University of Pisa
MSc in Computer Engineering
Advanced Network Architectures and Wireless Systems

SDN project

Daniela Comola
Sara Lotano
Eugenia Petrangeli

Academic Year 2019/2020

Contents

1 Introduction

In this project we have two networks, Network A is composed of three clients and a SDN-enabled switch (SS), which is managed by a SDN controller. The Network B is composed of two servers and one legacy switch (LS). The two networks are connected by means of two routers.

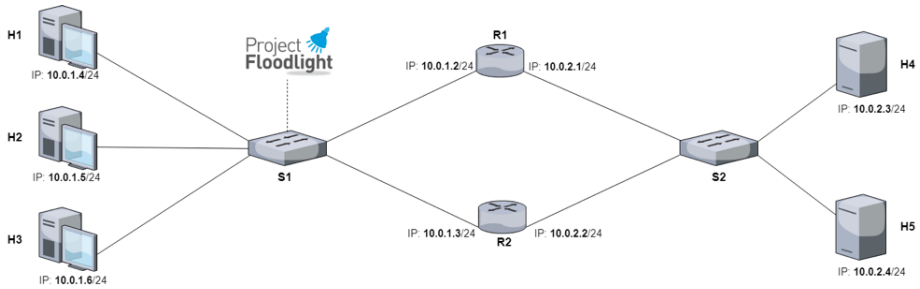

Figure 1: Network Architecture

The aim of the project is to implement a centralized mechanism to let clients in Network A communicate with servers in Network B through one gateway (R1 or R2) that can be changed dynamically. At any time only one router must operate as the gateway for all clients, while the other one is available as a backup. Therefore, no load balancing between routers must be implemented. Moreover, clients must be unaware of which router is acting as a gateway at all times.

2 Design

The network is emulated by using the Mininet emulator and in the table below are shown the addresses chosen for the two networks.

Network	Device	Interface	IP address	Netmask	Default Route
Core	**R1**	Ethernet 0	10.0.1.2	/24	
		Ethernet 1	10.0.2.1	/24	
	R2	Ethernet 0	10.0.1.3	/24	
		Ethernet 1	10.0.2.2	/24	
A	**H1**	Ethernet 0	10.0.1.4	/24	10.0.1.1
	H2	Ethernet 0	10.0.1.5	/24	10.0.1.1
	H3	Ethernet 0	10.0.1.6	/24	10.0.1.1
B	**H4**	Ethernet 0	10.0.2.3	/24	10.0.2.2
	H5	Ethernet 0	10.0.2.4	/24	10.0.2.2

The switch S1 is an SDN switch linked to the controller through the loopback address 127.0.0.1 and using the port 6653 in order to interact with it by using the OpenFlow protocol.
R1 and R2 implement the mechanism of virtual router. This one has the virtual address 10.0.1.1 and virtual MAC address 00-00-5E-00-01-01. As shown in the table all the clients in the Network A use the address 10.0.1.1 as default route.

2.1 Protocol Description

The routers R1 and R2 send in broadcast every **ADVERTISEMENT_INTERVAL** seconds an advertisement message having this format: ***router_name*:*priority*** using the application port **1234**. The controller uses these messages to learn about the routers participating in the virtual router implementation and it will choose as Master the one with the highest priority. Priority's values are in the range of [0,255]. In particular, we used the following ones:

- priority = 255 for R1
- priority = 100 for R2

If the controller does not detect messages from the master for more than **numAdv * advertisementInterval** then it will choose another master from the registered routers, looking for the one with the highest priority, that has sent the last message in less than **numAdv * advertisementInterval**.

Below are shown the default values for the parameters used in the protocol:

- ADVERTISEMENT_INTERVAL = 1 (second)
- advertisementInterval = 1000 (milliseconds)
- numAdv = 3
- masterDownInterval = numAdv * advertisementInterval

The **ADVERTISEMENT_INTERVAL** parameter is used to configure the sending period in the python script implementing the routers behaviour. The **advertisementInterval**, **numAdv** and **masterDownInterval** parameters, instead, are used by the Controller.

3 Configuration

Add in the folder *floodlight* the following files:

- Add the file *net.floodlightcontroller.core.module.IFloodlightModule* in the folder *floodlight/src/main/resources/META-INF/services*
- Add the file *floodlightdefault.properties* in the folder *floodlight/src/-main/resources*
- Add the package *net.floodlightcontroller.task2* in the folder *floodlight/src/main/java*

In the folder *net.floodlightcontroller.task2* there are the following files:

- *Utils.java*, contains all the parameters and structures necessary for the controller
- *Router.java*, contains all the information about the router
- *ARPController.java*, manages the ARP messages coming from the hosts in the network A and from the Routers
- *VirtualAddressController.java*, manages the advertisement messages coming from the routers, the election of a new Master and the IPv4 messages coming from the hosts in network A and from the routers
- *RestController*, exposes the Restful interface and contains the class *RestControllerWebRoutable* that represents the RESTFul
- *IRestController*, interface that exports the methods *getNetworkInfo()* and *setNumAdv(int newValue)*
- *GetNetworkInfo*, define a custom resource and return the information about the network
- *ChangeNumAdv*, define a custom resource and set the *numAdv* variable

4 Running the Network

In order to emulate the whole network is sufficient to start floodlight and digit the command in the terminal:

```
# python network_topology.py
```

Once Mininet is running is possible to open the routers or hosts terminal (in the example the host is H1) with the following command:

```
> xterm H1
```

Through its terminal is possible to ping any host on the network, for example if we want to ping H4 is sufficient to digit:

```
# ping 10.0.2.3
```

If we want to force the controller to find a new master, digit the following command in the Mininet terminal:

```
> link S1 R1 down
```

In order to restore R1 as the master digit the following commands from the Mininet terminal (the second one in the R1 terminal)

```
> link S1 R1 up
# python3 router.py R1 255
```

4.1 REST

In order to retrieve the resources exposed by the controller you can use the add-ones "RESTED" for Firefox.
It's possible to perform the *GET* request on the resource *"http://127.0.0.1:8080/vr/network/info/json"* as shown in the figure below.

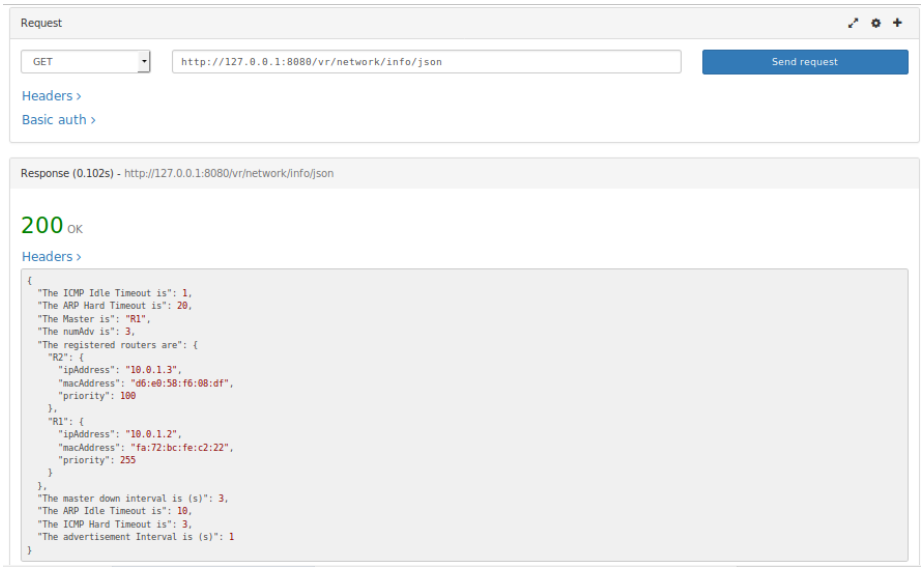

Figure 2: REST GET

It gives you the following information:

- Registered routers and their information
  - Router's name
  - IP Address
  - MAC Address
  - Priority
- The elected Master
- The *numAdv* variable
- The advertisement interval
- The Master down interval
- The ICMP idle timeout
- The ICMP hard timeout
- The ARP idle timeout
- The ARP hard timeout

This functionality is useful to monitor changes in the network. For example, the figure above shows that router R1 has been elected as Master, but, if the link between it and the switch goes down, a new election process will start.
It's possible to retrieve which is the new elected Master performing the *GET* request again.
The figure below shows this new information.

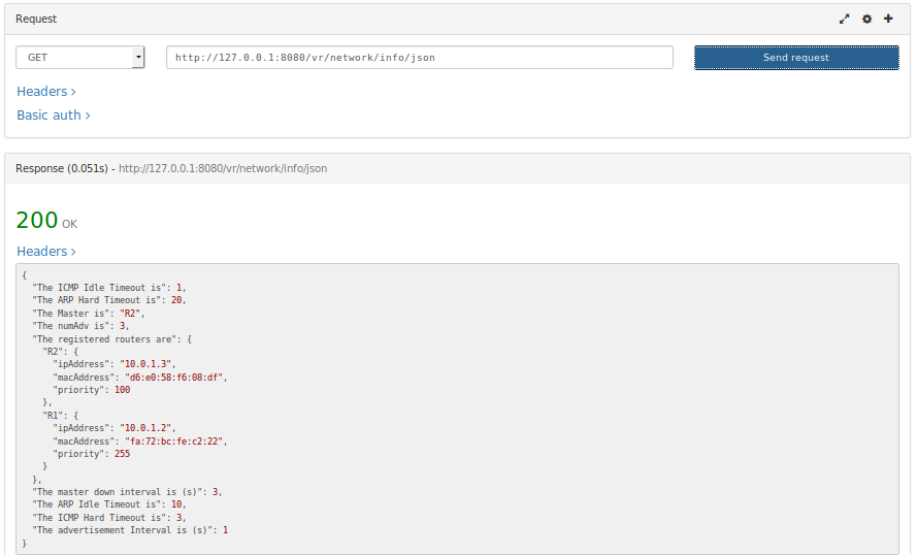

Figure 3: REST GET

It's also possible to change the value of the *numAdv* variable performing the *POST* request on the resource *"http://127.0.0.1:8080/vr/network/numadv/json"* as shown in the figure below.

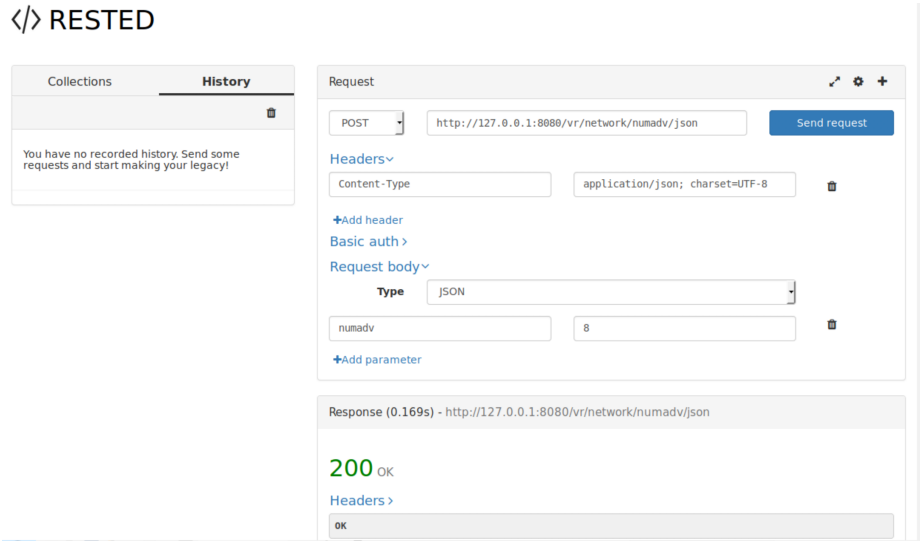

Figure 4: REST POST

Through a new *GET* request the changed value of the *numAdv* variable can be verified.
Each request can be performed also directly in the browser using the appropriate URI, as shown in the figure below.

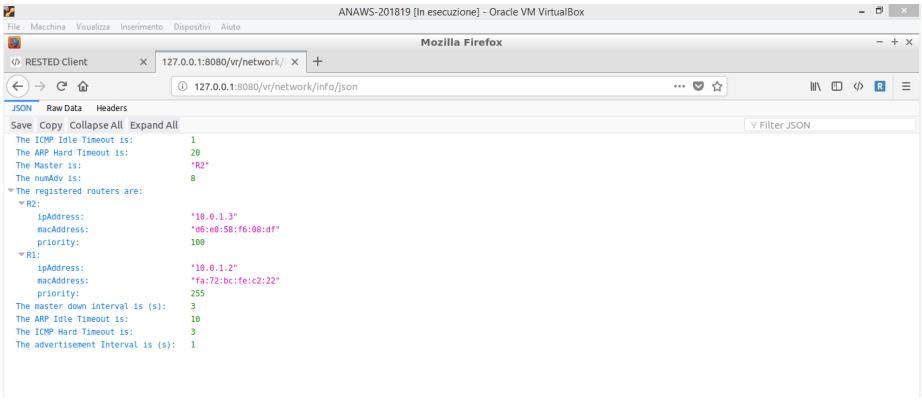

Figure 5: REST GET