Departamento de Engenharia Electrotécnica

Configuração e Gestão de Redes

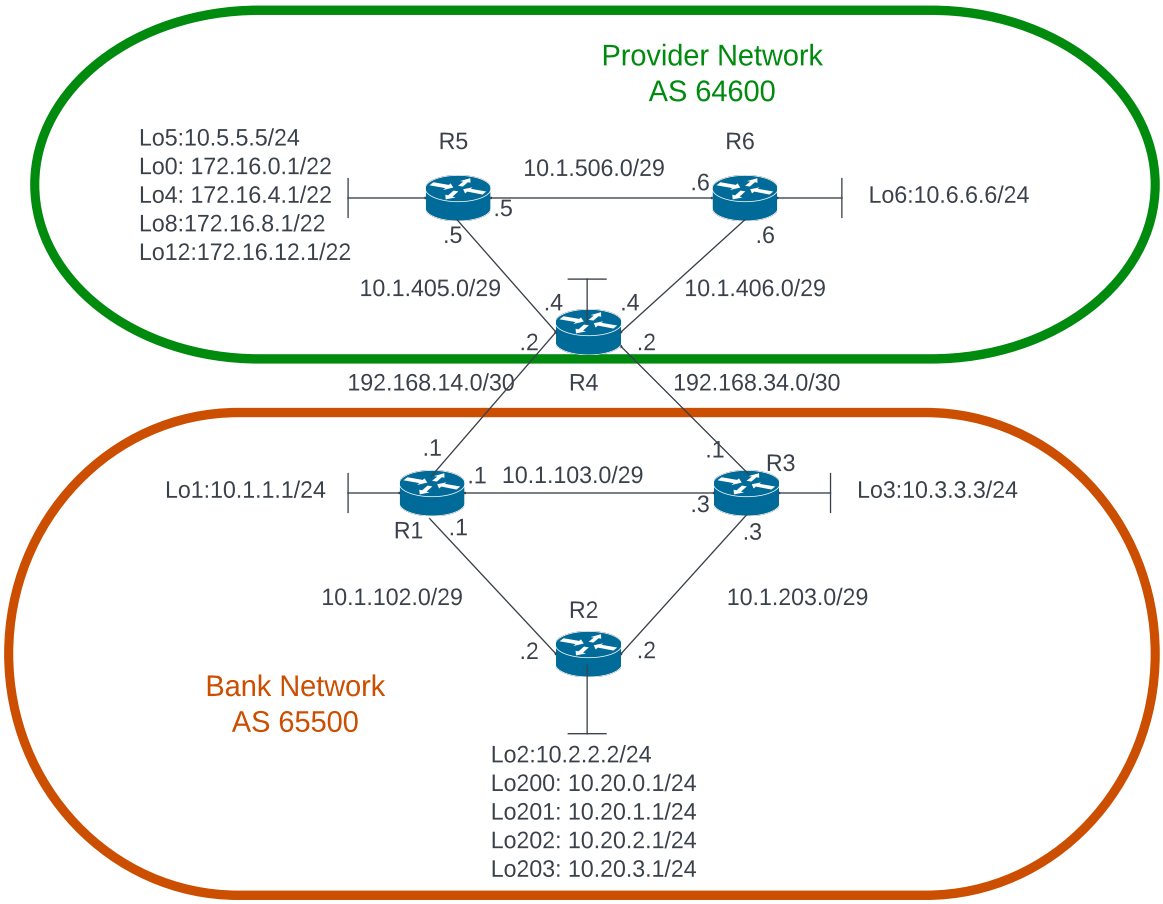
2024 / 2025

2º Lab:
Part 1 – BGP Scenario (30%)
Part 2 - Software Defined Networks (70%)

pfa@fct.unl.pt **Pedro Amaral**

PART 1 – BGP SCENARIO

The second BGP routing scenario is represented in the following figure:



Work Plan and links for examples

You should implement and verify the following requirements.

1. Use the addressing scheme shown in the diagram
1. Configure OSPF inside the bank Network with a single area (area 0).
2. Configure OSPF inside the provider network with a single area (area 0).
3. Configure the Bank network to be in BGP AS 65500 and the provider network in BGP AS 64600.
4. **Do not include** the 192.168.14.0/30 and the 192.168.34.0/30 networks in the OSPF instances inside the two ASes.
5. All routers will be participating in BGP. Configure all routers for a full mesh of IBGP peers in each system.
6. In the Bank Network the networks of Lo200; Lo201; Lo202 and Lo203 interfaces in R2 should be advertised via BGP as a summary, these are the only networks advertised by the bank AS via BGP.
7. R5 should send a summary route via BGP to the bank network representing the Lo0; Lo4; Lo8; Lo12 loopback interfaces, these are the only interfaces to be announced via BGP.
8. R4 should prefer the path to the Bank network via the Ethernet link R4-R3.

9. Routers in the Bank network should prefer the link R1-R4 to reach provider networks.
10. Test connectivity between the BGP announced networks of R5 and R2.

PART 2 – SOFTWARE DEFINED NETWORKS

1. OVERVIEW

In this lab you will develop an SDN app using the floodlight controller (documentation can be found in the link: https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+Documentation). The app will control a topology of OpenFlow enabled switches. The app will configure the switches in a network to work as learning switches. Communication between hosts is limited according to an access control policy that can limit the number of destinations that a host can communicate with simultaneously (after reaching the connection limit, to send packets to another destination a certain time must have elapsed without communication with one of the previous destinations). Finally, the duration and byte count of every TCP connection should be stored in a data file. To simulate the Openflow enabled switches and build a test network you will use a simulator called mininet http://mininet.org/walkthrough/ that simulates OpenFlow enabled switches in a computer using Virtual Switches in the linux kernel (Open vSwitch) http://openvswitch.org/.

2. SETUP INSTRUCTIONS

The goal is to make a first contact with the use of the OpenFlow protocol to control the switches of a network. You will use the mininet simulator http://mininet.org/ to simulate the OpenFlow controlled network. You will then connect an external controller to the topology. The controller uses OpenFlow to receive information from the switches (unmatched packets) and instructs the switches how to deal with them.

For mininet you should download a mininet VM directly in the mininet website.

In the http://tele1.dee.fct.unl.pt/cgr_2023_2024/pages/laboratorios.html you can find a link to download a VM that already has all the needed setup including the floodlight project in eclipse with the class you need to start coding. YOU ARE ADVISED TO USE IT for a faster setup. The following instructions up until the section about RUNNING FLOODLIGHT, are **ONLY** if you want to setup the environment in your own linux distribution (which is NOT ADVISED).

For floodlight you need a Linux installation (Ubuntu or Xubuntu preferably) that has JAVA 8 (other Java Versions work with some tweaking afterwards), Maven, the Python Development package and the latest version of Eclipse for java installed. If you start with a fresh ubuntu or Xubuntu installation, you can install Java 8 in with the following commands:

```
sudo apt install openjdk-8-jre-headless
sudo apt install openjfx
```

The second command is needed to install the JavaFX package that does not come with openjdk8.

Then install Maven and Python-dev with the following commands:

```
sudo apt-get install build-essential ant maven python-dev
```

You can find the eclipse installer files in the project website in http://tele1.dee.fct.unl.pt/cgr_2021_2022/pages/laboratorios.html. Just unarchive them and then run the eclipse-inst file. (You should choose the Eclipse IDE for Java Developers option).

After you have Java 8, Maven Python-dev and Eclipse you can download the Floodlight folder that is available at the course website in http://tele1.dee.fct.unl.pt/laboratorios.html. The Folder contains an Eclipse project already pre-compiled as well as the skeleton of a new module that you should use to build your work. Just import the project to eclipse following these instructions:

- Open eclipse and create a new workspace
- File -> Import -> General -> Existing Projects into Workspace. Then click "Next".
- From "Select root directory" click "Browse". Select the parent directory where you placed floodlight earlier.
- Check the box for "Floodlight". No other Projects should be present and none should be selected.
- Click Finish.

At this stage the project might have some errors due to JAVAFx not being included in the built path of the project in eclipse. To solve these you should:

1) Right Click on the Floodlight project and select Build Path -> Configure Build Path
2) Go to the Libraries tab and click Add External JARs
3) Navigate to the /usr/share/openjfx/lib and select all JARs and Zip files.

After importing the floodlight project in eclipse you find under the src/main/java folder of the floodlight project the packages net.floodlightcontroller.cgrmodule and net.floodlightcontroller.cgrmodule.util. The cgrmodule contains the class file CGRmodule.java where you should build your code. The util package contains a class called SwitchCommands.java that contains some useful methods to build the most common OpenFlow messages.

Floodlight can be executed in two different ways: from eclipse or from the terminal.

Running Floodlight in Eclipse

Since Floodlight uses a dynamic module loading system in order to run the project directly through Eclipse, you must configure it to launch it in the correct manner.

Create the FloodlightLaunch target:

- Click Run->Run Configurations
- Right Click Java Application->New
- For Name use FloodlightLaunch
- For Project use Floodlight
- For Main use net.floodlightcontroller.core.Main
- Click **Apply**

To then run Floodlight click on the drop-down arrow next to the Play button and select the proper target to run.

Running Floodlight in Terminal

Assuming java is in your path, you can also directly run the floodlight.jar file produced by compiling the project using ant from within the floodlight directory:

```
$ java -jar target/floodlight.jar
```

Floodlight will start running and print log and debug output to your console.

Extra information (if you have problems importing and running the code provided at the website)

IF you have any problem opening the Floodlight Project provided in the web site you can try the following instructions to build a new Floodligt project from scratch:

To manually install Floodlight and compile it in to an eclipse project following these simple instructions:

```
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ git submodule init
$ git submodule update
$ ant

$ sudo mkdir /var/lib/floodlight
$ sudo chmod 777 /var/lib/floodlight
```

If your Linux installation does not have git installed you can installed with the following command:

```
sudo apt install git
```

If you need you can find these instructions also at:
https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343544/Installation+Guide
(Start in the Download and build section and Stop at the “Simulating a Network” section).

After clamming the Floodloght code in to your system you need to set it up to use the Eclipse. You can do this by using the eclipse ant target

```
$ ant eclipse
```

This creates several files: Floodlight.launch, Floodlight_junit.launch, .classpath, and .project. From these you can setup a new Eclipse project.

2. PROJECT DESCRIPTION

Floodlight is an open source SDN controller written in Java. Floodlight's module-based loading system offers a flexible and simple management of various service components and applications.
In the VM the run button in the eclipse toolbar is already configured to run floodlight loading all modules needed for your L2Switch application to work.

The L2 learning switch to implement is a simple switch, a bit smarter than a repeater (hub). Unlike a hub which floods every incoming packet to every port (except the incoming port), our learning switch sends the packet to only one port if the switch already knows the port which the destination node is connected to. This

is very similar to the behaviour of a traditional L2 Switch, our switch application will also implement an access control policy and a TCP traffic data collection mechanism:

a) Connection Limitation: For some hosts, the controller only allows a limited number of simultaneous destinations. These host are in a list that contains the hosts IP addresses and the limit of connections that they have, hosts that are not on the list do not have a limit of connections. When a host is actively communicating to the maximum number of destinations, the controller should only allow it to talk to a new destination if one of the previous connections has terminated (HINT: you should use the idle timeout of the Flow Modification object properly).
b) TCP traffic data collection - For every TCP connection, the byte count and duration of the flow should be stored in a text file. (HINT: You can install a flow rule using a Flow Mod message to match packets of TCP flows using an Idle Timeout and collect the statistics of the removed Flow when it expires).

3. SPECIFICATIONS

The learning switch application maintain a MAC table were packets' incoming port and source MAC address mappings are stored. Upon the arrival of a packet, if the destination MAC address is found in the table, the packet is sent directly to the port. If the MAC address was not learned yet the mapping in port/source MAC address should be stored in the MAC table and the packet is flooded.

When the destination port for a MAC is already known the controller can handle the packet in two ways:

1. Tell the switch to send the given packet to the destination port using an Openflow PacketOut message (OFPacketOut object in Floodlight).
2. Ask the switch to forward all future packets for a particular MAC address to the corresponding port by installing an OpenFlow Flow entry in the switch using a FlowModification message (a OFFlowMod object in Floodlight).

The latter is preferred because it reduces the amount of communication between the switches and the controller with the switch handling the following packets using the installed rule.

The controller must check if a Host source address has a maximum limit of simultaneous communications. This is done by checking a connection limitation list that contains the hosts source addresses that have this limitation and the corresponding MAX_DESTINATION_NUMBER for each of the hosts. For limited host the controller only the sending of packets to a limited number of destinations simultaneously. When such a host reaches the MAX_DESTINATION_NUMBER of destinations, the controller should block traffic for other destinations. When the host stops sending traffic to one of the destinations then it can send traffic to a new destination. For example if MAX_DESTINATION_NUMBER = 3 for Host 10.0.0.2, the desired behaviour is the following:

When a packet arrives from 10.0.0.2 the list is checked to see if this host source address is one if the limited hosts and what is its MAX_DESTINATION_NUMBER. The controller sees that it is and that MAX_DESTINATION_NUMBER = 3. Host 10.0.0.2 can then send traffic (say for example a ping) if it already communicating with less than 3 destinations. While traffic is being sent to those 3 destinations the host is not able to ping a 4th different destination. When one of the 3 original pings stops sending packets then the host can communicate with another destination.

The statistics of every TCP flow in the network, identified by the tuple: (SourceIP; DestinationIP; SourcePort; DestinationPort) containing the Data: (Bytecount; Duration) should be stored in a CSV (Comma Separated Values) text file.

This means that the controller must detect when a TCP communication begins and deploy at that time a rule to gather statistics at one of the switches that the TCP connection traverses.

All features should work regardless of the network topology. In other words, you should test the application in multiple switch networks (without loops in a loop you might run in to trouble but by now you already know why). To simplify we assume that the topology never changes. Remember that in a multi switch topology the same packet traverses multiple switches that are all controlled by the same application.

3. INSTRUCTIONS AND HINTS

Download the VM provided in the course website that contains the floodlight controller. Install VM workstation player and import the VM and power it on. You should then also download a mininet VM directly in the mininet website.

Start by familiarizing yourself with Mininet

3.1 Using Mininet

Mininet emulates an OpenFlow network and end-hosts within a single machine. It includes built-in support to create several common topologies, plus it allows for construction of custom topologies using a python script.

Take some time to follow the Mininet walkthrough available at http://mininet.org/walkthrough/ . You can also use it as a reference. Concentrate on understating the options used in the following mininet startup command.

```
sudo -E mn --topo tree,2 --mac -x --controller=remote,ip= 172.18.0.3,port=6653 -
-switch ovsk,protocols=OpenFlow13
```

A small description of each of the options is also given here:

- `sudo -E` runs as root but maintains use environment variables (needed for X11 forwarding to work).
- `mn` runs mininet.
- `--topo tree,2` creates a tree topology of depth 2 with the default fanout of 2 (i.e., binary).
- `--mac` makes the mac address of mininet hosts the same as their node number.
- `--x` automatically opens a xterm for each host and switch in the topology.
- `--switch ovsk,protocols=OpenFlow13` uses Open vSwitch in kernel mode for each of the switches using open flow 1.3, this is important since otherways OpenVswitch will use OpenFlow15 which is different in some aspects and most of the floodlight documentation was written for OpenFlow13.
- `--controller remote,ip=<ip>, port=<port>` specifies the IP and port of the controller

Once Mininet is running, you can obtain information about the network, generate traffic (for example using IPerf, Ping etc..), and run other commands on individual hosts. You can also use the XTerms for each network element and run the necessary commands on them.

3.2 Developing the Application

After following the initial setup instructions a new module to implement the project called CGRmodule can be found in the eclipse project under the package net.floodlightcontroller.CGRmodule. Its implementation is on the file CGRmodule.java that contains the code for the java class that implements the module.

The file as it is provided instructs the switches to flood the packets they receive and therefore works like a hub. You should change the class code to provide the specified functionality.

You can run floodlight in two different ways:

The first one is using the Eclipse IDE itself by choosing the main.java file under the net.floodlight.core package and choosing run as java application or directly in the run button of the IDE. The log messages will be visible in the console tab in Eclipse.

The second way is to build floodlight using ant. For this you should issue the following commands in a terminal windows inside the floodlight directory:

```
pfa@dev-virtual-machine:~/floodlight$ ant clean
```
(to clean up previous builds)

```
pfa@dev-virtual-machine:~/floodlight$ ant
```
(to build)

You can then run the controller with java using the .jar file with the following command:

```
pfa@dev-virtual-machine:~/floodlight$ java -jar target/floodlight.jar
```

You should search for the following two lines of output that signal that our module project 2 was loaded:

```
12:06:51.279 INFO [n.f.p.CGRmodule] CGR L2 Switch module initialized
```

and that the controller is listening for connections from switches:

```
12:06:51.644 INFO [n.f.c.i.OFSwitchManager:main] Listening for OpenFlow switches on
[0.0.0.0]:6633
```

This second is particularly important since it indicates that the controller is listening on all interfaces of the machine (hence the 0.0.0.0/0.0.0.0 IP and mask) on the port 6653. This means that when you start mininet you should define in the remote controller IP address the IP address of the VM running floodlight and the port should be the one you see in the above line.

3.3 Testing the Application

Start by checking the flooding behaviour that the CGRmodule module produces in the switches.

Start a mininet simulation as described above and start the floodlight controller with the CGRmodule module included.

You will see on the floodlight output the indication of the negotiation between the various modules of floodlight and the switches with at some point the following messages indicating that the switches have connected with the controller.

```
15:32:09.993 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:03
connected.
15:32:10.016 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:02
connected.
15:32:10.017 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01
connected.
```

You can verify that hosts can ping each other, and that all hosts see the exact same traffic (the behaviour of a hub). To do this you can use the Xterms of the hosts.

Arrange each Xterm so that they're all on the screen at once. This may require reducing the height of them to fit a cramped laptop screen.
In the Xterms for each host, run tcpdump, a utility to print the packets seen by a host:
For example for hosts h2 and h3:

```
# tcpdump -XX -n -i h2-eth0
```

and respectively:

```
# tcpdump -XX -n -i h3-eth0
```

In the xterm for h1 for example, send a ping to host 2:

```
# ping -c1 10.0.0.2
```

The ping packets are now going up to the controller (floodlight installs a flow entry with 0 priority, a match any and an action forward controller causing all unmatched packets in other flows entries to be sent to the controller in a PacketIn message) the controller resends them via a PacketOut message to the switches with the action set to flood the packet out all interfaces except the sending one. You should see identical ARP and ICMP packets corresponding to the ping in all xterms running tcpdump. This is how a hub works; it sends all packets to every port on the network. So the ping packet is seen in all hosts.

If you start mininet with the -x option an Xterm also opens for one of the switches. The reason for only one terminal opening is that mininet uses linux name spaces to create a different network name spaces. By default only hosts are put in a different namespace. Switches are all in the linux space so they all share the same environment so you can issue commands in any of them in the same window. If you wish to see the OpenFlow flow entries installed in a switch you can use the following OpenVswitch command in the Xterm of the switch opened by mininet or in a terminal of the host system (since switches run in the host namespace):

```
ovs-ofctl dump-flows s1
```

This for switch 1 you can do the same for switch 2 replacing `s1` by `s2` and so on.

You can also use the controller Graphical User Interface (GUI) by using a browser in the controller linux machine using the address http://localhost:8080/ui/index.html

NOTE: Since the network namespace of the switches in mininet is the same as the host system where it is installed if you use the mininet in the Floodlight VM you will receive multicast packets that are sent by any applications you have installed in the switches of mininet. Switches in mininet appear as network interfaces of the host system therefore if an application sends a multicast packet that packet will be received in all interfaces. If the mininet switches have a rule to send unmatched packets to the controller this means that you will receive these packets in the controller. If you use the mininet VM these packets are less frequent since the OS is tailored for mininet only. Either way this is not a

problem for the implementation you should only be aware that such packets might appear if you use mininet in the floodlight VM:

In mininet you can use other commands like iperf:

```
mininet> iperf
```

This runs an iperf TCP server on one virtual host, then runs an iperf client on a second virtual host. Once connected, they blast packets between each other and report the results.

3.6 Implementing the application

You should change the CGRmodule class in file CGRmodule.java to perform according to the specifications. Some of the needed code for the implementation of the learning switch feature is already commented in the file. You can also use the methods in class SwitchCommands.java in the package net.floodlightcontroller.cgrmodel.util already imported in class CGRmodule.

To understand the general structure of the class read the coding parts of the floodlight tutorial on creating a module https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+Write+a+Module

Besides the methods the IFloodlightModule and IOFMessageListener interfaces, the class has methods already defined to handle PacketIn Messages, write PacketOut Messages from Packet In messages as well as a series of suggested methods to handle the data structure to store the MAC address / Port mappings for the learning switch feature.

The tutorials on how to process a Packet_In message https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+Process+a+Packet-In+Message and on how to create a Packet_out message https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+Create+a+Packet+Out+Message are also a great source of examples on how to use the several needed OpenFlow structures as implemented by Floodlight.

Floodlight uses the OpenFlowJ-Loxi Java generated API to implement Openflow concepts in Java objects. For reference on how to create Matches, Flow entries etc, you can check the OpenFlowJ-Loxi tutorial in floodlight https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+use+OpenFlowJ-Loxigen

You can also use the base java documentation for OpenFlowJ and floodlight that can be found in floodlight.github.io/floodlight/javadoc/openflowj-loxi/index.html (OpenFlowJ) floodlight.github.io/floodlight/javadoc/floodlight/index.html (Floodlight base classes), these links are also added in the firefox browser bookmarks tab in the VM provided at the project website.

HINT

You can take a look in the original floodlight modules for code examples.