NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Who has this IP address? Simulating ARP protocol with sockets RAW
(TPC3 – Redes de Computadores)

1. Introduction

The goal of this assignment is to implement a client and a server in an ethernet network. The client's objective is to discover the physical location (ethernet address) of a server given its IP. This assignment is similar to the ARP protocol, which is used in local area networks for devices to find the location (ethernet address) of a device given its IP address. In this assignment it will be necessary to program with RAW sockets, build our own ethernet frames and process them, including the headers.

It is recommended to first check and follow the guide of the lab session 9 (Lab09-ethernetFrames.pdf), as it introduces much of the code and concepts necessary for this assignment.

2. Expected scenario/interaction

Assume multiple servers are running in a local area network, all of them continuously listening to incoming ethernet frames. A client starts its execution by broadcasting an ethernet frame containing the IP of the server it wants to contact. The client receives this IP as a parameter when it starts.

Since it is a broadcast, everyone (**including the client!**) will receive the ethernet frame. However, only the server with the requested IP should reply to the client. This server should send an ethernet frame directed to the client, stating that he is the one that has the requested IP address. The client receives this ethernet frame and extracts the server's ethernet address from the received frame's header.

As a **starting point**, use the source code of etherClient.py, etherServer.py and ethernetTools.py available from CLIP. Some of the steps necessary for this work are already provided, namely opening/closing the RAW sockets, creating/extracting frames and system calls to obtain (and show) the ethernet address and IP address.

The servers have to do the following steps:

1. Obtain both, the ethernet and IP address, associated to their network interface. This can be done with system calls, which are already provided in ethernetTools.py.
2. Create a RAW socket and associate the network interface to the socket.
3. Listen to incoming ethernet frames.
4. When an ethernet frame is received, analyze it. Check the requested IP in the frame's payload and compare it with its IP address.
5. If it doesn't match, ignore the frame and go back to step 3. If it does, create a new frame and send it to the client. The message to include in the payload is up to you (for example: "Hi, I am the server you are looking for"). Send the frame and go back to step 3.

The client has to do the following steps:

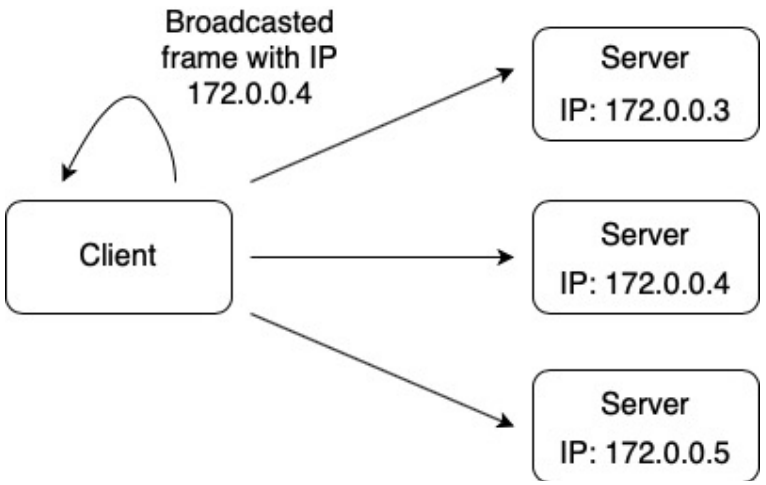
1. Obtain the ethernet address associated with its network interface.
2. Create a RAW socket and associate the network interface to the socket.
3. Create the ethernet frame to be broadcast. Remember to set the ethernet addresses correctly - namely, the target address should be the special broadcast one (`ff:ff:ff:ff:ff:ff`). Set as payload the IP address you are looking for.
4. Send the ethernet frame to the network.
5. Wait for a reply.
6. When a reply is received, analyze it. Check if it is the frame we're looking for - since the client's request is broadcasted, the client will also receive it and should, thus, ignore it and wait for the next frame.
7. When the right frame is received, obtain the server's ethernet address from the frame's header. Make sure to **print the server's ethernet address** and also the intended payload.

Figure 1 shows a diagram showcasing the expected interaction.

The client should receive the IP address that is being searched for as a parameter of the program. That is, if the client file is named client.py, it should be able to be run as follows:

```
python3 client.py IP_to_look_for
```

Note: You can assume the client will only ask for IPs that belong to one of the servers - you do not need to process the case in which none of the servers has the requested IP.



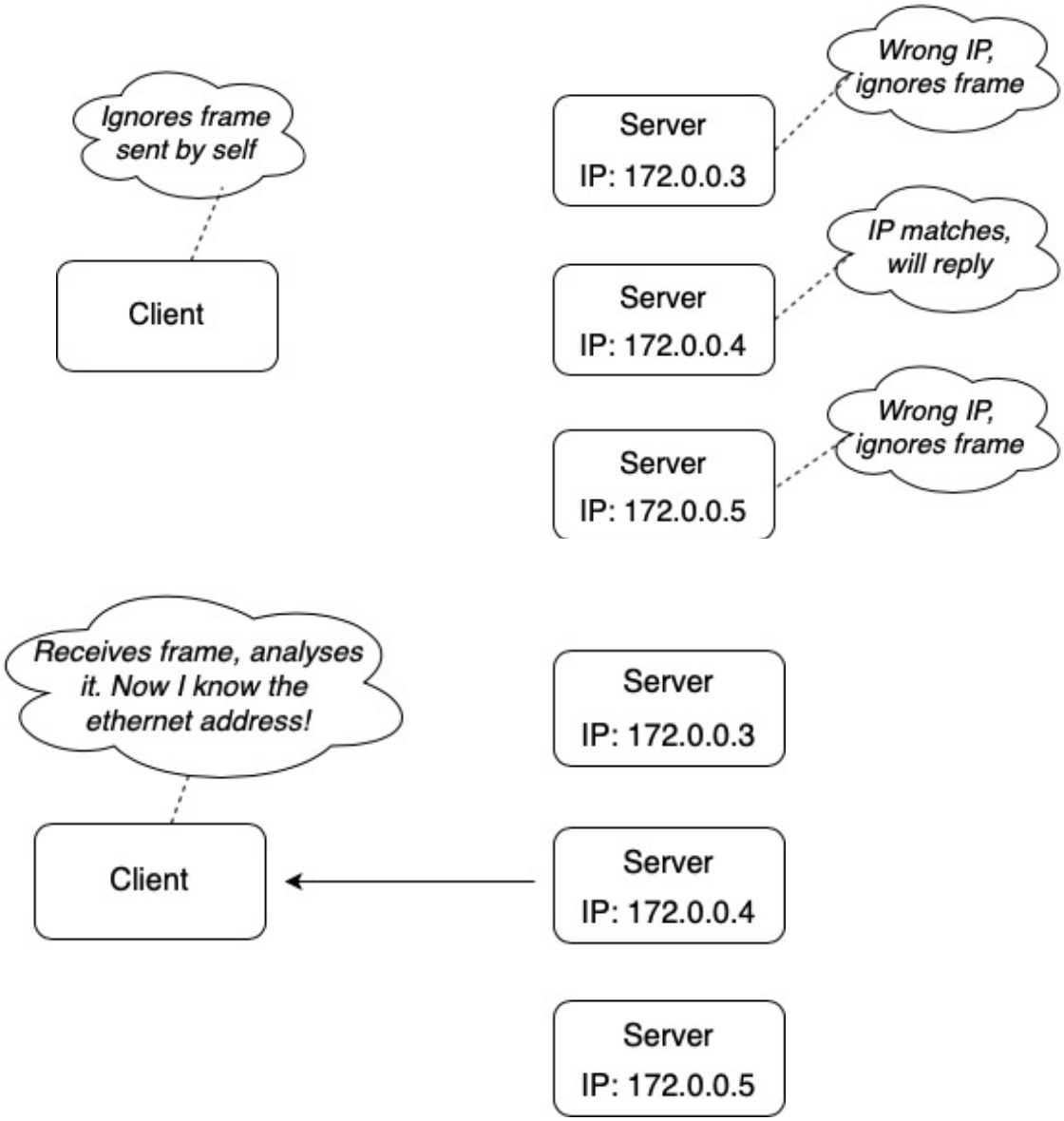


Figure 1 - Interaction between the client and the servers

3. Testing scenario

To ensure your solution works correctly, it should be tested using Docker, instantiating at least 3 servers and 1 client. Check the guide of the practical session 7 and 8 (Lab07.pdf and Lab08.pdf) as it contains the necessary Dockerfile and instructions on how to create an image and multiple containers.

To test it, start all the containers. Don't forget to start all servers before starting the client. On start each server should print its own ethernet and IP addresses. See if it was the right server that replied to the client and if the client was able to obtain the correct ethernet address, that is, the one that matches the requested IP address.

Try it with different IP addresses and observe how each time the server that replies is different.

Suggestion: use a different name for each container. This can be done with the option "`--name container_name`" when running the "docker run" command. With this, you can easily stop the containers (either with "exit" inside the container or "docker stop container_name" in another terminal) and restart them with "docker start container_name".

4. Delivery

The delivery will use a Google form. All the necessary files should be present in one folder, packed into a zip archive. Files to be delivered:

- Dockerfile that creates the image file;
- All the python source files necessary for the solution;
- A text file with the docker commands used to create the image and the containers;

More details on the delivery will be sent later. The deadline for delivery is Saturday 19th November 2022, at 11:59pm.