University of Crete
Computer Science Department
**Lecturer:** Prof. X. Dimitropoulos

Autumn Semester 2022
HY335B - Computer Networks

# Network Configuration Tutorial

This tutorial contains some useful instructions for you to configure the network. §1 tells you how you can access the different devices. §2 shows you how you can maintain your configuration files. §3, §4, and §5 present how you can configure a host, Open vSwitch, and FRRouting IP router, respectively. Finally, §6 describes how you can perform throughput measurements in our mini-Internet.

## 1 Accessing your devices

In this section, we show you how you can access (i) your docker container, and from there, (ii) your FRRouting routers, Open vSwitches, and hosts, such that you can configure them.

### 1.1 Accessing your docker container

To make your life easier, you do not need to run the virtual network inside your laptop. Instead, we have setup a remote container for each group where your virtual network already runs and desperately awaits you to configure it. We have also setup our server so that each AS is connected to its neighboring ASes only.
In the following, we describe how you can connect to your container using SSH and how you can copy directories from the container to your own machine.

**Connecting to your container with SSH** To access your container, you will use SSH. SSH is a UNIX based command interface and protocol for securely getting access to a remote computer. It is widely used by system administrators to control network devices and servers remotely. An SSH client is available by default on any Linux and MAC installation through the Terminal application. For Windows users, a good and free SSH client is PuTTY[1]; alternatively, you can use Windows Subsystem for Linux[2]. Once you have installed an SSH client, use the following command to connect yourself to your container:

```
> ssh -p X 147.52.203.13
```

where $X = 2000 + group\_number$. For instance if you are group 7, use the following command:

```
> ssh -p 2007 147.52.203.13
```

We have sent you your password and group number by email or message by Microsoft TEAMS. If you cannot connect to your container, please report it immediately in the Moodle forum.

### 1.2 Accessing your routers, switches and hosts

The following paragraphs describe how you can access your routers, switches and hosts from your container.

**Access to routers** When you are in your container, you can use the script `goto.sh` to connect to a router, a switch or a host. For example, with the following command, you will access the

[1] `https://www.chiark.greenend.org.uk/~sgtatham/putty/`
[2] https://docs.microsoft.com/en-us/windows/wsl/install-win10

router NEWY:

```
> ./goto.sh NEWY router
```

Now you are directly in the CLI of the router NEWY. We describe the FRRouting CLI in more detail in §5.1.

**Access to switches** Similarly, you can access a switch with the `goto.sh` script. Simply use `UNIV` as the first argument for the script, and the switch name as the second argument. An example for the CERN switch:

```
> ./goto.sh UNIV CERN
```

Unlike FRRouting, there is no CLI for Open vSwitch. Hence, after you have run the above command, you are not *directly on* (an interface of) the switch, but in an instance that *contains* and simulates the switch. In other words, what has changed is that you are now in a different terminal from which you can externally configure the simulated switch by running OpenvSwitch commands (see §4).

**Access to hosts** From your container, you can also go to a host. For example, if you want to go to the host which is connected to BOST, just use the following command:

```
> ./goto.sh BOST host
```

This works for all router names except ZURI and GENE, which are not connected to hosts but the layer-2 network.
If you want to access the student host connected to EPFL in the Swiss local network, you can use the following command (see also the layer-2 topology in the assignment):

```
> ./goto.sh UNIV student_3
```

If you want to access a staff host, just replace `student` by `staff`.

**Important:** When you want to exit from a host, a switch or a router, just type exit in the terminal.

# 2 Maintaining your configuration

To save the configuration of all your routers and switches, we provide you with a script called `save_configs.sh`. Once executed (`./save_configs.sh`) it will generate two equal folders following the naming convention: `configs_[date]_[time]`. One folder is already zipped for a simpler download. We advise you to run this script regularly and to save the generated folders on your local machine to have a backup of your work.

# 3 Configuring a host

This section first describes the naming conventions inside hosts for this project, and secondly, how to configure the IP addresses and interfaces on a host.

**Naming conventions** The interface to the router is called `<router-name>router`, e.g. `BOSTrouter` for BOST. The loopback interface has the name `lo`.

**Reading the configuration** You can see the interfaces with `ip address show`.

**Changing the configuration** The host configuration is rather simple. First, you have to assign an IP address and subnet to one of the interfaces available on the host. For that, you can use the following command:

```
> ip address add IP/SUBNET_SIZE dev INTERFACENAME
```

For example, with

```
> ip address add 111.0.222.3/24 dev LONDrouter
```

you assign the IP address `111.0.222.3` inside the corresponding /24 subnet to the interface called `LONDrouter`. Note that we configure the IP address and the corresponding subnet all at once.
At this point, the host knows how it can reach all the IP addresses inside the subnet `111.0.222.0/24`. Unfortunately, we cannot reach any other IP. (The subnet declaration tells the host in what subnet it is; the IP address in turn is the address by which other devices can reach this host.) Hence, you normally add a route towards a default gateway that the host uses to reach all unknown IP addresses. To do that, use the command:

```
> route add default gw IP_ADDRESS INTERFACE_NAME
```

Coming back to our previous example, to add a route towards the router (using the interface 1-CERN which connected to the Internet) with IP address `111.0.222.1`, we could use the following command:

```
> route add default gw 111.0.222.1 1-CERN
```

To see the current default gateway, use

```
> netstat -rn
```

To delete past entries, use

```
> route del default gw IP_ADDRESS INTERFACE_NAME
```

In our example, in case we want to remove the default gateway in the above host, we should type

```
> route del default gw 111.0.222.1 1-CERN
```

# 4 Configuring Open vSwitch

Open vSwitch[3] [1] is one of the most popular software switches. It can typically be used in virtual environments, for instance to connect two virtual machines.

The next sections present naming conventions for switch ports in this project, some useful commands to get an overview of the switch state and configuration, and how to change the VLAN configuration.

## 4.1 Naming conventions

On each switch, one port has the name `br0` and the corresponding interface has type `internal`. This is a local port used by the host to communicate with the switch. You do **not** need to use this port. The ports to student and staff members follow the name pattern `X-student_i`, `X-staff_i` respectively, with X the AS number and i the student/staff number (e.g. `84-student_2` for student 2 in AS 84). The ports to other switches have the name `X-SWITCHNAME`, e.g. `84-ETHZ` for the port to ETHZ in AS 84. If there is a port to a router, it is called `ROUTERNAMErouter`, e.g. `GENErouter`.

[3] https://www.openvswitch.org

### 4.2 Reading the current switch state and configuration

**Brief switch overview** 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.

**Port status** To get more precise information about the status of the ports, you can use the following command:

```
> ovs-ofctl show br0
```

(Please note that the bandwidths in the output of this command do not correspond to the actual bandwidths used in the assignment.)

**Current configuration and statistics** 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 dump
```

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

```
> 645981b6c 0 false [] 0  false [119f5-2be8bf5] [] [] 84-ETHZ {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 `84-ETHZ` and this port is a trunk port in VLANs 10 and 20. Please note that `grep` [2] is also available to further filter the output.

### 4.3 Changing the VLAN configuration

Below are some commands to adapt the VLAN configuration of a switch.
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
```

Otherwise, to clear a specific VLAN tag from port PORT_NAME, you can type:

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

## 5 Configuring IP routers

FRRouting (FRR) routers [3] are software routers running on top of Linux. In other words, running FRR on Linux allows you to tranform your laptop or server into an IP router, where the Linux interfaces act as interfaces for the FRR router. You can configure a FRR router through a Command Line Interface (CLI). Each IP router vendor (e.g., Cisco or Juniper) or

software routing suite (e.g., FRR) has its own CLI. Fortunately, those CLIs are similar, and if you know how to configure a router using the FRR CLI, you can easily configure a Cisco or a Juniper router as well.

### 5.1 The FRRouting command line interface

We now briefly describe how to configure a FRR router[4]. The following paragraphs describe the autocompletion features of the CLI, how to test connectivity, how to switch to configuration mode, and how to remove a command/reverse your configuration actions.

**Autocompletion features** When you enter the FRR 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 (some of the shown commands are):

```
router# ?

     clear           Reset functions

     configure       Configuration from vty interface

     exit            Exit current mode and down to previous mode

     no              Negate a command or set its defaults

     ping            Send echo messages

     quit            Exit current mode and down to previous mode

     show            Show running system information

     traceroute      Trace route to destination

     write           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.[5] 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 tabulator key.

**Testing connectivity** If you want to test your connectivity, you can use `ping` or `traceroute` (also with the measurement container as mentioned in the assignment). Please note that you cannot run traceroutes starting from a tier-1 AS.

**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. Commands that work *in* configuration mode do not (necessarily) work *outside* configuration mode and vice versa. Notably, commands starting with `show` do not work in configuration mode.

**Reverse a command** If you want to delete parts of the configuration, you can prefix the command you want to remove with `no`.

[4]This is a very short FRR introduction, we strongly recommend you to take a look at the official documentation [4] to get more information.

[5]The interfaces are not visible in the output at the beginning, but will start appearing as soon as you have configured them.

## 5.2 Configure the router interfaces

In order to explain how to configure interfaces, we will first describe their purpose, and then tell you about the naming conventions for this project. Next, we will show you how to read and change the interface configuration.

**The purpose of an interface** 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 subnet[6].

**Interface naming conventions** Each router has interfaces to its neighbouring routers whose names follow the pattern `port_<neighbor>`. For instance, the interface on NEWY connected to ATLA is named `port_ATLA`. Moreover, each router has an interface to the host named `host` and a loopback interface called `lo` (exceptions: GENE and ZURI do not have a host). An interface connecting to another AS is called `ext_<AS-number>_<router-name>`. For example, the interface on NEWY connecting to ZURI in AS 82 has the name `ext_82_ZURI`.

**Reading interface configurations** To get an overview of the interfaces, use the command

```
router# show interface
```

or for a a briefer overview

```
router# show interface brief
```

You can get information for one specific interface with

```
router# show interface INTERFACENAME
```

**Changing interface configurations** 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 INTERFACENAME
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`. 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 INTERFACENAME
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, INTERFACENAME
```

We see that 2.0.0.0/24 is directly connected and reachable with the interface `INTERFACENAME`. At this stage, a packet with a destination IP that is not in a directly connected subnet will be

[6] Try to think what would happen if that wasn't the case

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.

### 5.3 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
```

### 5.4 Configure OSPF

OSPF routers flood IP routes over OSPF adjacencies. FRR routers continuously (and automatically) probe any OSPF-enabled interface to discover new neighbors to establish adjacencies with. By default, FRR 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 area 0.
To check the OSPF neighbors of a router, you can use the following command:[7]

```
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 port_ATLA:1.0.0.1 0 0 0
    2.0.0.2 1 Full/Backup 2.0.0.2 port_BOST:2.0.0.1 0 0 0
```

Here, we see that the router has established two OSPF session 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 at look at the routes received by OSPF with the following command.

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

[7] The `Neighbor` may be a different IP than you expect since it is an ID that may come from any interface, e.g. also the `ssh` interface. Do not worry about this.

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

You can see that our router has learned how to reach the subnet 10.104.0.0/24. The `O` 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 `INTERFACENAME` to 900:

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

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

### 5.5 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. For example, if you are group 7 and you want to configure an iBGP session from NEWY to LOND, you could use the following commands (on NEWY):

```
router# conf t
router(config)# router bgp 7
router(config-router)# neighbor 7.151.0.1 remote-as 7
router(config-router)# neighbor 7.151.0.1 update-source lo
```

7.151.0.1 is the loopback interface IP of LOND. With `update-source lo` you make sure that the NEWY router is using its lo interface address as source address.

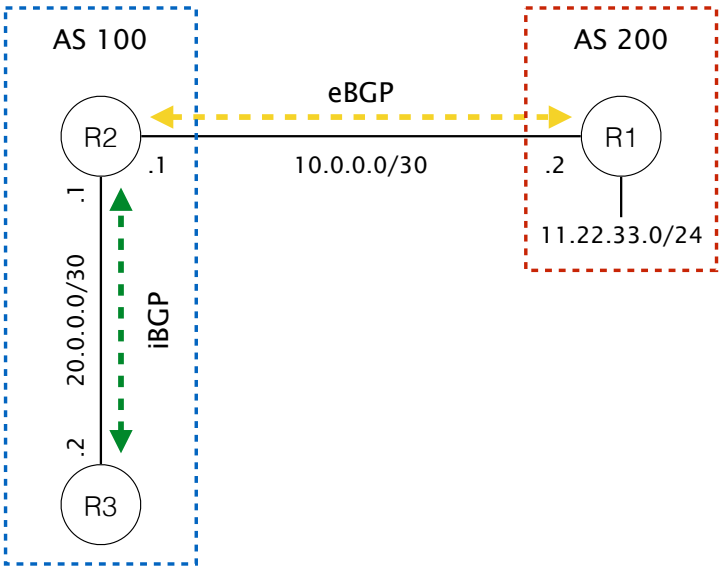


Figure 1: AS 100 and AS 200 have an eBGP session between each other. In AS 100 there are two routers with an iBGP session. Router R3 is only able to reach the prefix 11.22.33.0/24 if router R2 is using BGP next-hop-self for the iBGP session.

**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 FRR 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
```

### 5.6 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` prepending, a more specific prefix advertisement or the `MED` attribute (in case of multiple different connections to the same AS neighbor). In FRR, BGP policies are defined using `route-maps`. In the following section we describe how a FRR 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, attribution manipulation) defined in the *set* part.

**Match part** Let's take a look on what you can *match* on (most important options are shown):

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

     as-path         Match BGP AS path list
     community       Match BGP community list
     interface       Match first hop interface of route
```

```
ip              IP information
metric          Match metric of route
origin          BGP origin code
peer            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 (most important options are shown):

```
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 5.5). 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 integer 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 FRR 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.

# 6 iperf3 Throughput Measurements

`iperf3`[5] is a program to perform network throughput measurements. It works by setting up a connection between two endpoints (a client and a server) and then transferring a stream of data between them.

`iperf3` is installed on all hosts in our mini-Internet. To perform a measurement between two hosts, you need to start an iperf3 server on one of the hosts and then start an iperf3 client on the other host.

To start an `iperf3` server, you can use the following command:

```
> iperf3 --server --one-off
```

The flag `--server` instructs `iperf3` to start in server mode. The `--one-off` flag makes `iperf3` exit automatically after it handled the connection and measurement of one client.

To start an `iperf3` client, you can use the following command:

```
> iperf3 --client SERVER_IP --time TIME
```

The flag `--client` instructs `iperf3` to start in client mode. Then, you need to provide the client with the IP address on which the server is running. The `--time` flag determines how long the measurement will run in seconds.

For example, if you were group 8 and wanted to measure the throughput between MIAM and BOST, you could start a server on the host connected to BOST and a client on the host connected to MIAM:

On the host connected to BOST, you would issue the following command:

```
> iperf3 --server --one-off
```

And on the host connected to MIAM, you would issue the following command:

```
> iperf3 --client 8.106.0.1 --time 5
```

# 7 Acknowledgments

**This tutorial is based on material from a similar course at ETH Zurich [6].**

# 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] grep. [Online]. Available: http://man7.org/linux/man-pages/man1/grep.1.html

[3] FRRouting. [Online]. Available: https://frrouting.org/

[4] FRRouting User Guide. [Online]. Available: http://docs.frrouting.org/en/latest/

[5] iperf3 - perform network throughput tests - Manpage. [Online]. Available: https://manpages.ubuntu.com/manpages/xenial/en/man1/iperf3.1.html

[6] ETH Zurich, Communication Networks. [Online]. Available: https://comm-net.ethz.ch/