Middle East Technical University

Department of Computer Engineering

# CENG 435

Data Communications and Networking

Programming Assignment - Phase 1

Due Date: 5 November 2024 (Tuesday) - 23:59

## 1 Introduction

In this assignment, you are going to create a basic ICMP sender and receiver application in a container environment. You will use Python programming language with a specific version (Python 3.10.12). You do not have to create an environment for this assignment, instead, it was designed and shared with you as a docker environment. Additionally, you will use Git to clone the project that was created for the assignment.

You will work as a group of two students, and in the second phase you will continue with the same group, therefore; please fill the grouping assignment in ODTUClass before submitting the assignment.

With this assignment, you will gain introductory information for various important computer engineering skills. In your remaining work life, you will improve these skills, however; this programming assignment might be an introductory point for that.

In this assignment, you are going to

- fork the shared public repository to your own account and clone it to your local to work on it (only one of the team members should do it, the other member should be added as a collaborator.),
- create two containers using the docker-compose.yaml file which generates the images using the relevant Dockerfile,
- implement and run the ICMP packet sender and receiver using SCAPY,
- create a documentation using an automatic documentation system (Sphinx) (note that you will have to write the report and also document the source codes extensively using restructedtext and the Sphinx makefile is already provided.),
- upload the required files to ODTUClass.

**Important Note:** This is your first phase of the programming assignment, and it gives introductory information about the entire programming assignment. Therefore, it is important to complete these steps for the second phase, and some of the steps will also be similar to the second phase of the programming assignment.

## 2 Setup & Implementation

Each step of the assignment is explained one by one in the following.

## 2.1 Cloning GitHub Repository

First of all, you should open a GitHub account to fork the existing repository (covertovert). Forking is a copying process of an existing repository to a new repository that is in your own account. To fork an existing repository, click the **fork** on the upper right in the 'covertovert' repository and follow the instructions.

Afterward, you should open a new directory for the assignment on your local machine to work on it and clone the forked repository that is in your GitHub account. To clone the repository, you should click the **code** on the upper right, copy the command, and run the copied command in a terminal window which is in your newly created folder.

**Note:** According to the operating system you use, you may need to install Git if it is not automatically installed. To install, you can follow the instructions in the link.

## 2.2 Container Creation

This step was explained in the README.md file of the GitHub project. Please follow the instructions to create your container environment.

Docker containers are isolated environments that can be personalized for your own purposes. Therefore, they are very helpful tools for environment-dependent projects.

**Note**: In this assignment, it is important that you write your codes using specific versions of the tools in the environment. Therefore, you **must** use the shared container environment while developing.

## 2.3 Implementation of ICMP Sender and Receiver

This is the core part of the assignment. In this part, you should develop a Python code using scapy (using scapy is a **must**) where you should send an ICMP request whose TTL value is 1 from the 'sender' container to the 'receiver' container.

You should create the ICMP packet whose TTL value is 1 in the 'icmp_sender.py'. Afterward, you should send the created packet using the scapy library.

To receive the ICMP packet, you should also use the scapy library. In the 'icmp_receiver.py' file, you should capture the receiving ICMP packet whose TTL value is 1 and print it to the screen using the scapy's 'show' function.

**Important Note**: Operating systems create automatically ICMP response packet for each incoming ICMP request packets. When you send an ICMP packet from the 'sender' container, the 'receiver' container sends an ICMP response packet back. However, you should **not** print the ICMP response packet, instead; you should print only the incoming ICMP request packet in the 'receiver' container.

**Note**: As mentioned in the 'README.md' file in the GitHub repository, the 'code' folder in the host machine is mounted to the 'app' folder in both containers. These folders are the same, and any change that is made in any container or the host machine will directly affect the folder. Therefore, use this folder to avoid losing your codes, and be careful while editing the folder.

To run the system, first, you should run the 'icmp_receiver.py' to capture and then the 'icmp_sender.py' to send the ICMP packet.

## 2.4 Documentation Using Sphinx

Sphinx is a documentation generator for writing technical documentation. In this assignment, you will use it to create documentation of your codes automatically. To use it, you should generate configuration files and create them with Sphinx commands. However, in this assignment, it was provided for you. Therefore, you do not have to generate it again.

After you develop your codes, you need to change your directory to the '/app/docs' folder in one of the containers (**not** in the host machine because Sphinx was installed in the containers). Various files and folders exist for the Sphinx documentation in the 'docs' folder. It is enough to write the 'make html' command in a terminal window located in 'docs' folder, the required configuration for the documentation was provided to you.

After a successful run, you should check the '_build/html' folder, and you will see the automatically generated documentation of your codes.

**Note**: While Sphinx creates the documentation files for you automatically, it runs the 'icmp_receiver.py' and 'icmp_sender.py' files. Therefore, you should comment in the lines where it waits for the incoming packet.

**Note**: As you will see when you open the automatically generated documentation files ('index.html'), you have to change the 'index.rst' file by replacing the text with your names and group ID. After changing, you can safely rerun the 'make html' command.

In addition to the documentation created by Sphinx, a basic report for the assignment (.md file) should be created, that includes the following information:

- Name surname and student IDs of the group members,
- Group ID,
- GitHub repository link in your own account which was forked from 'covertovert'.

# 3 Submission & Grading

You should compress the following files and folders as <groupID>.tar.gz and upload it to ODTUClass (it is enough to upload by one of the group members):

- '_build' folder including documentation files and folders created by Sphinx
- 'ìcmp_sender' file
- 'ìcmp_receiver' file
- basic report (.md file)

The 'ìcmp_sender' and 'ìcmp_receiver' will be run and tested (black box). Therefore, when we run the 'ìcmp_receiver', it should wait for the incoming packet, and when we run the 'ìcmp_sender', it should send an ICMP packet whose TTL is 1. The testing process will include two steps, black-box testing and white-box testing. If your code does not pass the black-box test, you will get a 0 from the coding part. However, if your code is passed, a white-box test will also be applied to check whether you have followed the restrictions.

The grading of the assignment will be conducted using the rules shown below.

- successfully created Sphinx documentation (20 pts)
- ICMP sender and receiver that passes black-box and white-box testing (60 pts)
- completed report (.md file) (20 pts)