Computer Networks
Skill Check

Prof. Dr. rer. nat. Nils Aschenbruck

Leonhard Brüggemann, Bennet Janzen, 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 1-A: `simple-ping`

In this task, you will implement a simplified version of the well-known `ping` command-line tool, which we call `simple-ping`. A `simple-ping` server responds to PING messages with PONG messages. The time between sending a PING and receiving the respective PONG can be used to measure the roundtrip time (RTT) between client and server. Conveniently, the implementation of a `simple-ping` server is provided already. Your task is to implement a `simple-ping` client in Python.

1 Requirements

Your task is to implement a `simple-ping` client in the `client.py` file, using Python 3. Your implementation must do the following:

1. Send well-formed PING messages to the `simple-ping` server via UDP (cf. Section 3).
2. For up to **3 seconds**, wait for a PONG response to the message. When no matching response arrives within that time frame, the message must be considered **lost**.
3. When a PONG message arrives, record the RTT for the corresponding sequence number and print it to the terminal. For a lost PONG (or PING) message, print to the terminal that it got lost.
4. After you have collected **11 RTT samples**, close the socket. Then, print the **median** of collected samples to the terminal.

Example 1 illustrates one possible correct way to output the required information. Your solution may use a different output format, as long as all specified information is present.

You have passed the task when the RTT statistics are correctly printed in a reproducible manner.

2 Server Description

To help you verify your implementation, a `simple-ping` server is provided to you in the task folder. The server is started by running

```
student@host:~/task-a$ ./server [port]
```

with an optional `port` parameter. The server starts and 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. The server then waits for incoming messages. When it receives a message, the following steps are performed:

1. Check whether the message payload starts with the word `PING`, encoded in UTF-8. If it does not, ignore the message. If it does, continue with the next step.
2. Parse the sequence number s_{cur} from the message (cf. Section 3).

3. Transmit a message with the UTF-8-encoded prefix `PONG` and s_{cur} to the sender address.

To assist you in developing your implementation, the server logs messages to the terminal whenever a notable event occurs.

Example 1

In a correct implementation, this could be output for a `simple-ping` session:

```
student@host:~/task-a$ python client.py
Packet  0: rtt = 147.58 ms
Packet  1: timed out
Packet  2: rtt = 970.41 ms
Packet  3: rtt = 136.64 ms
Packet  4: rtt = 795.47 ms
Packet  5: rtt = 976.31 ms
Packet  6: rtt = 830.18 ms
Packet  7: rtt = 649.90 ms
Packet  8: timed out
Packet  9: rtt = 399.94 ms
Packet 10: rtt = 560.18 ms
Packet 11: timed out
Packet 12: rtt = 706.32 ms
Packet 13: rtt = 224.61 ms
Median RTT (over 11 samples): 706.32 ms
```

3 Protocol Format

The `simple-ping` protocol defines two message types: PING and PONG messages. By sending a PING message, a client requests a server to respond back with a PONG message. PONG messages must only be sent in response to PING messages, never on their own. `simple-ping` messages must be transported over UDP.

The format of `simple-ping` messages is described in Figure 1. Each PING and PONG message contains a 4-byte integer sequence number. No two PING messages with identical sequence numbers may be sent over the course of a `simple-ping` session, i.e., the sequence number must strictly increase. When a `simple-ping` server receives a PING message with sequence number s, it transmits back a PONG message with s to the sender address. This serves to identify request-response pairs since multiple messages may be in flight at the same time.

(a) PING message (b) PONG message

Figure 1: Format of `simple-ping` protocol messages. The numbers below the fields indicate the field length in number of bytes.

4 Follow-Up

Try to answer the following questions:

1. Would you recommend the use of TCP over UDP for the `simple-ping` protocol? Why or why not?
2. Describe a method to estimate the packet loss on the path between client and server.
3. Is the `simple-ping` protocol equipped to handle parallel communication of multiple clients with a single server? Why or why not?
   If not: How would the protocol need to be adapted to support this scenario?

Note

For transparency: You will not have to answer these questions during the Skill Check. Nonetheless, we encourage you to think about how you would answer them. See them as an opportunity to improve your understanding of the relevant course material. Questions similar to these might be part of the final exam.

Good luck!