Lab 2: Video Streaming over Legacy Networking

Advanced Computer Networks
Master 1 2024-25
Université Côte d'Azur
Instructor : Ramón APARICIO (ramon.aparicio-pardo@univ-cotedazur.fr)

2nd April 2025

1 Objective

This laboratory aims that you understand how a SDN can help to solve in a flexible and configurable manner some issues existing in legacy networks.

2 Rules

You can work in groups of 1 or 2 students.

You have to deliver : (i) some source code, and (ii) a final report with the answers of the Lab 2&3 of the lab by April 27th 2025.

3 Before the lab

For the labs, we will use Containernet, a fork of the famous Mininet network emulator, which allows using Docker containers as hosts in emulated network topologies.

You will need to install it in your Ubuntu VM following these instructions. I recommend you the Option 1: Bare-metal installation. I didn't check the *Option 2 : Nested Docker deployment*.

I suggest you that you install `Containernet` in your `/home/USERNAME` folder.

4 Configuring the network

In this laboratory activity, we are going to work with the topology shown in Fig. 1 representing a legacy network : a current state-of-art network not based on the SDN paradigm.

As you see in Fig. 1, we are going to build the legacy network by using `Containernet` and two containers. `Containernet` will implement the network itself (the routers and the connections) actually using the kernel mechanisms (namespaces, ,..) of your Ubuntu host system. It implemenrs that in a similar way as Docker does with the containers, but without

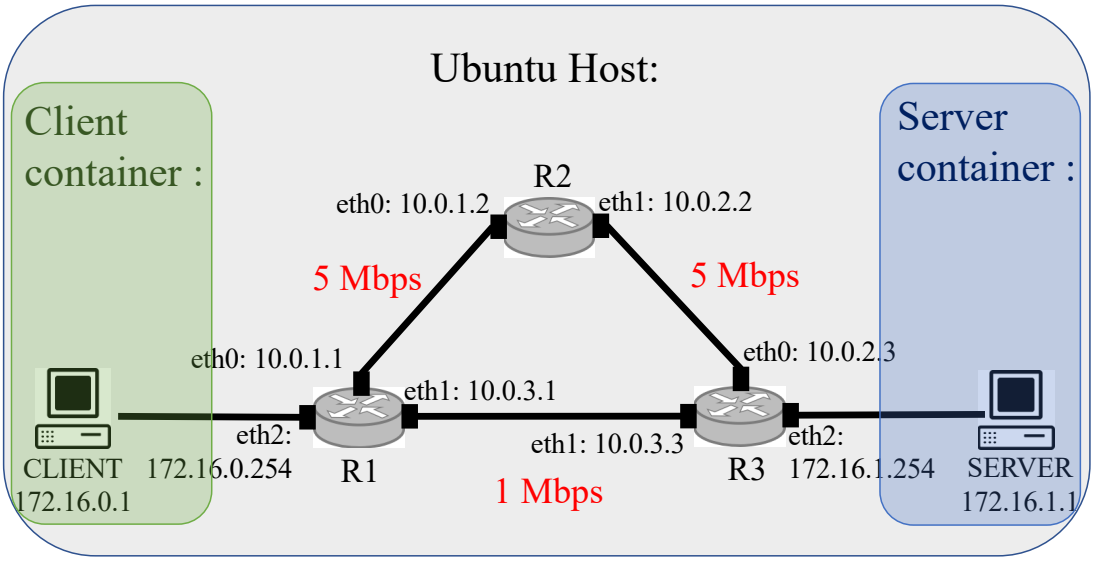

FIGURE 1 – Legacy network topology.

the same level of isolation with respect to the host. On the other hand, for the client and server will have two real Docker containers. The client and the server could have been *emulated* using the `Containernet` mechanisms, but we have preferred to implement them as actual containers to customize them more easily.

4.1 Setting up the network

In the `custom` folder, you have five elements (two files and three folders) :

1. the `topo_legacy.py` file, a *python file* defining the legacy network to set up in this Lab 2.
2. the `topo_sdn.py` file, a *python file* defining the SDN-based network to set up in the Lab 3.
3. the `server` folder, containing (i) the `dockerfile` to build the Dynamic Adaptive Streaming over HTTP (DASH) web server from the official `Ubuntu:jammy` distribution container, and (ii) the `vsftpd.conf` file, the config file of the FTP demon service.
4. the `client` folder, containing (i) the `dockerfile` to build the DASH client from the official `Ubuntu:jammy` distribution container, (ii) a `script_ftp.sh` shell file to launch a FTP download, and (iii) a `python` based emulated DASH video player in the compressed file `AS-stream-master.zip`. The video player corresponds to the github project `https://github.com/teaching-on-testbeds/AStream/tree/master`.
5. the `volumes` folder, that will be mounted in the server container to become a shared folder with your host operating system. This folder is initially empty. You need to download a compressed folder `media.zip` from the this url. This `zip` file contains the video frames that will be served. Once downloaded, unzip it and put it inside `volumes` to get the path `volumes/media`.

4.2 Setting up the `Containernet` network

In this subsection, we are going to deploy the `Containernet` network. `Containernet` allows defining our own networks by using an API in `Python`. `Containernet` documentation can be found in the GitHub wiki. The documentation for the underlying `Mininet` project can be found on the Mininet website. To get familiar with the `Mininet` network emulator, you should read these next two tutorials `http://mininet.org/walkthrough/` and `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`.

First, you should put the `/custom` folder inside the `Containernet` folder. The exact location of the `Containernet` folder depends on where you installed `Containernet`. If you follow my advice, it will be in your `/home/USERNAME` folder. Then, the full path to `/custom` will be `/home/YOUR_USERNAME/containernet/custom`.

Then, you need to build the images of the client container and server container. To do that you access to the corresponding folder (`client` or `client`), and you run :

```
USERNAME@HOSTNAME:~/containernet/custom/server$ docker build -t server-image .
USERNAME@HOSTNAME:~/containernet/custom/client$ docker build -t client-image .
```

Inside the `/custom` folder, as you know, you have the python code `topo_legacy.py`, thats implements the topology in Fig. 1.

Then, you can run this network by typing in the terminal :

```
USERNAME@HOSTNAME:~$  sudo -E env PATH=$PATH python3 custom/topo_legacy.py
```

or

```
USERNAME@HOSTNAME:~$  sudo python3 custom/topo__legacy.py
```

In the command prompt of your hosting system, the Containernet CLI will be displayed (`containernet>`). From this moment the Containernet emulator is running and you can launch networking commands from this CLI *as if you were in each network device* by using the Containernet (Mininet) commands (see `http://mininet.org/walkthrough/`). For example, if you want to test the connectivity from `r1` to `r2` using five ICMP echo messages (command `ping`), type :

```
containernet> r1 ping r2 -c 5
```

If your hosting system has `Xterm`, three `Xterms` windows have just opened. Each one emulates the terminal of each network router. If you go to the window named `host: r1`, you will be able to run commands as if you were in this router. For example, if you wish to ping `r2` from `r1` five times as before, type :

```
root@HOSTNAME:~# ping 10.0.1.2 -c 5
```

Finally, you need to access to the server container to activate the web and the FTP services since `containernet` ignores the `CMD` field in the `Dockerfile`. Then, the container will by default execute `/bin/bash`. To access to the server container.

```
$ docker exec -it mn.server /bin/bash
```

Once inside the server container, type :

```
root@server:/var/www/html# service apache2 start
root@server:/var/www/html# service vsftpd start
root@server:/var/www/html# service --status-all
```

The last command allows checking if the two services has correctly started up.

If the client and server containers are correctly connected to the `containernet` network a ping from the client to the sever should work. Access to client container first.

```
$ docker exec -it mn.client /bin/bash
```

And, once inside the client container, ping the server.

```
root@client:/home#  ping 172.16.1.1
```

Question 1 *: Check the the python code* `topo_legacy.py` *used to generate the network. Why do you think that the RTTs obtained in the pings are close to 20 ms ? Is this consistent with the routing tables in the devices ?*

5 Streaming an adaptive video

At this moment, we can play from the client, using the DASH protocol, the video hosted on the server. To play the video, we are going to use a client application written in `Python` called AStream (`https://github.com/teaching-on-testbeds/AStream/`). This `Python` program works (emulates) as a DASH player that generates the HTTP requests required to play the video and decides which quality file to download depending on an adaptation algorithm.

By-default, the client uses the adaptation algorithm called `basic`. This basic adaptation is a *rate-based policy* that chooses a video rate based on the observed download rate. It keeps track of the average download time for recent segments, and calculates a running average. If the download rate exceeds the current video rate by some threshold, it may increase the video rate. If the download rate is lower than the current video rate, it decreases the video rate to ensure smooth playback.You can see the source code for the basic policy here : `https://github.com/teaching-on-testbeds/AStream/blob/master/dist/client/adaptation/basic_dash2.py`

Now, you are going to make a first video play using this `basic` rate-based algorithm. As we did for the server in the previous section, you can enter in the client container by typing :

```
$ docker exec -it dash_client /bin/bash
```

You will land at the folder `/home`. From this folder, you type in the terminal :

```
python3 AStream-master/dist/client/dash_client.py -m
http://172.16.1.1/media/BigBuckBunny/4sec/BigBuckBunny_4s.mpd -p 'basic' -d
```

In the command prompt, you will see to appear the information about the playing. The temporal evolution of the playing will be saved as log files in the client directory `/home/ASTREAM_LOGS`. Play the video at least for several minutes. Once finished the play-out, you can copy this folder into the working directory of your host OS using the command :

```
docker cp <container_id>:/home/ASTREAM_LOGS/ .
```

To help with data visualization, you can use this Python notebook. You will need a Google account, since the Python notebook is hosted by Google Colab, and you should do a local copy into your Google drive. You can find also the Python notebook in the `lms`, But, you will need to install `jupyter notebook` from `https://jupyter.org/`. Follow the instructions to upload your log file, change the filename variable, and plot your results.

Question 2 *: Do as indicated above and plot with the Python notebook : the curves showing the video rate and the buffer status. Which is the actual end-to-end bandwidth seen by the client ? Is this consistent with the routing configuration ? What are the consequences over the video coding rate selected by the DASH adaptation algorithm ?*

6 Bandwidth sharing between video streaming and file transfer

In this section, we are going to test what happens when the HTTP-based streaming has to share the bandwidth with a TCP flow (as a FTP transfer). To do that, from the client, you play again the video using the `AStream` client, at the same time that you run the script called `script_ftp.sh` at the folder `/home` of the client container. This scripts starts downloading a file from the server using the FTP protocol after ca. 20s. Once the downloading is started, the file transfer is stopped after approx. one minute and a half. Plot the new curves using the Python notebook.

Question 3 *: How do the video quality and the buffered video time vary depending on the FTP transfer ? Does adaptive streaming allow attaining the maximal quality when the FTP transfer is active ? Why ? Hint : check this paper* **?**

7 Improving the video quality

We have seen in the previous section the limits of adaptive streaming when competing with a long-lived TCP flow (like FTP). Let's suppose that you are network engineering ordered to improve the video quality when there is no competition with the FTP transfer.

Question 4 *: How would you improve the video quality of the adaptive streaming in such a case ? Hint : as network engineer you cannot change the application protocols, that are standards, but you can introduce changes in your network configuration. Implement your solution and test it*

In case you need, you can find here information about how to set up static routes in Ubuntu machines. Note that the emulated routers in Containernet are actually Ubuntu hosts with several network interfaces and the IP forwading function activated.

Now, again, with your solution, repeat the experiment of running `script_ftp.sh` at the same time that you run the `AStream` client.

Question 5 *: What happens now with the video quality ? You are not anymore a networking engineer. You are completely free to propose a solution, even if it is not realistic with the current state of the art. Could you propose a better solution ?*