Multicast using PIM-Sarse-Mode

Author's notes

1. If you don't know how to make the project work don't hesitate to type `./help.sh` for more details on various commands and more
2. Remember to first use `sudo chmod 777 permissions.sh` and then run `./permissions.sh` to give execute permissions to all the scripts in the project
3. It is recommended to view this project in VsCode using Remote Explorer extension if you are connecting to a Virtual Machine via SSH, it makes it easier to download the video output, navigate the configuration files and read this README (but you don't have to !)

Project objectives

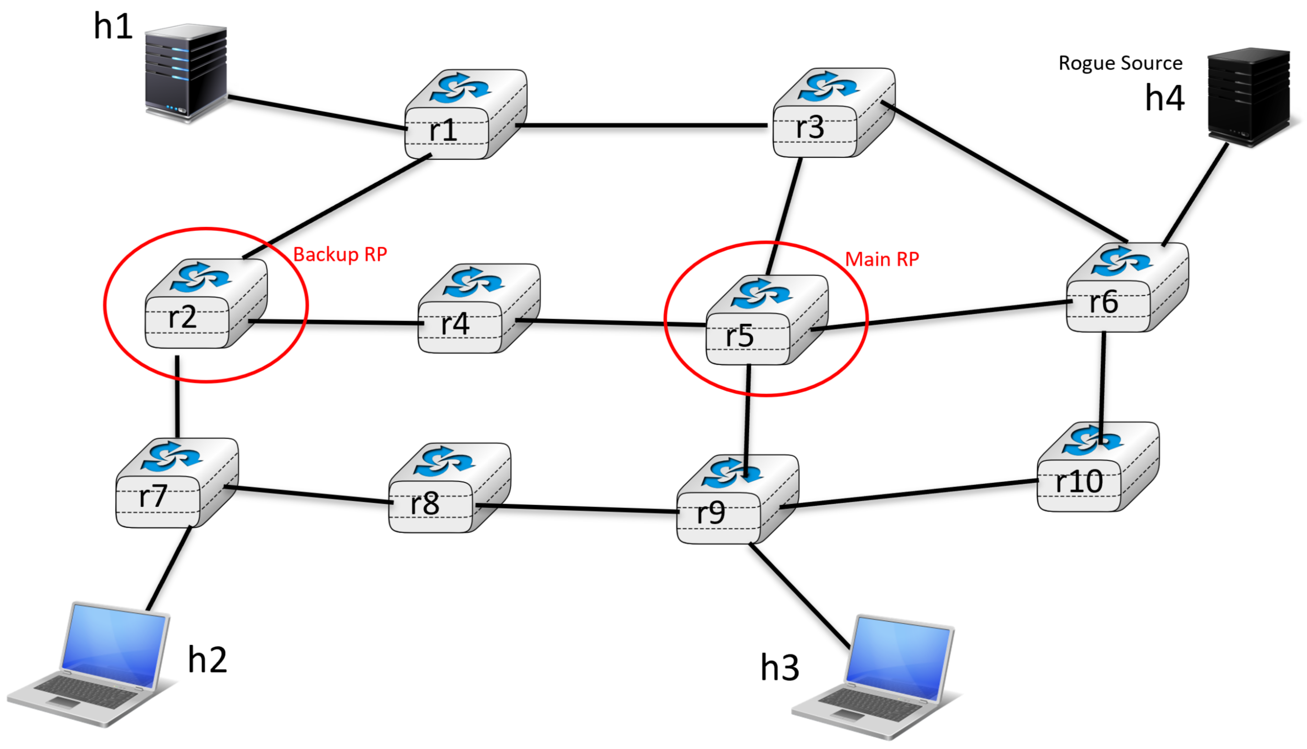
The objective of this project was to create a lab environment using FRRouting that supports PIM-Sparse-Mode and consisting of at least one sender/server streaming a video and several receivers/clients displaying it using `ffmpeg` to send a video stream via multicast and allowing multiple devices to access the same content without overloading the network.

We also had to choose a central location for the network Rendezvous Point (RP) and deploy security measures to avoid Rogue Servers and RP to attack the network/disrupt it.

For our multicast group prefix, we chose ff06::178/127 that was shown as available among the official reserved IPv6 addresses for multicast available on the IPv6 Multicast Address Space Registry.

In our scripts for streaming and subscribing to streams we used ff06::179 to show that both ff06::178 and 179 are available (you can check on the aforementioned website).

Network topology



Rendezvous Point (RP)

Rendezvous Point (RP) location choice

We decided the Rendezvous Point should be at the crossroad of as many routers as possible for maximum efficiency in redistributing the multicast packets between the server and the clients and in the case of our current network we found 'r5' to be perfect for this role. Additionally we added r2 as a backup RP should r5 fail unexpectedly.

Testing the backup Rendezvous Point

Run the script `./test_RP.sh` to verify that 'r2' does indeed take over 'r5' RP duties in case of failure. In this test scenario we manually break 'r5' by shutting it down and checking which router is detected as RP by every router in the network.

BSR location choice

To ensure the stability of the Rendezvous Point advertisement and because of its proximity with the RendezVous Point, we gave priority to r4 to be the bsr.

Security policies

Against Rogue RP

Our hosts can't be candidate bsr and can't receive candidate bsr advertisement from routers. This way, only routers can know who is the RendezVous Point and propagate the information in the network. We put the lowest priority on r5 to make sure he will be chosen as RendezVous Point.

Against Rogue Sources

We limited the authorized incoming hosts to the ones already registered in the network (h1) with a firewall using nftables. To do so, we only allow 'h1' to send messages to the multicast group range. He's the only host able to stream on the network. This way, 'h4' who's trying to be a server, gets his stream to the multicast group rejected.

Testing the protection against Rogue sources

Try running `./stream.sh h4`, and you will see that h4 does not have permission to start a stream to the multicast group.

Sometimes it seems the firewall fails to deploy the rules in the right order. If the firewall seems faulty and you see that the stream from h1 returns mutltiple "Operation not permitted" errors you might have to restart the containers yourself with `./run.sh`.

Project structure, scripts/commands and how to run them

General project structure

```
├── clab-igp
│   ├── h1 ─ frr.conf
│   ├── h2 ─ frr.conf
│   ├── h3 ─ frr.conf
│   ├── h4 ─ frr.conf
│   ├── r1 ─ frr.conf
│   ├── r2 ─ frr.conf
│   :   :
├── startup_files
│   ├── daemons
│   ├── vtysh.conf
│   ├── host.dockerfile
│   ├── router.dockerfile
│   ├── input.mp4
│   ├── execute_nftables.sh
│   ├── network_ready.sh
│   ├── nftables.sh
│   ├── save.sh
│   └── startup.sh
├── network_topology.png
├── topo.clab.yml
├── README.md
├── README.pdf
├── help.sh
├── network_check.sh
├── open.sh
├── permissions.sh
├── run.sh
├── stream.sh
├── test_RP.sh
└── watch_and_record.sh
```

Project root user commands

These scripts are here to help the user navigate the containers and execute commands in a simplified way. Feel free to browse through them to see how we run things such as streams, RP shutdown, etc

Remember you can type `./help.sh` at any moment to see the following.

```
List of scripts available in this project :
> ./help.sh
        Shows the currently shown list of scripts available in the project
> ./permissions.sh
        Gives execute permissions to all other bash scripts in the project
        Don't forget to first give permissions to this script with 'sudo chmod
777' before trying to run it
> ./run.sh
        Starts/restarts the containerlab then notifies the user when the firewall
is deployed and the routers are ready to transmit multicast packets
> ./open.sh
        Opens any container in the network
        Usage: ./open.sh [CONTAINER] [INTERFACE (optional)]
        Example 1: ./open.sh h1
        Example 2: ./open.sh r2 bash
> ./stream.sh
        Starts a stream on the specified server
        If the server trying to start a stream is considered external to the
network it will be stopped by the firewall and an error will be thrown
        Usage: ./stream.sh [SERVER_NAME]
        Example: ./stream.sh h1
> ./watch_and_record.sh
        Watches and records the video stream coming from the server onto the
specified client
        The resulting video recording is then downloaded to the project root where
you can watch it
        Usage: ./watch_and_record.sh [CLIENT_NAME] [DURATION (optional and > 5
seconds)]
        Example 1: ./watch_and_record.sh h3
        Example 2: ./watch_and_record.sh h2 42
> ./network_check.sh
        Checks if the routers are detecting the current network Rendezvous Point
and shows who the BSR and RP are
> ./test_RP.sh
        Verifies that 'r2' successfully takes over as a backup RP if the original
RP 'r5' breaks down
```

Startup_files background commands

These scripts are used automatically when using `./run.sh` (except `save.sh` but we didn't consider it a user command). You can browse them to see how we run the containers, start the firewall using nftables, create the images for routers and hosts, etc

- `daemons` : This file tells the frr package which daemons to start (isis, pimd)

- `vtysh.conf` : Configuration file for vtysh
- `host.dockerfile` : Dockerfile defining the image for the host containers
- `router.dockerfile` : Dockerfile defining the image for the router containers
- `input.mp4` : Input video file to test streaming over the network
- `execute_nftables.sh` : Launches `nftables.sh` containing the rules for the hosts firewalls
- `network_ready.sh` : Script launching `network_check.sh` to tell the user when the network has settled and the routers are ready to transmit multicast packets
- `nftables.sh` : Writes nftables rules for the hosts firewalls
- `save.sh` : Script for saving frr configuration from the containers to the local save location in clab-igp/
- `startup.sh` : Used to create the images when starting the containers

Sending a video stream via multicast

The following shows more in details how to send a video stream from a server to one/multiple clients.

Start a stream on the server of your choice

```
./stream.sh [SERVER_NAME]
```

This will start a stream on the server `SERVER_NAME` if it is a known server (in this case only h1 since h4 will be blocked by the firewall see above).

Sometimes it seems the firewall fails to deploy the rules in the right order. If the firewall seems faulty and you see that the stream from h1 returns mutltiple "Operation not permitted" errors you might have to restart the containers yourself with `./run.sh`.

Start watching and recording the video stream on clients of your choice

```
./watch_and_record.sh [CLIENT_NAME] [DURATION (optional and > 5 seconds)]
```

This will subscribe the client `CLIENT_NAME` to the stream, record the stream into a video during `DURATION` seconds (must be > 5 seconds) or 15 seconds by default and finally save it to the project root.

You can run this command in two different terminals to simulate both clients h2 and h3 watching the same stream at the same time.

Example of execution

Start a stream on h1

```
./stream.sh h1
```

Start watching and recording on h2 in terminal 1 for default 15 seconds

```
./watch_and_record.sh h2
```

Start watching and recording on h3 in terminal 2 for 42 seconds

```
./watch_and_record.sh h3 42
```

Download the video output from this folder to your physical computer

- Using VSCode Remote-SSH : simply right-click the video file and save it at the location of your choice
- Using `sftp` (the file will be saved in your user folder) :

```
sftp <NAME_OF_YOUR_VIRTUAL_MACHINE>
sftp> <project_folder_name>/<name_of_video_file>.mp4
```

Open the video with your favourite video player

Note that you might need to install MMPEG-2 codec in order to read it. If you can't make it work try using VLC media player. The forced MMPEG-2 formatting is due to `ffmpeg` way of transmitting video files

Enjoy the video !

Various show commands for unicast and multicast and reading the firewall rules

Reminder for entering a container the easy way !

You can access any container using `./open.sh [CONTAINER] [INTERFACE (optional)]` (for more examples use `./help.sh`).

Testing unicast connections with `ping` (or `traceroute`)

From any router towards any other router X :

```
ping fc00:2142::X
```

From any router towards h1 :

```
ping fc00:2142:1::
```

From any router towards h2 :

```
ping fc00:2142:7::
```

From any router towards h3 :

```
ping fc00:2142:9::
```

From any router towards h4 :

```
ping fc00:2142:6::
```

Seeing multicast informations on a router

Show how many packets and bytes passed through the router so far

```
show ipv6 multicast
```

Show which router is the RP and the group prefix

```
show ipv6 pim rp-info
```

Alternatively if you only want to see who the RP is you can simply run `./network_check.sh` to see who is detected as the RP.

Show which router is the BSR (the one who distribute the information about the RP's)

```
show ipv6 pim bsr
```

Alternatively you can simply run `./network_check.sh` to see who is the BSR.

Checking nft tables rules on a host

```
nft list ruleset
```

Alternatively the entire list of implemented rules can also be seen in `/startup_files/nftables.sh`

To prevent the firewall not taking into account some of the rules by mistake. If an error occurs, we relaunch the creation of the containers.

Authors

KHEIRALLAH Cédric & SHAFIEI Tania