1(Received): 0
[2024-4-21]08.30.53 s1_p5(Sent): 0 s3_p1(Received): 0
[2024-4-21]08.30.53 s1_p6(Sent): 0 s4_p1(Received): 1
[2024-4-21]08.30.53 s1_p6(Sent): 0 s4_p1(Received): 0
```

Rysunek 3: Konfiguracja łączy s1-s2 oraz s1-s4 dla load-balancingu.

Rysunek 4: Konfiguracja łączy s1-s3 oraz s1-s4 dla load-balancingu.

Rysunek 5: Konfiguracja wszystkich łączów dla load-balancingu

2.2 Test2 - implementacja intentu dla przepływu określonej ilości bajtów w ciągu sekundy

Drugi eksperyment dotyczył badania/monitorowania ilości bajtów przesłanych przez dane łącze w ciągu 1 sekundy. Stworzono plik monitorujący stan ścieżki, która przesyłany domyślnie jest flow dla określonego adresu docelowego. Monitorowanie stanu opiera się na opracowanym algorytmie, który mierzy ilość bajtów jako różnicę między ilością bajtów przesłanych w sekundzie o wartości np. "t" oraz "t+1". Zliczona ilość bajtów jest cyklicznie co sekundę wyświetiana przez program monitorujący, co widać na rysunku 8,9 czy 10.

Na rysunku 6 widnieje kod źródłowy dla pakietów okresowo co interval czasu wysyłane do hosta 10.0.0.5. W każdej interakcji rozmiar pakietu jest losowo zmieniany. Niestandardowy sposób wykonania operacji ping ma jedynie charakter eksperymentalny. Podczas testów, po kilku/kilkunastu sekundach wysyłany pakiet przekraczał swoim rozmiarem ustalony próg threshold.

```
GNU nano 2.5.3                                         File: h5-h1-ping.sh

#!/bin/bash

destination="10.0.0.5" # Destination IP address for ping
interval=0.1 # Interval between pings in seconds

while true; do
    packet_size=$(( ( RANDOM % 1436 ) + 64 )) #define packet size - this parameter changes every interval time

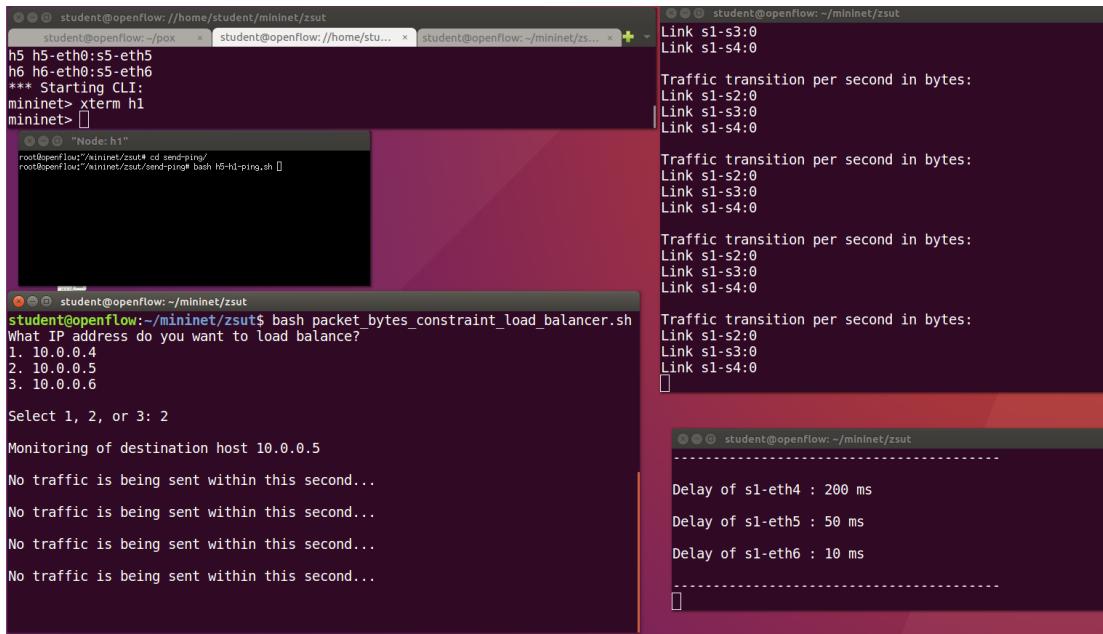
    echo "Sending ping with packet size: $packet_size bytes"

    ping -c 1 -s $packet_size $destination #send one ping within one interval time

    sleep $interval # Wait for the next iteration
done
```

Rysunek 6: Plik wysyłający pingi z h1 do h5 o zmiennym rozmiarze pakietu.

Na rysunku 7 przedstawiono zachowanie programu monitorującego, gdy przez monitorowane dane łącze nie jest przesyłany jeszcze żaden ruch sieciowy.



Rysunek 7: Widok przed rozpoczęciem przesyłania pakietów.

Na rysunku 8 następuje moment, gdy na monitorowanym łączu przesłane zostają pierwszy bajty. Pakiety kierowane do hosta docelowego o adresie 10.0.0.5 są przesyłane łączem s1-s4, czyli wychodzą portem wyjściowym

eth6 - tak jak to zostało ustalone na początku w pliku routing_controller.

```
student@openflow: ~/pox      student@openflow: ~/home/student/mininet/zsut      student@openflow: ~/home/stu...      student@openflow: ~/mininet/zs...
student@openflow: ~/pox      student@openflow: ~/home/student/mininet/zsut      student@openflow: ~/home/stu...      student@openflow: ~/mininet/zs...
h5 h5-eth0:0:5-eth5          Link s1-s3:0
h6 h6-eth0:0:5-eth6          Link s1-s4:42
*** Starting CLI:           Traffic transition per second in bytes:
mininet> xterm h1           Link s1-s2:0
mininet> [                  Link s1-s3:0
mininet> [                  Link s1-s4:84
mininet> [ "Node: h1"
mininet> [ "Node: h1"
I packet transmitted, 0 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 245.704/235.704/245.704/0.000 ms
Sending ping with packet size: 75 bytes
PING 10.0.0.5 (10.0.0.5) 73(101) bytes of data,
81 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=234 ms
--- 10.0.0.5 ping statistics ---
8 packets transmitted, 0 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 234.579/234.579/234.579/0.000 ms
Sending ping with packet size: 1225 bytes
PING 10.0.0.5 (10.0.0.5) 1225(1255) bytes of data.
[

student@openflow: ~/mininet/zsut
student@openflow: ~/mininet/zsut
No traffic is being sent within this second...
Traffic on the specified link is within normal range => bytes_per_second 42
Traffic on the specified link is within normal range => bytes_per_second 42
Traffic on the specified link is within normal range => bytes_per_second 42
Traffic on the specified link is within normal range => bytes_per_second 1936
Traffic on the specified link is within normal range => bytes_per_second 1918
[

student@openflow: ~/mininet/zsut
-----
Delay of s1-eth4 : 200 ms
Delay of s1-eth5 : 50 ms
Delay of s1-eth6 : 10 ms
-----
```

Rysunek 8: Początkowy ruch kierowany przez domyślne łącze.

Na rysunku 9 zobrazowano moment, gdy ilość bajtów przesyłanych na sekundę łączem s1-s4 przekroczyła ustaloną wartość progową, czego późniejszym rezultatem była aktywacja procedury load-balancingu i zrównoważenie obciążenia pomiędzy inne aktywne łącza.

Rysunek 9: Aktywacja load-balancigu po przekroczeniu wartości threshold.

2.3 Test3 - implementacja intentu dla określonej wartości opóźnienia na łączu

Ostatni eksperyment polegał na wdrożeniu mechanizmu load-balancingu na podstawie zdefiniowanych intentów zmieniających wartość opóźnienia na łączu, przez które przesyłany jest ruch sieciowy. Ustalono wartość threshold opóźnienia = 60ms, po której przekroczeniu zostanie aktywowana procedura load-balancingu.

Na rysunku 10 widnieje kod źródłowy, który odpowiada za automatyczną zmianę opóźnienia na monitorowanym łączu - tutaj jest to łącze s1-s3 (output port: s1-eth5). Łączem tym kierowany jest ruch do docelowego hosta o adresie 10.0.0.6 na podstawie którego monitorowany jest przepływ w tej części eksperymentu. Do modyfikacji opóźnienia wykorzystano mechanizm "traffic control". Nowa wartość opóźnienia podawana jest przy wywoływaniu skryptu .sh na poziomie minineta przez switch s1 (Rys 11 - lewy górny róg ekranu) Modyfikowana zmiana opóźnienia jest na bieżąco wyświetlana w programie monitorującym stan danego łączka (Rys 11 lewa strona ekranu) oraz w programie sprawdzającym na bieżąco wartości opóźnień na wszystkich wychodzących łączach switcha s1 w kierunku hostów h4,h5,h6 (Rys 11 prawa dolna strona ekranu).

```
GNU nano 2.5.3                                         File: configDelay_port5.sh

#!/bin/bash

$eth_out=s1-eth5

tc qdisc del dev $eth_out root

tc qdisc add dev $eth_out root netem delay $1

output=$(tc qdisc show dev $eth_out)
delay_value=$(echo "$output" | grep -oP 'delay \K\d+\.\d+' | awk '{printf "%.0f", $1}')
echo -e "$delay_value" > getCurrentDelay.txt
```

Rysunek 10: Plik konfiguracyjny do zmiany opóźnienia na łączu.

The screenshot shows a terminal window with several tabs open. The active tab displays a configuration script for a network interface:

```
student@openflow: ~/home/student/mininet/zsut
student@openflow: ~pox  x student@openflow: ~/home/stu...  x student@openflow: ~/mininet/zsut...  +
```

```
mininet> s1 config-delay/configDelay_port5.sh 51ms
mininet> s1 config-delay/configDelay_port5.sh 52ms
mininet> s1 config-delay/configDelay_port5.sh 55ms
mininet> s1 config-delay/configDelay_port5.sh 57ms
mininet> [REPLACEMENT]
```

Below this, a ping command is shown:

```
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=4 ttl=64 time=282 ns
64 bytes from 10.0.0.6: icmp_seq=5 ttl=64 time=289 ns
64 bytes from 10.0.0.6: icmp_seq=6 ttl=64 time=279 ns
64 bytes from 10.0.0.6: icmp_seq=7 ttl=64 time=290 ns
64 bytes from 10.0.0.6: icmp_seq=8 ttl=64 time=290 ns
64 bytes from 10.0.0.6: icmp_seq=9 ttl=64 time=314 ns
64 bytes from 10.0.0.6: icmp_seq=10 ttl=64 time=295 ns
64 bytes from 10.0.0.6: icmp_seq=11 ttl=64 time=295 ns
64 bytes from 10.0.0.6: icmp_seq=12 ttl=64 time=287 ns
64 bytes from 10.0.0.6: icmp_seq=13 ttl=64 time=291 ns
```

Following this, a series of "Delay is acceptable" messages are printed:

```
Delay is acceptable => current_delay=51
Delay is acceptable => current_delay=51
Delay is acceptable => current_delay=52
Delay is acceptable => current_delay=55
Delay is acceptable => current_delay=57
Delay is acceptable => current_delay=57
Delay is acceptable => current_delay=57
```

To the right of the terminal, three separate windows show traffic statistics for different links:

- Link s1-s3:1050**: Traffic transition per second in bytes: Link s1-s2:0, Link s1-s3:1148, Link s1-s4:0
- Link s1-s2:0**: Traffic transition per second in bytes: Link s1-s2:0, Link s1-s3:1246, Link s1-s4:0
- Link s1-s4:0**: Traffic transition per second in bytes: Link s1-s2:0, Link s1-s3:1344, Link s1-s4:0

At the bottom right, a window displays current delay values:

```
Delay of s1-eth4 : 200 ms
Delay of s1-eth5 : 57 ms
Delay of s1-eth6 : 10 ms
```

Rysunek 11: Moment, gdy opóźnienie na danym łączu nie przekracza wartości progowej 60ms.

Na rysunku 12 przedstawiono moment, gdzie w chwili ustawienia nowej wartości opóźnienia przekraczającej próg opóźnienia threshold, program monitorujący błyskawicznie reaguje i uruchamia mechanizm równoważenia

obciążenia. Warto zauważyc, że load-balancing nie obejmuje wszystkich łączy. Co więcej podczas wywoływanie pliku monitorującego, nie podawaliśmy przez które łącza ma przechodzić ruch sieciowy w trakcie aktywnego load-balancingu. W tej części eksperymentu program monitorujący został dodatkowo uzupełniony o funkcje, która cyklicznie monitoruje stan opóźnień na łączach wychodzących switcha s1. Ta funkcja zachowuje się dokładnie tak samo, jak wyświetlony na Rys11/Rys12 program w prawym dolnym rogu ekranu, o którym wcześniej wspominaliśmy. Tak działająca funkcja pozwala programowi monitorującemu, na sprawdzenie czy opóźnienie na pozostałych aktywnych łączach nie przekracza ustalonej wartości progowej opóźnienia threshold = 60ms. Ustaliliśmy, że początkowo na łączu s1-s2 opóźnienie wynosić ma 200ms, co jest większe od 60ms. W takiej sytuacji program monitorujący nie pozwala na to by ruch był równoważony łączem o wartości opóźnienia większej od wartości threshold. Jedynym łączem spełniającym kryteria jest łącze s1-s4 (output port=6). Jak możemy zauważyc program monitorujący właśnie tym łączem przesyła ruch podczas, gdy pozostałe łącza pozostają nieaktywne do momentu, gdy ich opóźnienie znowu nie będzie mniejsze od wartości progowej.

```
student@openflow:~/home/student/mininet/zsut
student@openflow:~/.pox  x student@openflow:~/home/stu...  x student@openflow:~/mininet/zsut...
mininet> s1 config-delay/configDelay_port5.sh 52ms
mininet> s1 config-delay/configDelay_port5.sh 55ms
mininet> s1 config-delay/configDelay_port5.sh 57ms
mininet> s1 config-delay/configDelay_port5.sh 60ms
mininet>

"Node: h1"
64 bytes from 10.0.0.8: icmp_seq=119 ttl=64 time=331 ns
64 bytes from 10.0.0.8: icmp_seq=120 ttl=64 time=265 ns
64 bytes from 10.0.0.8: icmp_seq=121 ttl=64 time=201 ns
64 bytes from 10.0.0.8: icmp_seq=122 ttl=64 time=200 ns
64 bytes from 10.0.0.8: icmp_seq=123 ttl=64 time=209 ns
64 bytes from 10.0.0.8: icmp_seq=124 ttl=64 time=216 ns
64 bytes from 10.0.0.8: icmp_seq=125 ttl=64 time=236 ns
64 bytes from 10.0.0.8: icmp_seq=126 ttl=64 time=223 ns
64 bytes from 10.0.0.8: icmp_seq=127 ttl=64 time=246 ns
64 bytes from 10.0.0.8: icmp_seq=129 ttl=64 time=238 ns
64 bytes from 10.0.0.8: icmp_seq=130 ttl=64 time=295 ns
[...]
student@openflow:~/mininet/zsut
Delay is acceptable => current_delay=57
High delay detected on the specified link => current_delay=60>59
High delay detected on the specified link => current_delay=60>59

student@openflow:~/mininet/zsut
Link s1-s3:12628
Link s1-s4:0

Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:12628
Link s1-s4:98

Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:12628
Link s1-s4:196

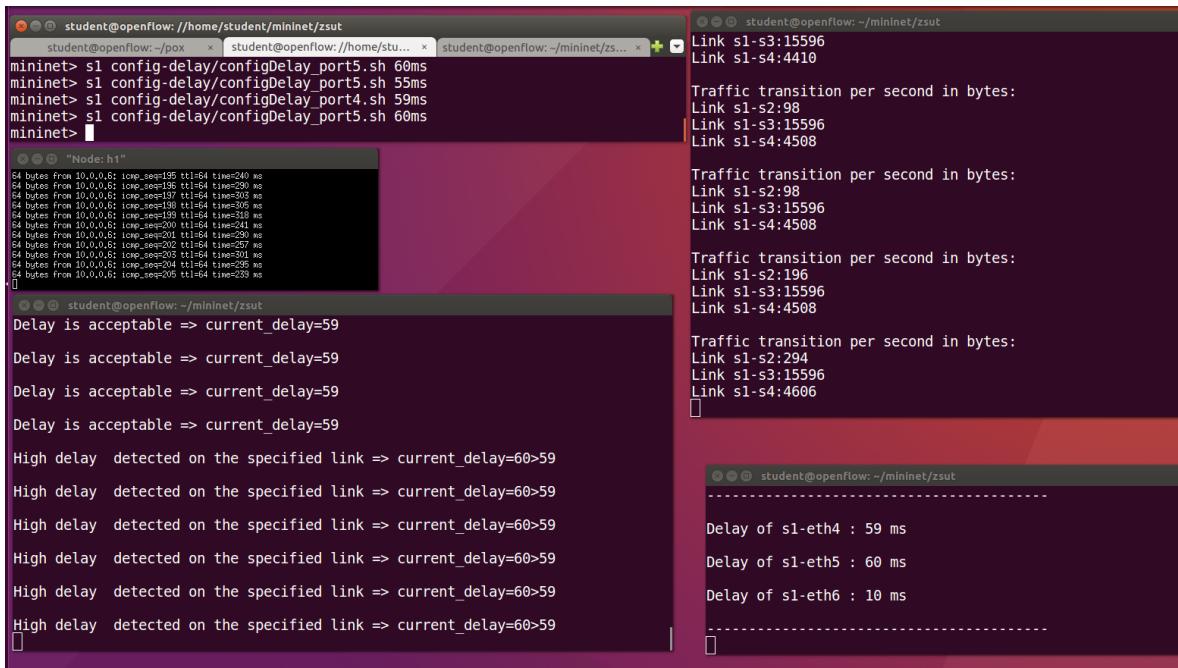
Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:12628
Link s1-s4:294

Traffic transition per second in bytes:
Link s1-s2:0
Link s1-s3:12726
Link s1-s4:294

[...]
student@openflow:~/mininet/zsut
-----
Delay of s1-eth4 : 200 ms
Delay of s1-eth5 : 60 ms
Delay of s1-eth6 : 10 ms
[...]
```

Rysunek 12: Moment przekroczenia wartości progowej opóźnienia na danym łączu - aktywacja load-balancingu.

Na rysunku 13 zobrazowano sytuację, gdzie została zmodyfikowana wartość opóźnienia na łączu s1-s2, która początkowo wynosiła 200ms i niespełniała kryteriów do wartości poniżej wartości progowej. Na obrazku możemy zauważyc, że nowa wartość opóźnienia łącza s1-s2 wynosi 59ms. Program monitorujący stan przepływów do hosta h5 (10.0.0.5) nie dostrzega tej zmiany ponieważ ma on za zadanie śledzić dany przepływ oraz opóźnienie łącza ścieżki domyślnej dla rozważanego przepływu. Zmiana opóźnienia na łączu s1-s2 z kolei widoczna jest w programie w prawej dolnej części ekranu - wartość opóźnienia s1-eth4. Za sprawą tej modyfikacji doszło do zmiany w kontekście łącz biorących udział w procedurze load-balancingu. Nieaktywne wcześniej łącze s1-s2, teraz spełnia kryteria dotyczące opóźnienia i tym samym również obsługuje ruch z domyślnej ścieżki (Rys 13. prawa górną część ekranu - pole Link s1-s2: ilość bajtów przesłanych).



Rysunek 13: Zmiana opóźnienia z 200ms na łączu s1-s2 - load-balancing na wszystkich łączach.

3 Podsumowanie

Projekt pozwolił nam na uzmysławienie sobie tego, jak implementować różne polityki sterowania ruchem, od najprostszych, po - z każdym kolejnym przeprowadzonym testem - coraz bardziej skomplikowane. Wprowadzone przez nas reguły pozwoliły w pewien sposób zoptymalizować ruch w sieci, chociaż jesteśmy jednocześnie świadomi, że aplikacje stosowane profesjonalnie obsługują dużo więcej (i znacznie bardziej skomplikowanych) sytuacji oraz intentów z nimi związanych. Co do samego projektu, uważamy, że dobrze byłoby, gdybyśmy korzystali z nowszej wersji Pythona. Jest to między innymi powód, dla którego przy projekcie w dużym stopniu korzystaliśmy ze skryptów bashowych.