Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Telekomunikacji

Zakład Sieci i Usług Teleinformatycznych

INTELLIGENT NETWORK SYSTEMS

Project 1: QoS routing with SDN

Jakub Kitka, Andrzej Gawor, Mateusz Pawłowski

student IDs: 300552, 300528, 300488

project supervisor

dr inż. Dariusz Bursztynowski

WARSAW 2023

1. General description

The goal of this project is to gain basic understanding of the role of SDN controller in providing northbound services based on the capabilities offered by the southbound interface.

The controller has some capabilities which allows to dynamically select the routes for traffic flows in accordance with the requirements specified for a given flow and based on estimated link delays in the network.

Decisions taken:

1. Intent's model:

```
class Inent{
        var FROM: enum{H1, H2, H3}
        var TO: enum{H4,H5,H6}
        var LIMIT: Int
}
```

Example:

```
Intent(FROM: H1, TO: H5, LIMIT: 60)
```

The example represents an Intent that says for flow H1->H5 the max delay is expected to be 60ms.

2. Load balancing

Possible flows:

a. H1 – H4

b. H1 – H5

c. H1 – H6

d. H2 – H4

e. H2 – H5

f. H2 – H6

g. H3 – H4

h. H3 – H5

i. H3 – H6

Possible routes:

a. S1 – S2 (route 1)

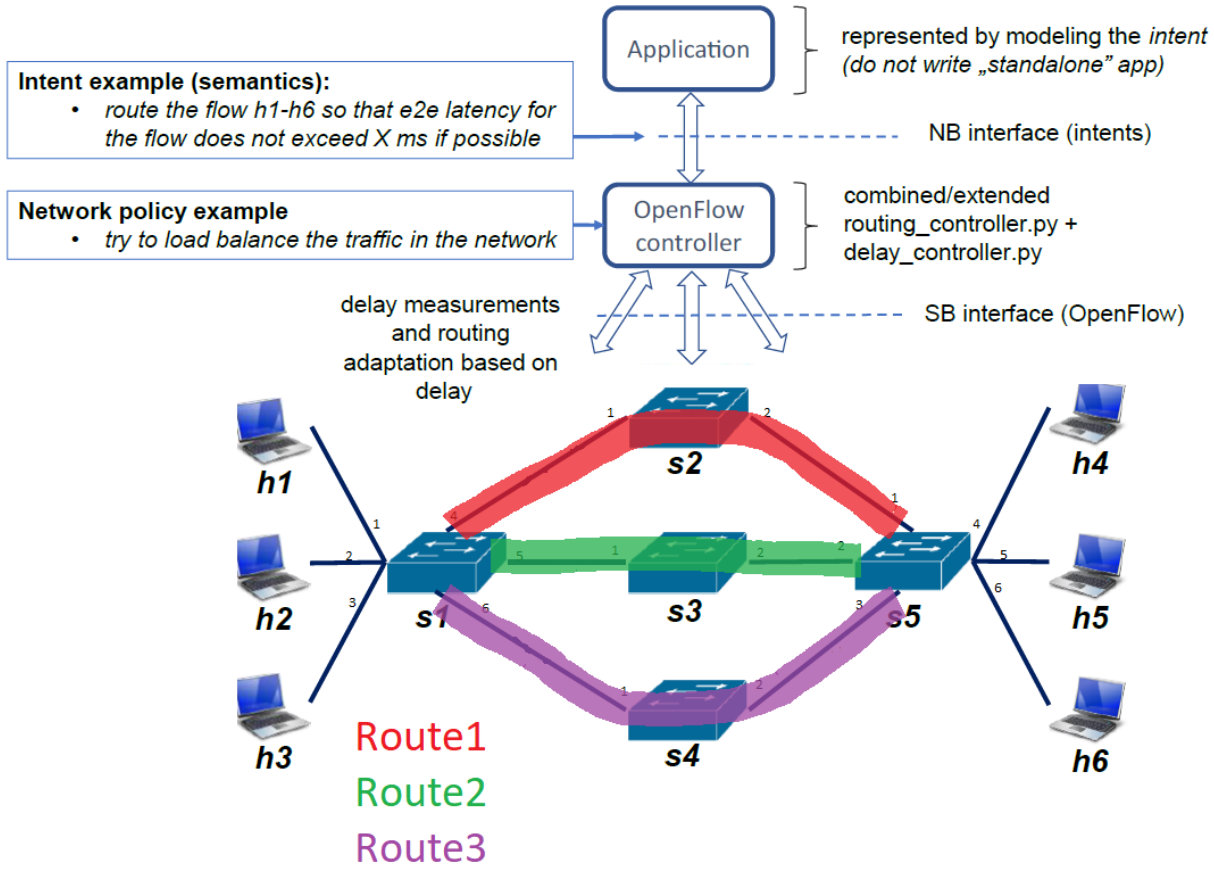
b. S1 – S3 (route 2)

c. S1 – S4 (route 3)

By default, no flow has a route assigned to it. When any flow starts to become non-zero (e.g. we ping), a PACKET_IN message will appear in the Controller. Our job now is to distribute the routes for incoming flows, so that each route will have fairly equal number of flows on it.

In our project we set the counter for each route (1, 2, 3) and give it a value of 0. A flow comes in, then if there is a tie, we give it to Route1 (to S1-S2). Then we make the maximum difference between the counter for each route to be 1. This whole algorithm is in one function, for which the trigger can be either PACKET_IN or INTENT_IN or INTENT_REROUTE. That is, if an Intent comes along and its action "messes us up", i.e. makes a difference of 2 somewhere, then we have to rearrange these default flows. In the same way, if we need to reroute an Intent-controlled flow (because the network traffic has made it so that its current path no longer meets the requirements), we will also need to trigger load balancing.

3. Topology



To sum up, below are the options we have selected in accordance with the project's instruction:

In our project, students are expected do define the load balancing policy. Some examples to select from are as follows:

- equalize the number of active flows on the links in the link set s1-s2/3/4 (more precisely, apply min-max rule to the number of flows on these links)

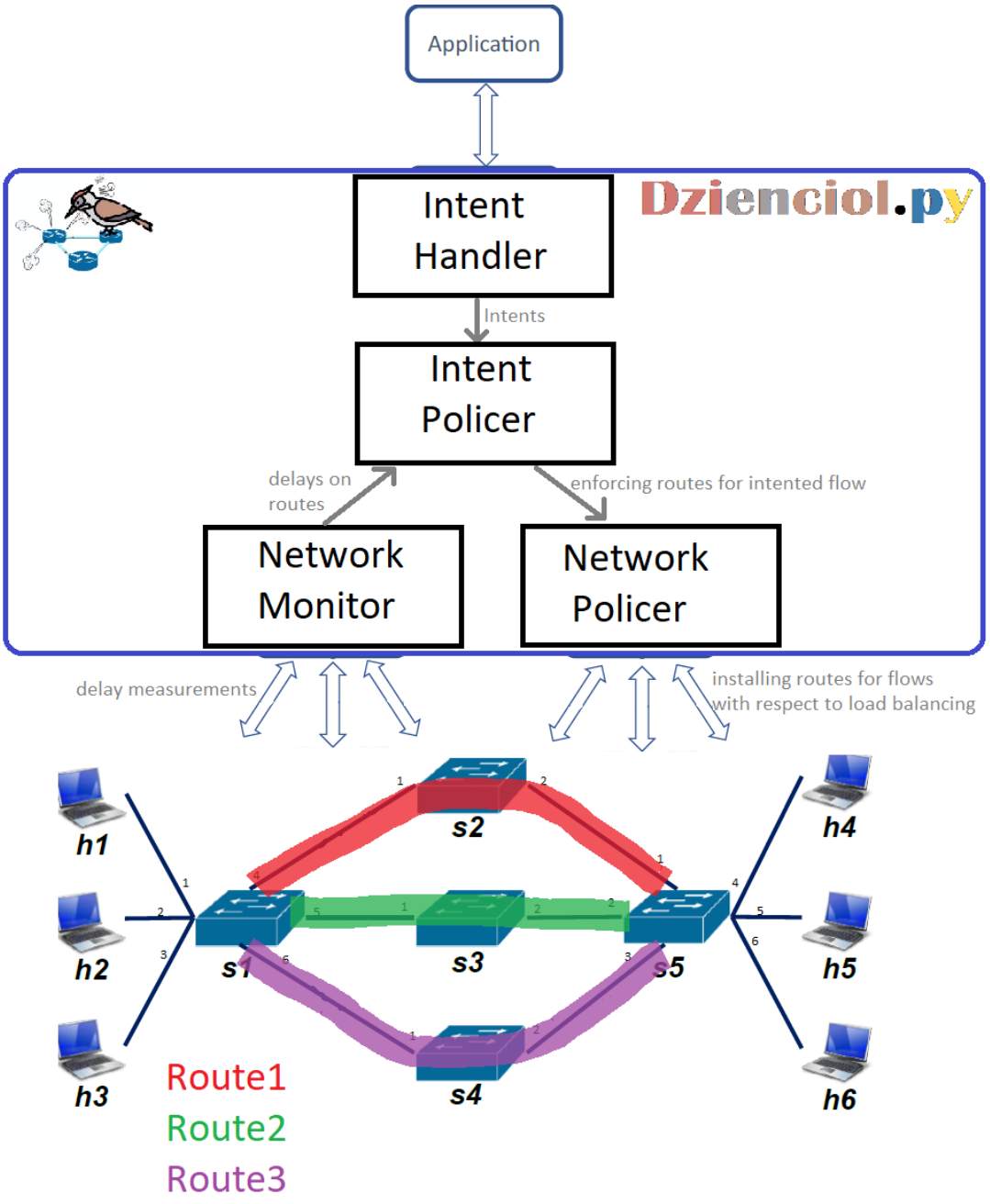
One flow at a time

- choosing one out of the two following design options:
  - ***basic***: it is sufficient that only one flow at a time can be subject to intent-based *assurance* treatment (i.e., watched during it's lifetime and rerouted by the controller application when needed according to the specification given in the intent); nevertheless, new intents can come and be routed according to global (network-level) load balancing policies;

2. The Dzienciol solution

Dzienciol.py is our implementation of an OpenFlow Controller, that stands as a solution to the project problem.

2.1. Dzienciol Architecture



2.2. NetworkPolicer

Installs routes in all switches. NetworkPolicer solely independently manages the non-intented flows. NetworkPolicer is the component responsible for **load balancing** required in the project scope. NetworkPolicer can be requested by the IntentPolicer to force the given route for the intented flow. After such operation, NetworkPolicer needs to trigger its load balancing check, and potentially rearrange the flows distrbution.

2.3. NetworkMonitor

Triggers the delay measurement procedure on routes 1, 2, 3. The procedure is based on the one used in lab1. NetworkMonitor at any given moment has the values of delays on each route and provides interface for such information. The measurement is triggered every 5 seconds.

2.4. IntentPolicer

IntentPolicer is the only component that understands the Intent concept. Our project assumption is to keep track of only one intent. IntentPolicer gets an Intent, then checks appropriate route for it and enforces the route for the intented flow on the NetworkPolicer. After that IntentPolicer monitors intented flow via NetworkMonitor and takes apprioprate actions based on that (e.g., it can reroute the intented flow).

2.5. IntentHandler

Listens for the intent request from the external application. IntentHandler is implemented as a TCP socket server (python “socket” lib). We can send an Intent as a string from our simulated application (script `application.py`). IntentHandler and application.py are deployed in a client-server manner.

3. Demo

3.1. Test the NetworkPolicer load balancing

In this scenario we will use only non-intented flows. They will be generated with ping commands. The subject of the test is to check if NetworkPolicer distributes the routes along to its load balancing policy. It means that each route has to be loaded (in a number of flows on it) equally. The only acceptable difference is 1. Let’s check it out.

We’ve started a non-intented non-zero flow from host H1 to host H4.

```
*** Starting CLI:
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=102 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=100 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=142 ms
```

Dzienciol.py console output:

```
student@openflow:~/pox$ POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-04 2] connected
ConnectionUp:  00-00-00-00-00-04
s4_dpid= 4
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
ConnectionUp:  00-00-00-00-00-01
s1_dpid= 1
INFO:openflow.of_01:[00-00-00-00-00-05 6] connected
ConnectionUp:  00-00-00-00-00-05
s5_dpid= 5
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
ConnectionUp:  00-00-00-00-00-03
s3_dpid= 3
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
ConnectionUp:  00-00-00-00-00-02
s2_dpid= 2
NetworkMonitor: route 3 delay: 7 [ms]
NetworkMonitor: route 2 delay: 8 [ms]
NetworkMonitor: route 1 delay: 8 [ms]
NetworkMonitor: route 3 delay: 16 [ms]
NetworkMonitor: route 2 delay: 17 [ms]
NetworkMonitor: route 1 delay: 17 [ms]
-------------------------------------------------------
NetworkPolicer: identified flow: Flow[H1<->H4]
NetworkPolicer: selected route: 1
NetworkPolicer: [  Flow[H1<->H4] route 1  ]
NetworkPolicer: Route Flows Counter: [1, 0, 0]
NetworkMonitor: route 3 delay: 19 [ms]
NetworkMonitor: route 2 delay: 23 [ms]
NetworkMonitor: route 1 delay: 23 [ms]
```

Let’s check the flow distribution state after 5 distinctive pings.

```
-------------------------------------------------------------
NetworkPolicer: identified flow: Flow[H2<->H5]
NetworkPolicer: selected route: 2
NetworkPolicer: [  Flow[H1<->H4] route 1  ]
NetworkPolicer: [  Flow[H1<->H5] route 2  ]
NetworkPolicer: [  Flow[H1<->H6] route 3  ]
NetworkPolicer: [  Flow[H2<->H4] route 1  ]
NetworkPolicer: [  Flow[H2<->H5] route 2  ]
NetworkPolicer: Route Flows Counter: [2, 2, 1]
```

As we can see NetworkPolicer distributes the routes according to our expectations.

3.2. Test the intent handling with flow rearranging

First we've set the topology to have delays:

Route 1 -> 10ms

Route 2 -> 100ms

Route 3 -> 100ms

Then we've started the mininet and pox and created flow from H1 to H4 with ping. NetworkPolicer assigned route 1 for this flow, as this is the default mechanism.

Current state:

```
------------------------------------------------------------
NetworkPolicer: identified flow: Flow[H1<->H4]
NetworkPolicer: selected route: 1
NetworkPolicer: [  Flow[H1<->H4] route 1  ]
NetworkPolicer: Route Flows Counter: [1, 0, 0]
NetworkMonitor: route 1 delay: 16 [ms]
NetworkMonitor: route 3 delay: 134 [ms]
NetworkMonitor: route 2 delay: 135 [ms]
```

Now we are emulating our application which sends intent for flow H2->H6, but with the delay limit that only route 1 can satisfy.

```
student@openflow:~/pox$ python3 application.py 2 6 30
student@openflow:~/pox$
```

Let's see what happened in Dzienciol.py

```
IntentHandler: got  Intent [H2<->H6, 30ms]
IntentPolicer: intent.limit 30
IntentPolicer: Route1.delay 12
IntentPolicer: Route2.delay 134
IntentPolicer: Route3.delay 133
IntentPolicer: route 1 is matching the requirement
NetworkPolicer: [  Flow[H1<->H4] route 1  ]
NetworkPolicer: [* Flow[H2<->H6] route 1 *]
NetworkPolicer: Current State [2, 0, 0]
NetworkPolicer: Do we need to balance? True
NetworkPolicer: Moved flow Flow[H1<->H4]
NetworkPolicer: After balance state [1, 1, 0]
NetworkPolicer: [* Flow[H2<->H6] route 1 *]
NetworkPolicer: [  Flow[H1<->H4] route 2  ]
```

As we can see IntentHandler received intent and forwarded it to the IntentPolicer. IntentPolicer checked the route delays measured by NetworkMonitor and selected route 1 for the intented flow. As we remember we already have a flow on this route. NetworkPolicer checks if the load balancing is

done properly, if not it moves the non-intented flow. Intented flow is marked with "*" in the route flow table.

Also its worth to add that later in logs we can see, that IntentPolicer monitors if the required delay limit is satisfied on the route assigned.

```
NetworkMonitor: route 1 delay: 12 [ms]
NetworkMonitor: route 3 delay: 129 [ms]
NetworkMonitor: route 2 delay: 130 [ms]
IntentPolicer: delay 12ms for intented flow 30ms is satisfied
NetworkMonitor: route 1 delay: 10 [ms]
NetworkMonitor: route 3 delay: 129 [ms]
NetworkMonitor: route 2 delay: 130 [ms]
IntentPolicer: delay 10ms for intented flow 30ms is satisfied
```

3.3. Test the intent monitoring when delay on route changes

First (using techniques from previous scenarios) we've let to the state below.

```
NetworkPolicer: After balance state [1, 1, 0]
NetworkPolicer: [* Flow[H2<->H6] route 1 *]
NetworkPolicer: [  Flow[H1<->H5] route 2  ]
```

As in the previous scenario IntentPolicer monitors the intented flow.

```
NetworkMonitor: route 2 delay: 10 [ms]
NetworkMonitor: route 1 delay: 11 [ms]
NetworkMonitor: route 3 delay: 78 [ms]
IntentPolicer: delay 11ms for intented flow 20ms is satisfied
NetworkMonitor: route 2 delay: 10 [ms]
NetworkMonitor: route 3 delay: 10 [ms]
NetworkMonitor: route 1 delay: 11 [ms]
IntentPolicer: delay 11ms for intented flow 20ms is satisfied
```

But this time, we changed the delay on route 1 to a really high one.

```
mininet> exit
++++++++++++ 50ms delay started on s1-eth4 interface ++++++++++++
```

Let's see what happens.

```
NetworkMonitor: route 2 delay: 10 [ms]
NetworkMonitor: route 3 delay: 11 [ms]
NetworkMonitor: route 1 delay: 52 [ms]
IntentPolicer: delay 52ms for intented flow 20ms is too high
IntentPolicer: Moving flow...
NetworkPolicer: [  Flow[H1<->H5] route 2  ]
NetworkPolicer: [* Flow[H2<->H6] route 2 *]
NetworkPolicer: Current State [0, 2, 0]
NetworkPolicer: Do we need to balance? True
NetworkPolicer: Moved flow Flow[H1<->H5]
NetworkPolicer: After balance state [1, 1, 0]
NetworkPolicer: [* Flow[H2<->H6] route 2 *]
NetworkPolicer: [  Flow[H1<->H5] route 1  ]
```

As we can see IntentPolicer detected that the delay was too high, and it moved the flow to the route with satisfying delay. Additionally, the NetworkPolicer had to rearrange the flows after this enforcement.

4. Summary

To sum up we've implemented the OpenFlow controller with "Dzienciol.py" as a product name. Our Dzienciol is a component of pox controller and works on Mininet emulated network. The external component of our solution is application.py (python script emulating the "Application"). All of the code, along with "Run" instruction and documentation is available github repository linked below:

https://github.com/0x41gawor/eines-proj

5. Conclusions, observations, and complaints

During the project implementation we've observed a few things:

- mouse scrolling is not working well on the `openflow-lab-project` Virtual Machine,
- We do not like python programming – its Dynamic Typing was disturbing for us.

The whole thing took about 6 man-days. We've learnt a lot about OpenFlow, Pox controller, Mininet commands and extended our python/threading/socket/Linux programming skills.

The usage of sockets wasn't our first try to externally send the intent to the controller. First, we've tried Linux Signals. We abandoned the idea due to no code-access to the pox thread that is accessible from the console. The idea of hardcoding the intents or reading it from beforehand prepared file was unnatural for us. We wanted to send the real time intents, as it is in the real world.