SPIW Projekt 1:
QoS routing w SDN

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Warszawa, 12.04.2023 r.

—

Autorzy: Michał Fedorczyk
Michał Adamczyk
Piotr Budzyński
Jakub Pasierb

Kierunek: Telekomunikacja 5. semestr
Przedmiot: SPIW

Spis treści

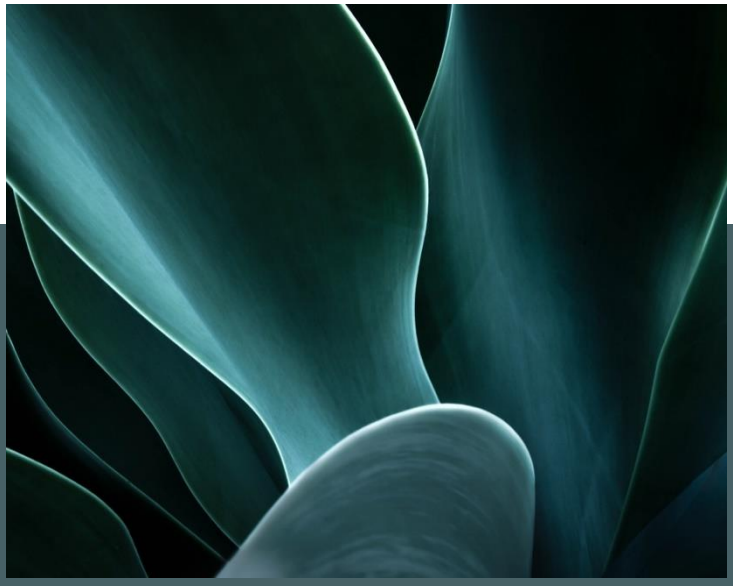
Zadanie 1……...3
Treść zadania.....3

Zadanie 2……...6
Treść zadania.....6

Zadanie 3……..……..9
Treść zadania.....9
Opis skryptu.....9
Wykresy wykresów błędów.....9
Wnioski.....10

Listing kodu……..11

Bibliografia i netografia……..……..13



Opis elementów projektu

Opis „Intent”:

W naszym kontrolerze zdecydowaliśmy się na wyspecyfikowanie dwóch celów działania sieci, które są możliwe do uruchomienia na podczas tworzenia sieci jak i w trakcie działania kontrolera:

1. Opóźnienie e2e flow h1-h6 nie przekracza 200 ms, jeśli jest to możliwe.
2. Opóźnienie e2e flow h2-h5 nie przekracza 100 ms, jeśli jest to możliwe.

Opis topologii:

Topologia jest zgodna z topologią zaproponowaną w instrukcji. Opóźnienia na łączach klarują się następująco:

1. S1-S2 – 250 ms
2. S1-S3 – 50 ms
3. S1-S4 – 100 ms
4. Reszta – 0ms

Opis metody „load-balance”:

Algorytm, który zaimplementowaliśmy w ramach „load-balance” skupia się na rozlokowaniu obciążenia ruchu na podstawie analizy wielkości pakietów z uwzględnieniem aktualnego intentu. Jego działanie można przedstawić krokami:

1. W sieci cały czas (konkretnie co 0.5 sekundy) dokonywane są pomiary opóźnienia na łączach S1-S2/S1-S3/S1-S4.
2. Za każdym razem jak przyjdzie informacja o statystykach switchy (funkcja _handle_portstats_received) sprawdzana jest ilość bajtów odbieranych na porcie 1 dla switchy S2/S3/S4.
3. Co 30 sekund następuje podsumowanie aktualnie zmierzonych opóźnień poprzez obliczenie z nich średniej arytmetycznej i wyznaczenie średniego opóźnienia dla wyżej wskazanych łączy. Dodatkowo sumowana jest ilość bajtów odebranych przez switche S1/S2/S3 tak aby uzyskać całkowitą ilość bajtów jaka przeszła w ciągu okresu pomiarowego (30 sekund).

4. Spośród łączy wybierane jest to z najmniejszym opóźnieniem a następnie przypisywany jest do niego ruch związany z założeniami aktualnego intentu.
5. Do pozostałych łączy przypisujemy ruch dla pozostałych hostów w sposób losowy.
6. Jeśli nie zmieniono intentu realizujemy w kolejności punkty 1/2/3 a następnie porównujemy ilość bitów z punktu 3 dla dwóch łączy (nie bierzemy pod uwagę łącza obsługującego ruch dla intent). Jeżeli na łączu o większym opóźnieniu jest więcej bajtów to zamieniamy wpisy w tablicy rutingu dla tych dwóch łaczy.
   Jeśli zmieniono intentu powtarzamy wszystkie punkty od początku.

Bardziej skrótowo dla łącza o najmniejszym opóźnieniu przypisywany jest ruch zdefiniowany w aktualnym intencie. Dla pozostałych dwóch łaczy ruch jest podzielony tak aby łącze o mniejszym opóźnieniu obsługiwało większy ruch.

Skrypt kontrolera:

Kontroler przechowuje informacje o aktualnej tablicy rutingu dla switchy S1/S5 w specjalnie zdefiniowanej klasie w formie par (adres/port wyjściowy):

```
class RoutingTable:
  def __init__(self, s1_dpid, s5_dpid):
    self.s1_dpid = s1_dpid
    self.s5_dpid = s5_dpid
    self.switch1 = {
      "10.0.0.1" : 1,
      "10.0.0.2" : 2,
      "10.0.0.3" : 3,
      "10.0.0.4" : 4,
      "10.0.0.5" : 5,
      "10.0.0.6" : 6}
    self.switch5 = {
      "10.0.0.1" : 1,
      "10.0.0.2" : 2,
      "10.0.0.3" : 3,
      "10.0.0.4" : 4,
      "10.0.0.5" : 5,
      "10.0.0.6" : 6}

  def change_routes(self, dpid, address, port):
    if(dpid == self.s1_dpid):
      self.switch1[address] = port
    elif(dpid == self.s5_dpid):
      self.switch5[address] = port
```

Informacje o dostępnych intentach oraz o aktualnie wybranym przechowywane są w klasie:

```
class Intent:
  bandwidth_latency = 200  # Maximal bandwidth latency for connection Mb/s
  src_address = "10.0.0.1" # Source address
  dst_address = "10.0.0.6" # Destination address
  intent = 1

  @classmethod
  def change_intent(cls, intent):
    cls.intent = intent
    if(intent == 1):
      cls.bandwidth_latency = 200
      cls.bytes_amount = 20000000
      cls.src_address = "10.0.0.1"
      cls.dst_address = "10.0.0.6"
    elif(intent == 2):
      Intent.bandwidth_latency = 100
      Intent.bytes_amount = 10000000
      Intent.src_address = "10.0.0.2"
      Intent.dst_address = "10.0.0.5"
```

Ustawienia pozostałych ścieżek w zależności od aktualnego intentu znajduja się w klasie:

```
class AvailableRoutes:
  Intent_src_address = "10.0.0.1"
  Intent_dst_address = "10.0.0.6"
  Intent_src_port = 5
  Intent_bw_port = 2

  Normal1_src_address = "10.0.0.1"
  Normal1_dst_address = "10.0.0.6"
  Normal1_src_port = 5
  Normal1_bw_port = 2

  Normal2_src_address = "10.0.0.1"
  Normal2_dst_address = "10.0.0.6"
  Normal2_src_port = 5
  Normal2_bw_port = 2

  @classmethod
  def set_available(cls):
```

```
    if Intent.intent == 1:
      cls.Normal1_src_address = "10.0.0.2"
      cls.Normal1_dst_address = "10.0.0.5"
      cls.Normal1_src_port = 6
      cls.Normal1_bw_port = 3

      cls.Normal2_src_address = "10.0.0.3"
      cls.Normal2_dst_address = "10.0.0.4"
      cls.Normal2_src_port = 4
      cls.Normal2_bw_port = 1
    elif Intent.intent == 2 or Intent.intent == 3:
      cls.Normal1_src_address = "10.0.0.1"
      cls.Normal1_dst_address = "10.0.0.6"
      cls.Normal1_src_port = 6
      cls.Normal1_bw_port = 3

      cls.Normal2_src_address = "10.0.0.3"
      cls.Normal2_dst_address = "10.0.0.4"
      cls.Normal2_src_port = 4
      cls.Normal2_bw_port = 1
```

Linki S1-S2/S1-S3/S1-S4 zdefiniowane są jako instancje klasy, w której można znaleźć informacje o policzonych opóźnieniu na łączu, ilości bajtów jakie przeszły przez łącze oraz o portach wykorzystywanych w łączu:

```
class Link:    # Class to store which port to choose for every dst address
  def __init__(self, src_dpid, dst_dpid, src_port, bw_src_port):
    self.src_dpid = src_dpid
    self.dst_dpid = dst_dpid
    self.src_port = src_port
    self.bw_src_port = bw_src_port
    self.bytes_amount = 0
    self.previous_bytes_amount = 0
    self.bandwidth_latency = 0
```

Dokonaliśmy wielu pomiarów i okazało się, że opóźnienie mierzone na łączach przez kontroler jest zawyżone w porównaniu do rzeczywistego (może to wynikać z dodatkowego czasu wykonywania pomiarów przez software, niedokładność funkcji pobierających obecny czas itp). Dlatego też zdecydowaliśmy się na manualną korekcję mierzonych opóźnień. Z nasz pomiarów wynika, że opóźnienia powyżej 100 Mb/s zawyżone są o około 30 Mb/s, powyżej 50 Mb/s o 20 Mb/s a opóźnienia poniżej 50 Mb/s o około 7 Mb/s. Korekcji dokonujemy w funkcji:

```
def correct_measures_delay(delay):
  if delay > 100:
    delay = delay - 30
  elif delay > 50:
    delay = delay - 20
  else:
    delay = delay - 7
  return delay
```

Opóźnienie mierzone jest w ten sam sposób co w pliku delay_controller.py. Ilość przepływających bitów mierzona jest natomiast w funkcji _handle_portstats_received korzystając z parametru rx_bytes, np. dla switcha2:

```
if event.connection.dpid==s2_dpid:
  for f in event.stats:
    if int(f.port_no)<65534:
      if f.port_no==1:
        pre_s2_p1=s2_p1
        s2_p1=f.rx_packets
        link_s1_s2.bytes_amount=f.rx_bytes-link_s1_s2.previous_bytes_amount
  print getTheTime(), "s1_p4(Sent):", (s1_p4-pre_s1_p4), "s2_p1(Received):",
  (s2_p1-pre_s2_p1)
```

W funkcji _handle_ConnectionUp ustawiane są wstępne parametry opisujące sieć oraz timery dla pomiarów i konfiguracji tablicy rutingowej:

```
if s1_dpid<>0 and s2_dpid<>0 and s3_dpid<>0 and s4_dpid<>0 and s5_dpid<>0:
    link_s1_s2 = Link(s1_dpid, s2_dpid, 4, 1)
    link_s1_s3 = Link(s1_dpid, s3_dpid, 5, 2)
    link_s1_s4 = Link(s1_dpid, s4_dpid, 6, 3)
    Intent.change_intent(1)
    routing_table = RoutingTable(s1_dpid, s5_dpid)

    Timer(0.5, measure_delays, recurring=True)
    Timer(30, set_routing, recurring=True)
    Timer(60, switch_intent, recurring=True)
```

W funkcji _handlePacketIn wszystkie wiadomości jakie są wysyłane do switchy w związku z ustawieniem tablic rutingowych korzystają z informacji zapisanych w klasie Routing_Table.

Wyznaczanie tablic rutingowych odbywa się w funkcji set_routing (wywoływanej co 30 sekund). Najpierw sprawdza się czy nastąpiła zmiana intentu lub czy intent przypisany jest do łącza o najmniejszym opóźnieniu oraz konfiguruje się ruch dla trasy opisanej w intencie:

```
# ***************** Setting route for Intent **************
 links_delays = [link_s1_s2, link_s1_s3, link_s1_s4]
 links_delays.sort(key=lambda x: x.bandwidth_latency)
 intent_set = False

 print "Intent dst adr:", Intent.dst_address, " src port:",
Intent.src_address

 if links_delays[0].bandwidth_latency < Intent.bandwidth_latency and
links_delays[0].src_port != routing_table.switch1[Intent.dst_address]  and
links_delays[0].bw_src_port != routing_table.switch5[Intent.src_address]:
    print "******** Intent Changed ********"
    routing_table.change_routes(s1_dpid, Intent.dst_address,
links_delays[0].src_port)
    routing_table.change_routes(s5_dpid, Intent.src_address,
links_delays[0].bw_src_port)
```

Ustawienia pozostałych łączy:

```
links_delays.pop(0)   # Removing link for intent network traffic
links_bytes = [links_delays[0], links_delays[1]]
links_bytes.sort(key=lambda x: x.bytes_amount)

 if not intent_set:  # If intent was set in this turn dont change other routes
   if links_delays[0].bandwidth_latency == links_bytes[0].bandwidth_latency:
      print "******** Another Routes Changed ********"
      # Second host
      routing_table.change_routes(s1_dpid,
AvailableRoutes.Normal1_dst_address, links_delays[1].src_port)
      routing_table.change_routes(s5_dpid,
AvailableRoutes.Normal1_src_address, links_delays[1].bw_src_port)
      # Third host
      routing_table.change_routes(s1_dpid,
AvailableRoutes.Normal2_dst_address, links_delays[0].src_port)
      routing_table.change_routes(s5_dpid,
AvailableRoutes.Normal2_src_address, links_delays[0].bw_src_port)
```

Skrypty i ich działanie

Skrypt topologia:

Topologia została zaimplementowana analogicznie jak w pliku routing_net.py przy czym dodaliśmy adresy MAC dla każdego switcha oraz umożliwiliśmy personalizację ustawień łączy poprzez wykorzystanie argumentów wywołania topologii (najważniejsze fragmenty zmienione):

```
class MyTopo(Topo):
    "Single switch connected to n hosts."
    def __init__(self, delay1 = 10, delay2 = 10, delay3 = 10, buffor_size =
1000, bandwidth = 1):

        ...
        self.addLink(h3, s1, bw=int(bandwidth), delay='0ms', loss=0,
max_queue_size=int(buffor_size), use_htb=True)
        self.addLink(s1, s2, bw=int(bandwidth), delay= str(delay1) + 'ms',
loss=0, max_queue_size=int(buffor_size), use_htb=True)
        self.addLink(s1, s3, bw=int(bandwidth), delay= str(delay2) + 'ms',
loss=0, max_queue_size=int(buffor_size), use_htb=True)
        self.addLink(s1, s4, bw=int(bandwidth), delay= str(delay3) + 'ms',
loss=0, max_queue_size=int(buffor_size), use_htb=True)
        self.addLink(s2, s5, bw=int(bandwidth), delay='0ms', loss=0,
max_queue_size=int(buffor_size), use_htb=True)
        ...

def perfTest():
    "Create network and run simple performance test"
    if len(sys.argv) > 1:
        topo = MyTopo(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4],
sys.argv[5])
    else:
        topo = MyTopo()
    ...
    s1.setMAC("1:0:0:0:0:1")
    s2.setMAC("1:0:0:0:0:2")
    s3.setMAC("1:0:0:0:0:3")
    s4.setMAC("1:0:0:0:0:4")
    s5.setMAC("1:0:0:0:0:5")
```

Testy kontrolera:

Tworzenie sieci:

```
student@openflow:~/mininet/zsut$ sudo python project_net.py 250 50 100 1000 1
```

Test dostosowania sieci do zmiany intentów

Domyślne tablice rutingowe w switchach są skonfigurowane identycznie jak domyślne tablice rutingowe w pliku routing_controller.py.

Przydział odpowiednich łączy tak aby spełniać aktualnie zdefiniowany intent (na screenach pokazano tylko momenty w których zmieniło się opóźnienie na łączach):

h1 ping h6

```
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
64 bytes from 10.0.0.6: icmp_seq=4 ttl=64 time=796 ms
64 bytes from 10.0.0.6: icmp_seq=5 ttl=64 time=351 ms
64 bytes from 10.0.0.6: icmp_seq=6 ttl=64 time=358 ms
64 bytes from 10.0.0.6: icmp_seq=7 ttl=64 time=370 ms
64 bytes from 10.0.0.6: icmp_seq=8 ttl=64 time=353 ms
64 bytes from 10.0.0.6: icmp_seq=9 ttl=64 time=365 ms
64 bytes from 10.0.0.6: icmp_seq=10 ttl=64 time=368 ms
64 bytes from 10.0.0.6: icmp_seq=11 ttl=64 time=351 ms
```

```
64 bytes from 10.0.0.6: icmp_seq=28 ttl=64 time=369 ms
64 bytes from 10.0.0.6: icmp_seq=29 ttl=64 time=370 ms
64 bytes from 10.0.0.6: icmp_seq=30 ttl=64 time=350 ms
64 bytes from 10.0.0.6: icmp_seq=31 ttl=64 time=353 ms
64 bytes from 10.0.0.6: icmp_seq=32 ttl=64 time=358 ms
64 bytes from 10.0.0.6: icmp_seq=33 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=34 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=35 ttl=64 time=107 ms
64 bytes from 10.0.0.6: icmp_seq=36 ttl=64 time=110 ms
64 bytes from 10.0.0.6: icmp_seq=37 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=38 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=39 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=40 ttl=64 time=101 ms
```

```
64 bytes from 10.0.0.6: icmp_seq=60 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=61 ttl=64 time=113 ms
64 bytes from 10.0.0.6: icmp_seq=62 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=63 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=64 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=65 ttl=64 time=203 ms
64 bytes from 10.0.0.6: icmp_seq=66 ttl=64 time=220 ms
64 bytes from 10.0.0.6: icmp_seq=67 ttl=64 time=201 ms
64 bytes from 10.0.0.6: icmp_seq=68 ttl=64 time=201 ms
64 bytes from 10.0.0.6: icmp_seq=69 ttl=64 time=201 ms
64 bytes from 10.0.0.6: icmp_seq=70 ttl=64 time=201 ms
64 bytes from 10.0.0.6: icmp_seq=71 ttl=64 time=209 ms
```

h2 ping h5

```
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=1364 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=336 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=100 ms
64 bytes from 10.0.0.5: icmp_seq=6 ttl=64 time=115 ms
64 bytes from 10.0.0.5: icmp_seq=7 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=8 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=9 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=10 ttl=64 time=100 ms
64 bytes from 10.0.0.5: icmp_seq=11 ttl=64 time=102 ms
64 bytes from 10.0.0.5: icmp_seq=12 ttl=64 time=118 ms
```

```
64 bytes from 10.0.0.5: icmp_seq=25 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=26 ttl=64 time=105 ms
64 bytes from 10.0.0.5: icmp_seq=27 ttl=64 time=101 ms
64 bytes from 10.0.0.5: icmp_seq=28 ttl=64 time=110 ms
64 bytes from 10.0.0.5: icmp_seq=29 ttl=64 time=120 ms
64 bytes from 10.0.0.5: icmp_seq=30 ttl=64 time=202 ms
64 bytes from 10.0.0.5: icmp_seq=31 ttl=64 time=200 ms
64 bytes from 10.0.0.5: icmp_seq=32 ttl=64 time=202 ms
64 bytes from 10.0.0.5: icmp_seq=33 ttl=64 time=213 ms
64 bytes from 10.0.0.5: icmp_seq=34 ttl=64 time=206 ms
64 bytes from 10.0.0.5: icmp_seq=35 ttl=64 time=217 ms
64 bytes from 10.0.0.5: icmp_seq=36 ttl=64 time=200 ms
```

```
64 bytes from 10.0.0.5: icmp_seq=56 ttl=64 time=200 ms
64 bytes from 10.0.0.5: icmp_seq=57 ttl=64 time=201 ms
64 bytes from 10.0.0.5: icmp_seq=58 ttl=64 time=221 ms
64 bytes from 10.0.0.5: icmp_seq=59 ttl=64 time=201 ms
64 bytes from 10.0.0.5: icmp_seq=60 ttl=64 time=201 ms
64 bytes from 10.0.0.5: icmp_seq=61 ttl=64 time=213 ms
64 bytes from 10.0.0.5: icmp_seq=62 ttl=64 time=117 ms
64 bytes from 10.0.0.5: icmp_seq=63 ttl=64 time=109 ms
64 bytes from 10.0.0.5: icmp_seq=64 ttl=64 time=112 ms
64 bytes from 10.0.0.5: icmp_seq=65 ttl=64 time=108 ms
64 bytes from 10.0.0.5: icmp_seq=66 ttl=64 time=102 ms
64 bytes from 10.0.0.5: icmp_seq=67 ttl=64 time=102 ms
```

```
[2023-4-26]00.19.24 s1_p4(Sent): 0 s2_p1(Received): 1
[2023-4-26]00.19.24 s1_p5(Sent): 0 s3_p1(Received): 2
[2023-4-26]00.19.25 s1_p6(Sent): 0 s4_p1(Received): 2
Link S1-S2 mean delay:  283.9
Link S1-S3 mean delay:  77.2
Link S1-S4 mean delay:  127.684210526
Link S1-S2 corrected delay:  253.9
Link S1-S3 corrected delay:  57.2
Link S1-S4 corrected delay:  97.6842105263
Intent Latency:  200
Intent dst adr: 10.0.0.6  src port: 10.0.0.1
******** Intent Route Changed ********
Switch_1 DST address:  10.0.0.6   DST port:  5
Switch_5 DST address:  10.0.0.1   DST port:  2
Switch_1 DST address:  10.0.0.5   DST port:  6
Switch_5 DST address:  10.0.0.2   DST port:  3
Switch_1 DST address:  10.0.0.4   DST port:  4
Switch_5 DST address:  10.0.0.3   DST port:  1
Bytes received on S1-S2:  342
Bytes received on S1-S3:  3212
Bytes received on S1-S4:  3352
[2023-4-26]00.19.25 s1_p4(Sent): 0 s2_p1(Received): 1
[2023-4-26]00.19.25 s1_p5(Sent): 0 s3_p1(Received): 2
```

Jako rezultat każdego pomiaru w oknie konsoli kontrolera wyświetlają się informacje o pomiarach oraz aktualnych ustawieniach rutingu.

Po pierwszym pomiarze kontroler wykrył, że odpowiednia droga dla zdefiniowanego intentu nie jest ustawiona, dlatego ją dopasował. Po ustawieniu tablic rutingowych w switchach widać, że opóźnienie h1-h6 zmieniło się z ustawienia domyślnego (około 350 ms RTT) na h1->s1->s3->s5->h6 (około 100 ms RTT - > 50 ms w jedną stronę), tak więc opóźnienie jest najmniejsze możliwe i spełnia wymogi intentu (opóźnienie mniejsze niż 200 ms).
Dla ruchu z h2 do h5 ustawiono drogę h2->s1->s4->s5->h5 (około 200 ms -> 100 ms w jedną stronę).

```
Link S1-S2 mean delay:  279.7
Link S1-S3 mean delay:  84.25
Link S1-S4 mean delay:  114.333333333
Link S1-S2 corrected delay:  249.7
Link S1-S3 corrected delay:  64.25
Link S1-S4 corrected delay:  84.3333333333
Intent Latency:  100
Intent dst adr: 10.0.0.5  src port: 10.0.0.2
******** Intent Route Changed ********
Switch_1 DST address:  10.0.0.5   DST port:  5
Switch_5 DST address:  10.0.0.2   DST port:  2
Switch_1 DST address:  10.0.0.6   DST port:  6
Switch_5 DST address:  10.0.0.1   DST port:  3
Switch_1 DST address:  10.0.0.4   DST port:  4
Switch_5 DST address:  10.0.0.3   DST port:  1
Bytes received on S1-S2:  360
Bytes received on S1-S3:  3636
Bytes received on S1-S4:  3538
```

Kolejny pomiar jest po 60 sekundach, więc nastąpiła zmiana intentu na intent 2.

Kontroler wykrył zmianę i ustawił nową drogę na h2->s1->s3->s5->h5 (około 100 ms -> 50 ms w jedną stronę), która spełnia wymóg opóźnienia maksymalnego 100 Mb/s, ponieważ skorygowana wartość zmierzonego opóźnienia na łączu to 64 Mb/s. Teraz ruch z h1 do h6 realizowany jest na drodze na h1->s1->s4->s5->h6 (około 200 ms -> 100 ms w jedną stronę). Jak widać tablice zostały zmienione tak aby ruch wedle nowego intentu został skierowany na łącze o najmniejszym opóźnieniu

Test dostosowania sieci do zmiany obciążenia na łączach

h1 ping h6

```
64 bytes from 10.0.0.6: icmp_seq=29 ttl=64 time=359 ms
64 bytes from 10.0.0.6: icmp_seq=30 ttl=64 time=350 ms
64 bytes from 10.0.0.6: icmp_seq=31 ttl=64 time=353 ms
64 bytes from 10.0.0.6: icmp_seq=32 ttl=64 time=350 ms
64 bytes from 10.0.0.6: icmp_seq=34 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=35 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=36 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=37 ttl=64 time=126 ms
64 bytes from 10.0.0.6: icmp_seq=38 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=39 ttl=64 time=100 ms
```

h3 ping h4

```
PING 10.0.0.4 (10.0.0.4) 100(128) bytes of data.
108 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=912 ms
108 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=501 ms
108 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=506 ms
108 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=501 ms
108 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=501 ms
108 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=500 ms
108 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=541 ms
108 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=500 ms
108 bytes from 10.0.0.4: icmp_seq=14 ttl=64 time=500 ms
108 bytes from 10.0.0.4: icmp_seq=16 ttl=64 time=505 ms
```

```
[2023-4-26]01.24.07 s1_p5(Sent): 0 s3_p1(Received): 1
Link S1-S2 mean delay:  282.75
Link S1-S3 mean delay:  77.25
Link S1-S4 mean delay:  131.0
Link S1-S2 corrected delay:  252.75
Link S1-S3 corrected delay:  57.25
Link S1-S4 corrected delay:  101.0
Intent Latency:  200
Intent dst adr: 10.0.0.6  src port: 10.0.0.1
******** Intent Route Changed ********
Switch_1 DST address:  10.0.0.6   DST port:  5
Switch_5 DST address:  10.0.0.1   DST port:  2
Switch_1 DST address:  10.0.0.5   DST port:  6
Switch_5 DST address:  10.0.0.2   DST port:  3
Switch_1 DST address:  10.0.0.4   DST port:  4
Switch_5 DST address:  10.0.0.3   DST port:  1
Bytes received on S1-S2:  342
Bytes received on S1-S3:  342
Bytes received on S1-S4:  3530
[2023-4-26]01.24.07 s1_p5(Sent): 0 s3_p1(Received): 1
```

Na początku uruchomiono ruch h1 ping h6

Po pierwszym cyklu pomiarowym ustawiono drogę z najmniejszym opóźnieniem dla intentu 1. Wyznaczone drogi to:
h1->s1->s3->s5->h6 opóźnienie około 57 ms
h2->s1->s4->s5->h5 opóźnienie około 101 ms
h3->s1->s2->s5->h4 opóźnienie około 252 ms

h1 ping h6

```
64 bytes from 10.0.0.6: icmp_seq=71 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=72 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=73 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=74 ttl=64 time=111 ms
64 bytes from 10.0.0.6: icmp_seq=75 ttl=64 time=111 ms
64 bytes from 10.0.0.6: icmp_seq=76 ttl=64 time=106 ms
64 bytes from 10.0.0.6: icmp_seq=77 ttl=64 time=108 ms
64 bytes from 10.0.0.6: icmp_seq=78 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=79 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=80 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=81 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=82 ttl=64 time=117 ms
```

h3 ping h4

```
108 bytes from 10.0.0.4: icmp_seq=24 ttl=64 time=501 ms
108 bytes from 10.0.0.4: icmp_seq=26 ttl=64 time=533 ms
108 bytes from 10.0.0.4: icmp_seq=28 ttl=64 time=507 ms
108 bytes from 10.0.0.4: icmp_seq=30 ttl=64 time=503 ms
108 bytes from 10.0.0.4: icmp_seq=31 ttl=64 time=201 ms
108 bytes from 10.0.0.4: icmp_seq=32 ttl=64 time=203 ms
108 bytes from 10.0.0.4: icmp_seq=33 ttl=64 time=206 ms
108 bytes from 10.0.0.4: icmp_seq=35 ttl=64 time=201 ms
108 bytes from 10.0.0.4: icmp_seq=37 ttl=64 time=208 ms
```

h2 ping h5

```
PING 10.0.0.5 (10.0.0.5) 1000(1028) bytes of data.
1008 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=747 ms
1008 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=504 ms
1008 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=501 ms
1008 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=505 ms
1008 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=502 ms
1008 bytes from 10.0.0.5: icmp_seq=6 ttl=64 time=504 ms
1008 bytes from 10.0.0.5: icmp_seq=7 ttl=64 time=542 ms
1008 bytes from 10.0.0.5: icmp_seq=8 ttl=64 time=525 ms
1008 bytes from 10.0.0.5: icmp_seq=9 ttl=64 time=520 ms
1008 bytes from 10.0.0.5: icmp_seq=10 ttl=64 time=501 ms
```

```
Link S1-S2 mean delay:  279.75
Link S1-S3 mean delay:  69.85
Link S1-S4 mean delay:  135.857142857
Link S1-S2 corrected delay:  249.75
Link S1-S3 corrected delay:  49.85
Link S1-S4 corrected delay:  105.857142857
Intent Latency:  200
Intent dst adr: 10.0.0.6  src port: 10.0.0.1
Bytes received on S1-S2:  2716
Bytes received on S1-S3:  3398
Bytes received on S1-S4:  518
******** Another Routes Changed ********
Switch_1 DST address:  10.0.0.5   DST port:  4
Switch_5 DST address:  10.0.0.2   DST port:  1
Switch_1 DST address:  10.0.0.4   DST port:  6
Switch_5 DST address:  10.0.0.3   DST port:  3
[2023-4-26]01.24.39 s1_p6(Sent): 0 s4_p1(Received): 1
```

Po pierwszym cyklu pomiarowym uruchomiono h3 ping -s 100 h4

W drugim cyklu pomiarowym droga zdefiniowana dla intentu pozostaje bez zmian. Kontroler wykrył natomiast, że łącze S1-S2 jest obciążone mocniej niż S1-S4 mimo że ma większe opóźnienie. Drogi skonfigurowano tak aby ruch szedł przez łącze o mniejszym opóźnieniu. Wyznaczone drogi to:
h1->s1->s3->s5->h6 opóźnienie około 49 ms
h2->s1->s2->s5->h5 opóźnienie około 249 ms
h3->s1->s4->s5->h4 opóźnienie około 105 ms

h1 ping h6

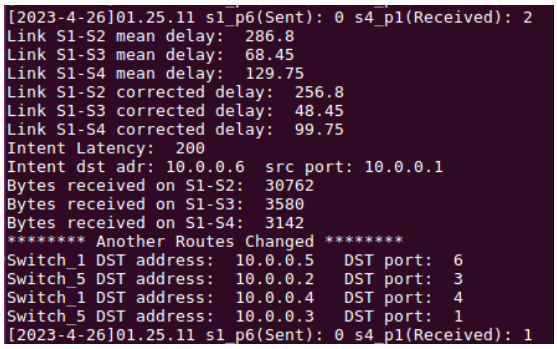
```
64 bytes from 10.0.0.6: icmp_seq=99 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=100 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=101 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=102 ttl=64 time=124 ms
64 bytes from 10.0.0.6: icmp_seq=103 ttl=64 time=104 ms
64 bytes from 10.0.0.6: icmp_seq=104 ttl=64 time=114 ms
64 bytes from 10.0.0.6: icmp_seq=105 ttl=64 time=102 ms
64 bytes from 10.0.0.6: icmp_seq=106 ttl=64 time=101 ms
64 bytes from 10.0.0.6: icmp_seq=107 ttl=64 time=110 ms
64 bytes from 10.0.0.6: icmp_seq=108 ttl=64 time=100 ms
64 bytes from 10.0.0.6: icmp_seq=109 ttl=64 time=101 ms
```

h3 ping h4

```
108 bytes from 10.0.0.4: icmp_seq=56 ttl=64 time=201 ms
108 bytes from 10.0.0.4: icmp_seq=57 ttl=64 time=202 ms
108 bytes from 10.0.0.4: icmp_seq=59 ttl=64 time=200 ms
108 bytes from 10.0.0.4: icmp_seq=61 ttl=64 time=201 ms
108 bytes from 10.0.0.4: icmp_seq=63 ttl=64 time=515 ms
108 bytes from 10.0.0.4: icmp_seq=65 ttl=64 time=501 ms
108 bytes from 10.0.0.4: icmp_seq=67 ttl=64 time=507 ms
108 bytes from 10.0.0.4: icmp_seq=69 ttl=64 time=511 ms
108 bytes from 10.0.0.4: icmp_seq=71 ttl=64 time=503 ms
```

h2 ping h5

```
1008 bytes from 10.0.0.5: icmp_seq=28 ttl=64 time=503 ms
1008 bytes from 10.0.0.5: icmp_seq=29 ttl=64 time=501 ms
1008 bytes from 10.0.0.5: icmp_seq=30 ttl=64 time=500 ms
1008 bytes from 10.0.0.5: icmp_seq=31 ttl=64 time=500 ms
1008 bytes from 10.0.0.5: icmp_seq=32 ttl=64 time=212 ms
1008 bytes from 10.0.0.5: icmp_seq=33 ttl=64 time=210 ms
1008 bytes from 10.0.0.5: icmp_seq=34 ttl=64 time=216 ms
1008 bytes from 10.0.0.5: icmp_seq=35 ttl=64 time=203 ms
1008 bytes from 10.0.0.5: icmp_seq=36 ttl=64 time=202 ms
```



Po drugim cyklu pomiarowym uruchomiono h2 ping -s 1000 h5

W trzecim cyklu pomiarowym droga zdefiniowana dla intentu pozostaje bez zmian. Kontroler wykrył natomiast, że łącze S1-S2 jest obciążone mocniej niż S1-S4 mimo że ma większe opóźnienie. Drogi skonfigurowano tak aby ruch szedł przez łącze o mniejszym opóźnieniu. Wyznaczone drogi to:
h1->s1->s3->s5->h6 opóźnienie około 48 ms
h2->s1->s4->s5->h5 opóźnienie około 99 ms
h3->s1->s2->s5->h4 opóźnienie około 256 ms

Podsumowanie

Wnioski itd.:

Zastosowana przez nas metoda load balancingu pozwala wykorzystać łącza o mniejszym opóźnieniu do transmitowania większego ruchu co przekłada na parametry QoS sieci jak np. opóźnienie przychodzących pakietów. Wadą rozwiązania jest fakt, że zmiana tablic rutingowych odbywa się co jeden cykl pomiarowy, czyli w naszym przypadku 30 sekund. Oczywiście czas ten mógł by być skrócony jednak mogło by mieć to negatywny wpływ na wiarygodność mierzonego średniego opóźnienia na łączach.