Computer Networks
Skill Check

Prof. Dr. rer. nat. Nils Aschenbruck

Leonhard Brüggemann, Leon Richardt, Alexander Tessmer

General information:

- Read the **complete** task sheet **carefully** before you start working on it!
- By starting to work on the task, you declare that you are able to do so both physically and mentally.
- You must only use your assigned computer and work in the directory *Task* on the desktop. This is vital for submission (see below)!
- In case of an attempt to deceive, the assignment of all parties involved will be graded immediately as failed. There is no prior warning. Mobile phones that are switched on and other communication devices (such as smart watches) are also considered an attempt at deception! Changing the font size is also considered an attempt at deception.
- You are not allowed to eat in the digital exam labs, drinking is allowed.
- If you have to go to the bathroom, notify the supervisor. They will make sure that no two people go at the same time.
- Submission:
  - Make sure that your final version is saved in the directory *Task*.
  - Notify the supervisor to present your solution and **do not** turn off your computer.
  - The solution is automatically copied from your computer.
- System Information:
  - The tasks assume you are working on **Ubuntu 22.04**, using **Python 3.10**. We cannot guarantee support for other system environments.
  - Files that are part of this task are provided in the directory *Task* on the Desktop.
- Provided Documentation:
  - Linux man pages
  - Python documentation (might take a few seconds to open)
  - OpenSSL Cookbook
- Recommended IDEs: VS Code, vim, GNU nano, gedit

Please note: In order to pass, your solution must fulfill **all requirements** specified in the task. It is not sufficient to only complete a subset.

Task 2-A: UDP Weather Station

In a small mountain town, a cutting-edge weather monitoring system was deployed. This system uses five autonomous weather sensors (UDP clients) scattered across the region to measure temperature, humidity, and wind speed. These sensors were designed to send their data every second to the central weather station (a UDP server). Due to harsh weather conditions, the link is occasionally lossy, which means that sensor data might get lost. The weather station needs to handle the weather sensors, ensuring no data losses. Your task is to develop such a robust UDP server using a server structure of your choice.

1 Requirements

At the center of this assignment are the five UDP clients. Each UDP client has a unique identifier (*uid*) which is an integer. Every second, each UDP client generates sensor data which is recorded alongside a sequence number. The client puts it into a datagram and sends the data every second to the server. However, the reliability of these transmissions is compromised by occasional data loss. The UDP server ensures that no data is lost, by checking the sequence numbers for missing entries. If one or multiple continuous entries are missing, the server sends retransmission requests to the corresponding client immediately. **You may assume that when packet loss occurs, the server's requests and clients' retransmissions are always transferred successfully. Furthermore, you may assume that the first datagram of each client is always transferred successfully.**

Use the template `weather_station.py` containing some definitions you might use. It is run by:

```
student@host:~/task-a$ python3 weather_station.py [port]
```

with an optional `port` parameter. The server binds to `127.0.0.1:`*port*, i.e., to the specified port on the local host. When the `port` parameter is omitted, the server binds to port `4711` per default.

In order to pass this task, run below `weather_clients` with five connections. The output on the client part should look similar to Example 1.

2 Client Description

To help you verify your implementation, an executable `weather_clients` is provided to you in the task folder that simulates the sensors. The client is started by running

```
student@host:~/task-a$ ./weather_clients [--port <port>] \
                       [--connections <connections>]]
```

Example 1

In a correct implementation, this could be the server-side output of a session:

```
student@host:~/task-a$ python3 weather_server.py
Averages for client 0: -9.2°C | 27.5% | 33.5km/h
Averages for client 1: 10.9°C | 14.4% | 69.4km/h
Averages for client 2: 6.3°C | 86.2% | 26.5km/h
Averages for client 3: 4.6°C | 8.2% | 50.8km/h
Averages for client 4: 8.5°C | 24.2% | 16.5km/h
Averages for client 0: 1.4°C | 47.6% | 83.7km/h
Averages for client 1: 10.5°C | 46.8% | 68.7km/h
Averages for client 3: -1.6°C | 4.6% | 46.7km/h
...
```

with the optional parameters `port`, and `connections`. `weather_clients` starts as many clients as specified by `connections`, and each connects to `127.0.0.1:`*port*, i.e., to the specified port on the local host.

3 Protocol Format

The actual exchange of messages proceeds as follows. You may assume that when packet loss occurs, the server's requests and clients' retransmissions are always transferred successfully. Furthermore, you may assume that the first datagram of each client is always transferred successfully.

1. Every second, each client sends a message to the `weather_station`, which contains
   - a unique client ID (UID) as *unsigned short*,
   - a sequence number as *unsigned integer*,
   - a temperature measurement as *float*,
   - a humidity measurement as *float*, and
   - a wind speed measurement as *float*.

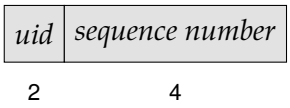
   The message format is as follows:

uid	*sequence number*	*temperature*	*humidity*	*wind speed*
2	4	4	4	4

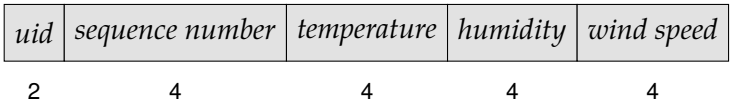
2. The `weather_station` saves the data in a data structure of your choice and checks for packet loss. Every time it receives a message, it calculates the arithmetic mean and prints the data to the console as visualized in Example 1.
3. If the `weather_station` detects a missing sequence number for a client, it sends a message to the corresponding client with the following format:

- the UID of the client as *unsigned short*, and
- the sequence number of the missing packet as *unsigned integer*.

The message format is as follows:



4. The contacted client transfers the missing data immediately, in the same format as in Step 1:



Good luck!