

Communication Networks

Project 1: Build your own Internet

Deadline: May 3 2018 at 11.59pm

In this document, we first introduce in §1 a set of commands you may need to configure an Open vSwitch. We then show in §2 how to configure a Quagga router.

1 Configuring Open vSwitch

Open vSwitch¹ [1] is one of the most popular software switches. It can typically be used in virtual environments, for instance to connect two virtual machines. When an Open vSwitch is running, a set of commands are available to check its state and configure it. To print a brief overview of the switch state and its parameters, you can use the following command:

```
> ovs-vsctl show
```

This command also tells you the VLANs each port belongs to. One port has the name of the switch and has the type *internal*. This is a local port used by the host to communicate with the switch. You do **not** need to use this port. To get more precise information about the status of the ports, you can use the following command:

```
> ovs-ofctl show NAME
```

where NAME is the name of the switch (*e.g.*, ETH-IRCH). To get the current configuration and all the statistics of the switch, you can get a dump of the switch database with the following command:

```
> ovsdb-client
```

For example one entry of the database could look like this:

```
> 645981b6c 0 false [] 0 false [119f5-2be8bf5] [] [] eth-oer1 {stp-path-cost="100"}
[] {stp_error_count=0, stp_rx_count=312, stp_tx_count=3} {stp_port_id="8004", stp_role=alternate,
stp_sec_in_state="223", stp.state=blocking} [] [10, 20] []
```

The information is a little bit cryptic, but we have highlighted in red the important information that you may use for the assignment. For instance, this entry is for the port named `eth-oer1`, the spanning tree path cost for this link is 100, and the spanning tree protocol has turned this port in block mode. You can also find the cost of the path to the root bridge by looking at the field `stp_root_path_cost` (not shown here). This port is also a trunk port in VLANs 10 and 20. You can also display information about the MAC address table of the switch with the following command:

```
> ovs-appctl fdb/show NAME
```

with NAME the name of the switch. To change the spanning tree priority of a switch (used by the spanning tree protocol to elect the root switch), you can use the following command:

¹<https://www.openvswitch.org>

```
> ovs-vsctl set bridge NAME other_config:stp-priority=10
```

with NAME the name of the switch. The switch with the lowest priority will be elected as root. To change the spanning tree cost of a path, you can use the command:

```
> ovs-vsctl set port PORT_NAME other_config:stp-path-cost=1
```

This will set the spanning tree path cost of the port PORT_NAME to 1. A path with a low cost is preferred to a path with a higher cost. The default cost of a path is 2. When one port of a link has to be set in forward mode and the other one in block mode, the decision about which port will be in block mode is first based on the cost of the path to the root bridge, then on the bridge ID. To revert the stp-path-cost back to its default value, you can use the following command:

```
> ovs-vsctl clear port PORT_NAME other_config
```

To add a port to a VLAN, you can use the following command:

```
> ovs-vsctl set port PORT_NAME tag=10
```

This will add the port PORT_NAME to the VLAN 10. Since PORT_NAME is in one VLAN only, the port is in tag mode. Should you add a port to several VLANs, you can use the following command:

```
> ovs-vsctl set port PORT_NAME trunks=10,20
```

the port PORT_NAME will be in trunk mode for VLANs 10 and 20. To clear the VLAN configuration on a port (here for a trunk link), you can use the following command:

```
> ovs-vsctl clear port PORT_NAME trunks
```

2 Configuring IP Routers

Quagga routers are software routers running on top of Linux. In other words, running Quagga on Linux allows you to transform your laptop or server into an IP router, where the Linux interfaces act as interfaces for the Quagga router. You can configure a Quagga router through a Common Line Interface (CLI). Each IP router vendor (e.g. Cisco or Juniper) or software routing suite (e.g. Quagga) has its own CLI. Fortunately, those CLIs are similar, and if you know how to configure a router using the Quagga CLI, you can easily configure a Cisco or a Juniper router as well.

2.1 The Quagga Command Line Interface

We now briefly describe how to configure a Quagga router². When you enter the Quagga CLI, you see the following line:

```
router#
```

At any time in the CLI, you can type ? to see all the possible commands you can currently type:

```
router# ?
```

clear	Reset functions
configure	Configuration from vty interface
exit	Exit current mode and down to previous mode

²This is a very short Quagga introduction, we strongly recommend you to take a look at the official documentation [2] to get more information.

<code>no</code>	Negate a command or set its defaults
<code>ping</code>	Send echo messages
<code>quit</code>	Exit current mode and down to previous mode
<code>show</code>	Show running system information
<code>traceroute</code>	Trace route to destination
<code>write</code>	Write running configuration to memory, network, or terminal

For example, the command `show` will print various snapshots of the router state. To see what kind of information can be shown, just type `show ?`. For example, `show running-config` will print the running configuration. You can shorten the commands when there is no possible ambiguity. For instance `show run` is equivalent to `show running-config`. Similarly to the Linux terminal, you can also use auto-completion by pressing the tabulation key. If you want to test your connectivity, you can use `ping` or `traceroute`.

Switching to configuration mode To configure your router, you must enter the configuration mode with `configure terminal` (`conf t` for the short version). You can verify that you are in the configuration mode by looking for the “config” prefix in your CLI prompt. Use `exit` to leave the configuration mode and to go back to the previous mode.

Remove a command If you want to cancel parts of the configuration, you can prefix the command you want to remove with `no`.

2.2 Configure the router interfaces

A router interconnects IP networks through several IP interfaces. When receiving a packet from one interface, it forwards it to another based on pre-computed forwarding decisions. Each IP interface must have an IP address configured and must be in a different subnet³. To configure an interface, you must first enter the configuration mode, and then specify the name of the interface you want to configure:

```
router# conf t
router(config)# interface interface_name
router(config-if)# ip address 1.0.0.1/24
```

You can verify that the running configuration has been updated correctly with the command `show run`. With `show run`, you can also see the name of the interfaces of the router. You can also show information about the interfaces using `show interface` and more detailed information about a specific interface, using `sh interface interface_name`. **Important:** Do **not** configure two different IP addresses on one interface at the same time. If you have configured a wrong IP address, first remove the address with the `no` command and then configure a new IP address:

```
router# conf t
router(config)# interface interface_name
router(config-if)# no ip address 1.0.0.1/24
router(config-if)# ip address 2.0.0.1/24
```

Once you have configured an IP address and a subnet on an interface, the router knows that packets with a destination IP in this subnet must be forwarded to this interface. To show the subnets that are directly connected to your router, you can use the following command.

```
router# show ip route connected
C>* 2.0.0.0/24 is directly connected, interface_name
```

³Try to think what would happen if that wasn't the case

We see that 2.0.0.0/24 is directly connected and reachable with the interface `interface_name`. At this stage, a packet with a destination IP that is not in a directly connected subnet will be dropped. If you want your router to know where to forward packets with an IP destination in a remote subnet, you must use routing protocols, such as OSPF or BGP.

2.3 Configure static routes

OSPF and BGP are dynamic routing protocols which will automatically route around network failures. In some cases though, it is useful to define routes statically. Static routes are set once and for all, independently of the network conditions. Static routes are simply configured by specifying the IP prefix along with the IP next-hop. As an illustration:

```
router# conf t
router(config)# ip route 3.0.0.0/24 2.0.0.2
```

would force the router to direct all traffic destined to 3.0.0.0/24 to 2.0.0.2. You can verify that a static route has been installed correctly using `show ip route static`:

```
router# show ip route static
S 3.0.0.0/24 [1/0] via 2.0.0.2 inactive
```

2.4 Configure OSPF

OSPF routers flood IP routes over OSPF adjacencies. Quagga routers continuously (and automatically) probe any OSPF-enabled interface to discover new neighbors to establish adjacencies with. By default, Quagga router will activate OSPF on any interface whose prefix is covered by a `network` command under the `router ospf` part of the configuration. For instance, the following configuration would activate OSPF on any interface whose IP address falls under 1.0.0.0/24 or 2.0.0.0/24:

```
router# conf t
router(config)# router ospf
router(config-router)# network 1.0.0.0/24 area 0
router(config-router)# network 2.0.0.0/24 area 0
```

OSPF has scalability issues when there is a large number of routers. To make it more scalable, the router topology can be hierarchically divided in what is called “areas”. In this assignment, your network is small and you do not need more than one area: you will only use the area 0.

To check the OSPF neighbors of a router, you can use the following command:

```
router# show ip ospf neighbor

Neighbor ID Pri State Dead Time Address Interface RXmtL RqstL DBsmL
1.0.0.2 1 Full/Backup 1.0.0.2 pari:1.0.0.1 0 0 0
2.0.0.2 1 Full/Backup 2.0.0.2 mila:2.0.0.1 0 0 0
```

Here, we see that the router has established two OSPF sessions with two neighbors. The first one is connected via the interface 1.0.0.1 and its IP is 1.0.0.2. The second one is connected via the interface 2.0.0.1 and its IP is 2.0.0.2. Now that you are connected to two routers with OSPF, they can send you information about the topology of the network. Let's take a look at the routes received by OSPF with the following command.

```
router# show ip route ospf
0 1.0.0.0/24 [110/10] is directly connected, pari, 07:09:33
```

```
0 2.0.0.0/24 [110/10] is directly connected, mila, 06:14:24
0>* 10.104.0.0/24 [110/20] via 2.0.0.2, mila, 00:00:10
```

You can see that our router has learned how to reach the subnet 10.104.0.0/24. The 0 at the beginning of each line indicates that the router has learned this subnet from OSPF. To reach it, it must send the packets to its neighbor router 2.0.0.2. If you want to have more information about the routers of this OSPF area, you can use `show ip ospf database`. If you want to advertise additional directly connected networks, you should take a look at the `redistribute` command.

With OSPF, each link between two routers has a specific weight, and only the shortest paths are used to forward packets. Below is an example of how you set the weight of a link connected to interface `interface_name` to 900:

```
router# conf t
router(config)# interface interface_name
router(config-if)# ip ospf cost 900
```

Use these commands to configure all the OSPF weights in your own network.

2.5 Configure BGP

While OSPF is used to provide IP connectivity within an AS, BGP is used to advertise prefixes between different ASes. Unlike OSPF, BGP routers will not automatically establish sessions. Each session needs to be configured individually. The following commands show you how to: start a BGP process and establish two BGP sessions with neighboring routers. The integer following `router bgp` indicates your AS-number. Here, the local AS-number is 2:

```
router# conf t
router(config)# router bgp 2
router(config-router)# neighbor 150.0.0.1 remote-as 15
router(config-router)# neighbor 2.0.0.2 remote-as 2
```

By default, whenever the `remote-as` is different from the local number (here 2), the BGP session is configured as an external one (i.e., an eBGP session is established). In contrast, when the `remote-as` is equivalent to the local one, the BGP session is configured as an internal one (iBGP). Here, the first session is an eBGP session, established with a router in another AS (150.0.0.1), while the second one is an internal session, established with a router within your AS (2.0.0.2). You can check the state of your BGP sessions using the following command:

```
router# show ip bgp summary
Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/PfxRcd
2.0.0.2 4 2 3 6 0 0 0 00:01:16 0
150.0.0.1 4 15 2009 1979 0 0 0 01:31:42 1
```

You can see that our two BGP sessions are up. To advertise a prefix on a BGP session, you can just use the following command:

```
router(config-router)# network 10.104.0.0/24
```

As soon as you do that, your neighbor 150.0.0.1 receives this advertisement and knows that it can forward to you all the packets with a destination IP in the subnet 10.104.0.0/24. Note that when you advertise a prefix, it generally means you know how to reach this prefix, otherwise you will drop the packets, and your neighbor will not be very happy. Perhaps one of your BGP neighbor advertises you prefixes as well, let's check that:

```
router# sh ip route bgp
B>* 2.101.0.0/24 [20/0] via 2.0.0.2, interface_used, 00:03:17
```

In this case, we can see that neighbor 2.0.0.2 has advertised one prefix: 2.101.0.0/24. The B

at the beginning of the line indicates that the router has learned this prefix from BGP.

Similarly to OSPF, you can advertise additional subnets with the `redistribute` command. You can redistribute the directly connected subnets into BGP with `redistribute connected`. You can also redistribute routes learned by OSPF with `redistribute ospf`. Always pay attention which prefixes you advertise to your neighbors. They may not need to know all the prefixes used in your local network.

Important (iBGP connections): By default, when a router establishes an iBGP session with a peer, it uses the IP address of the interface closest to the iBGP peer as source address. This is fine as long as this interface is up. If this interface goes down but the actual router is still running, the iBGP connection will tear down even though the router may be reachable over a different path/interface. In practice, operators therefore often use so called “loopback” interfaces as source of the iBGP connections. Loopback interfaces are virtual interfaces with an IP address which uniquely identifies the router in your network. To save you some time, you do not have to configure additional loopback interfaces/addresses. Instead, you can use the host interface (each router is connected to a host) as loopback interface. For example, if you are group 7 and you want to configure an iBGP session from MILA to MUNI, you could use the following commands:

```
router# conf t
router(config)# router bgp 7
router(config-router)# neighbor 7.107.0.2 remote-as 7
router(config-router)# neighbor 7.107.0.2 update-source host
```

7.107.0.2 is the host interface IP of MUNI (our “loopback address”) and with the command `update-source host` you make sure that the MILA router is using its host interface address as source address.

BGP next-hop-self Figure 1 illustrates a possible pitfall when eBGP announcements are distributed over iBGP sessions. If router R1 is advertising its prefix 11.22.33.0/24 towards R2 (over the eBGP session), the next-hop of the advertisement is the IP of the outgoing interface of R1 (10.0.0.2). Router R2 is then distributing this advertisement to all its iBGP neighbors (R3). Per default, the next-hop of the advertisement is not changed and is still 10.0.0.2. That can lead to problems for router R3. In a normal network configuration, the subnets used between two eBGP routers (10.0.0.0/30) are not distributed in the connected ASes, e.g. via OSPF. Router R3 does therefore not know how it can reach the next-hop of the advertised prefix (11.22.33.0/24) and will discard the advertisement. To solve this problem, we can use the BGP `next-hop-self` command. When properly configured, router R2 is using its own interface IP as next-hop every time it distributes announcements from eBGP sessions to its iBGP neighbors. In this example, R2 would replace the next-hop 10.0.0.2 with its interface IP connected to R3 (20.0.0.1). Router R3 then receives an advertisement for prefix 11.22.33.0/24 with next-hop 20.0.0.1. The 20.0.0.0/30 subnet is distributed via OSPF and R3 does therefore know how to reach the next-hop of the advertisement and can install this prefix in its forwarding table.

To add the `next-hop-self` command on Quagga routers, you can use the following commands (on router R2):

```
router# conf t
router(config)# router bgp 100
router(config-router)# neighbor 20.0.0.2 remote-as 100
router(config-router)# neighbor 20.0.0.2 next-hop-self
```

2.6 Configure BGP policies

Now that you have BGP running, you want to configure some BGP policies. For example, you may want to prefer to send traffic to your cheapest provider. You may also want to avoid sending traffic to one AS for a particular prefix for security reasons. BGP offers several ways to configure such policies. As we have seen in the lecture, the `LOCAL-PREF` attribute can help you to influence the outbound routing decisions. To influence the inbound routing decisions, you can use `AS-Path`

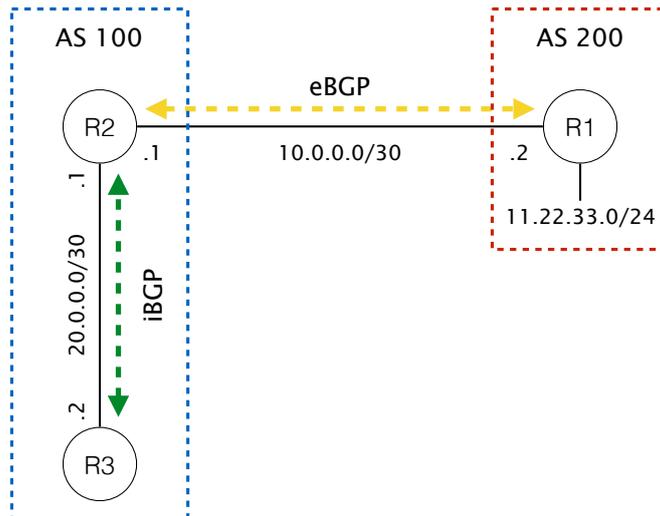


Figure 1: AS 100 and AS 200 have an eBGP session between each other. In AS 100 there are two routers with an iBGP session. Router R3 is only able to reach the prefix 11.22.33.0/24 if router R2 is using BGP next-hop-self for the iBGP session.

prepending, a more specific prefix advertisement or the MED attribute (in case of multiple different connections to the same AS neighbor). In Quagga, BGP policies are defined using `route-maps`. In the following section we describe how a Quagga route-map is configured and applied to certain BGP sessions. Then we look at BGP community values. Community values are a good way to mark certain BGP routes. At the end of the section we present an example of a route-map using BGP community values.

Route-maps Route-maps enable you to express input and output filters that will be applied immediately respectively, immediately after a BGP route has been received from a neighbor or just before it is sent to a neighbor. A route-map is composed of two parts: a *match* and a *set* part. This can somehow be compared to the *If...Then...* statements in many programming languages. The *match* part is a boolean predicate that decides on which BGP routes the route-map applies the actions (usually, attributions manipulation) defined in the *set* part.

Match part Let's take a look on what you can *match* on:

```
router# conf t
router(config)# route-map MY_ROUTE_MAP permit 10
router(config-route-map)# match ?
```

<code>as-path</code>	Match BGP AS path list
<code>community</code>	Match BGP community list
<code>interface</code>	Match first hop interface of route
<code>ip</code>	IP information
<code>metric</code>	Match metric of route
<code>origin</code>	BGP origin code
<code>peer</code>	Match peer address

As you can see, you can match pretty much any attribute contained in a BGP UPDATE including: the AS-PATH, community value, IP addresses or subnets, the peer IP address, etc.

Whenever you create a **route-map**, you must indicate if you want to permit or deny the input data. Intuitively, only permitted routes will go through the set of actions, while denied routes won't be considered further. Route-maps can also be chained together. The order in which route-maps are processed is given by the sequence number (here 10). The **route-map** with the lowest sequence number is executed first. If you have no match part, the set part is applied to all BGP routes.

Set part Now let's take a look on what you can do with the matched routes using the *set* part of the route-map:

```
router# conf t
router(config)# route-map MY_ROUTE_MAP permit 10
router(config-route-map)# set ?
```

as-path	Transform BGP AS-path attribute
community	BGP community attribute
ip	IP information
metric	Metric value for destination routing protocol
local-pref	BGP local preference path attribute
origin	BGP origin code

As described in the course, the set part enables you to modify any route attribute. Among others, you can: set a local-preference, modify the AS-path (to perform AS-path prepending), or set a community attribute. Observe that these operations will only be done on the routes you have matched with the **match** command if you used the *permit* clause. A set clause can be empty, which is useful if you only want to filter certain routes (these which pass the match part). Finally, you can also modify multiple route attributes in the same set clause.

Apply the route-map Once you have defined a route-map, you need to apply it to a BGP session. As mentioned before, a route-map can either be applied on incoming routes (routes received from a neighbor) or on outgoing routes (routes sent to a neighbor). To apply the route-map you have to extend your BGP configuration (see section 2.5). In the following example, the route-map MY_ROUTE_MAP is applied to all incoming routes from neighbor 2.0.0.2:

```
router# conf t
router(config)# router bgp 15
router(config-router)# neighbor 2.0.0.2 route-map MY_ROUTE_MAP in
```

If you use *out* instead of *in*, the route-map is applied to all outgoing routes instead.

BGP community values A BGP community can be seen as a label or a tag that can be attached to any route in the set part of a route-map and matched against later on a different router. In practice, a community value is often written as two integers separated by a colon. The first integers identifies the AS and the second one describes the value/purpose of the community value. For example, if you are AS number 15 and you want to use the community value to tag certain BGP routes with the value "100", you could use the following BGP community value: 15:100. You can also add more than one BGP community value to a route.

If you want to use community values in the match part of a route-map, you need to use a so called "community list". To defined the community value from before (15:100) you can use the following command:

```
router# conf t
router(config)# ip community-list 1 permit 15:100
```

The community list has the name/number 1.

Example Let's assume you have a router which has two eBGP connections to two different neighbors with the IPs 11.11.11.11 and 22.22.22.22. You want to set the local preference of all routes coming from 11.11.11.11 to "1000". Furthermore, you want to use BGP community values to make sure that only the routes coming from neighbor 11.11.11.11 are advertised to neighbor 22.22.22.22:

```
router# conf t
router(config)# route-map MAP_IN permit 10
router(config-route-map)# set community 15:100
router(config-route-map)# set local-preference 1000
router(config-route-map)# exit
router(config)# ip community-list 1 permit 15:100
router(config)# route-map MAP_OUT permit 10
router(config-route-map)# match community 1
router(config-route-map)# exit
router(config)# router bgp 15
router(config-router)# neighbor 11.11.11.11 route-map MAP_IN in
router(config-router)# neighbor 22.22.22.22 route-map MAP_OUT out
```

Final remarks The `show ip bgp` command is very useful to display all prefixes/routes, for example to check that your route-maps are working correctly. You can also filter the output. If you want to display all the prefixes/routes with a certain BGP community value (e.g. 15:100), you can use the following command:

```
router# sh ip bgp community 15:100
```

Remember that you can always use `?` to show all available commands (e.g. `show ip ?`). Sometimes BGP and Quagga take time to converge. If so, you can use `clear ip bgp *` to force the convergence.

And finally, whenever you use a `route-map`, the following `route-map` is implicitly added:

```
router(config)# route-map implicit-default-route-map deny 65536
router(config-route-map)# match everything
```

This `route-map` denies everything and makes sure that only the routes which pass the match clauses of your route-maps are accepted.

This section only sketched what `route-maps` have to offer. In this assignment, one of your goals will be to get more familiar with them so as to implement the right routing decisions.

References

- [1] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 117–130. [Online]. Available: <https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff>
- [2] Quagga Routing Suite - Documentation. [Online]. Available: <http://www.nongnu.org/quagga/docs/docs-info.html>