Lecturer
Prof. Dr. Christian Tschudin (christian.tschudin@unibas.ch)

Tutors
Carlos Andrés Tejera (carlos.tejera@stud.unibas.ch)
Andreas Wassmer (andreas.wassmer@unibas.ch)

Uploaded on
Thu., 31 March 2022

Deadline
Wed., 13 April 2022
Upload on ADAM

Modalities

For this exercise sheet, we expect you to hand-in written solutions as PDF and a partial presentation in a online meeting. Besides explanations and answers to questions, the PDF should contain the detailed documented source code and for tasks not to be presented (noted in the question) a list of all executed commands and generated outputs (as screenshots if relevant).
The solutions can be handed-in in groups of two. The presentation will take place in the week of the deadline.
Programming implementations can be executed as **C++** or **Python3**, as you prefer. The handed-in source code has to be well documented containing :

- the description of the idea and realisation
- methods/function call description, i.e. the target and call arguments
- points of complex / difficult implementation
- emphasis points as asked by the task description
- boundary conditions , limitations and in-/valid values
- makefile or instructions for compilation via README file

The OS target system for code implementations shall be a Linux distribution. Please, implement the code in library/module style (called for a second file with main() for C++ or by Python3 package standards[1]).

[1] `https://packaging.python.org/tutorials/packaging-projects`

Task 1 - TCP Chat (6 Punkte)

The goal of this exercise is to implement and observe the traffic of an application with simple message exchange. Hence, you will implement a simple chat application.
In this first step, all communication shall be done via **TCP**. Furthermore, the following requirements have to be met :

- Two peers can **communicate directly** with each other (i.e. without an intermediate server, **peer-to-peer** respectively).
  Hint: With server we mean a central process that manages and sometimes forwards messages. A listener-"server" and a requesting "client" is still required to establish communication.
- The **IP** address and **port** of the communication partner are passed **as command line arguments**.
- Chat messages are plain UTF-8 strings (default strings).
- For chat-message input (and output), use CLI (standard IO)
- **Fast** message display (own and received)
  To display messages immediately, use `select()` for C[2], for Python[3] and for C++[4]. This call can be used **with sockets and CLI** In-/Output.
  Do NOT use multiprocessing, asyncio or something alike. Such methods remove any guaranty of ordered input/output (order preservation).

Task 2 - UDP Chat (6 Punkte)

Here, you implement an UDP variant of the previous chat application with the following differences:

- Two establish communication, all peers have to connect to a **"central" server**. Herefore, a unique nickname has to be delivered by the peer.
- **Peers communicate directly** with one another (P2P) NOT via server (PubSub).
- For to establish a chat connection with another peer (which is connected to the server), the **peer is chosen by its nickname**. The server then presents the IP and port to the "requester", i.e. peer establishing the connection. **All peers only know the server-IP and port** (outside a chat connection).

[2] http://man7.org/linux/man-pages/man2/select.2.html
[3] https://docs.python.org/3/library/select.html
[4] https://www.cplusplus.com/reference/streambuf/streambuf/in_avail/

Task 3 - Extend Your Implementation (6 Points)

Now, you are to extend the UDP version of the chat application to use chat rooms, i.e. communicate with a group (of arbitrarily many) people. Hence, the following criteria have to be fulfilled :

- **Server manages rooms** and presents it to clients/peers (i.e. chat room list additional to client-nickname list). Hence, chat rooms can be picked like nicknames by peers. Use **UDP unicast** to publish list(s).
- Clients/Peers can create rooms and join them.
- Peers in a chat room connect and communicate by UDP multicast directly with one another ("Peer-to-Peers", no server involved)

a) Implement

b) Explain : How to provide a list of participants (nicknames) to all peers in a chat-room? (Still without involving the server.) On what level of the network model (i.e. "internet protocol stack") would you implement it and why?

Task 4 - Network Analysis (6 Points)

In this exercise, you will capture and analyse the network traffic of your VMs running the previously implemented applications. To achieve this, you can use Wireshark, which you installed in the previous exercise, allowing you to observe the paket traffic without interupting it, i.e. so called paket sniffing. Wireshark does this by directly accessing and grabbing pakets from the network interfaces (e.g. ethernet controller).
To present your result and allow extended investigations, we recommand recording the traffic for analysis.

a) Filter the collected traffic information by applications the previous applications implemented in this exercise sheet and comment/group them by type/purpose. (1 Point)

b) Implement and Observe the network traffic of a TCP and an UDP (both Unicast and Multicast) communication.

- Mark the maintenance parts (initialising, reconnecting or else) of these network traffics and assign packets to all steps of handshake, initialisation, recovery and alike. (2 Points)
- For the non maintenance part, mark down to which layer of the internet protocol stack each packet belongs. (2 Points)
- Of all these different packets, what is the common theme and what are the differences (besides the content)? (1 Point)