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 2-D: `udp-uploader-server with LRC`

In Task 1-D, you programmed an udp-uploader-client with reliable communication. In this task, you will implement a server that reliably handles the upload of a file from a provided UDP client. While the focus was on the ARQ in Task 1-D, this time the focus will be on the integrity check. The server writes received data to disk in order and sends acknowledgments for each received packet. Each packet will be checked with parities and LRC and an error acknowledgment will be send, if the integrity check failed. Your job is to implement a server in Python that copes with these impairments and completes the upload without corruption correctly.

1 Requirements

Implement the `udp-uploader-server` in Python 3 in a file named `server.py`. Your implementation must:

1. Listen on `127.0.0.1:`*`port`* (default `4711`).
2. Create a new output file `received.txt`.
3. Start a session upon receiving the first `D` from a client and will only accept packets from that same client for the session duration.
4. On receiving a `D` packet:
   - If the parity bits of each character and the LRC are correct, the payload is written to the file, and an acknowledgment packet `A` (ACK) is send.
   - Otherwise, do not write and send an error packet `E` (Error) instead.
5. On receiving a the final `F` (Finalize) packet, close the file and reset the session state.
6. Always print concise status information to the terminal. Include the received message chunk and if the integrity check was correct. An example output is shown in Example 1.

You can assume, that there are never errors in the identifier or the length and that each package arrives exactly one time.

You have passed the task when your server completes the upload and the uploaded file matches the input file for the client with question marks for non-ASCII characters. Do not edit `test.txt`!

Example 1

In a correct implementation, this could be the output for the `udp-uploader-server`:

```
student@host:~/task-d$ python3 server.py
Server started!
Message: -------------------- | LRC Error: False
Message: -------\nLorem ipsum  | LRC Error: False
...
Message:  labore et dolore ma | LRC Error: False
Message: gna aliqua. Ut enim  | LRC Error: True
...
Message: gna aliqua. Ut enim  | LRC Error: True
Message: gna aliqua. Ut enim  | LRC Error: False
Message: ad minim veniam.\n?????? | LRC Error: False
```

2 Client Description

To start the client, run

```
student@host:~/task-c$ ./client
        --src_ip <client_ip> --src_port <client_port>
        --dest_ip <server_ip> --dest_port <server_port>
        --filepath <data_file>
        --max_payload <chunk_size>
        --debug_mode <'no','random','test'>
```

with all parameters being optional. The client binds to `src_ip:src_port`, which is `127.0.0.1:4710` by default, and then attempts to send the characters from the file found at `filepath`, which is `lorem.txt` by default, to the server address `dest_ip:dest_port`, which is `127.0.0.1:4711` by default. The characters from the file are send in chunks with a maximum size defined with `max_payload`, which is `20` by default. All characters are encoded with 7-bit ASCII and an added eigth most-significant parity bit. All non-ASCII character bytes are replaced with question marks. Each packet starts with the identifier 'D' and the number of characters in the packet chunk as short. Consecutive is the characters chunk with an added LRC byte. After each sent data packet, the client waits for an acknowledgement to continue or an error message to resend the chunk. The debug mode is for testing and introduces bit errors into the data. There are no errors with 'no'. 'test' has 4 different error cases, that are induced on packets with index 7, 9, 11, and 13. 'random' chooses a random error case or no error for each packet. After all data chunks have been acknowledged, the client sends a single-byte F (Finalize) packet and closes the socket. All packet formats are presented in Section 4. The current state of sending chunks is printed. An example output is shown in Example 2.

Example 2

This is an example output for the `udp-uploader-client`:

```
student@host:~/task-d$ python3 client.py lorem.txt
Client started!
Chunk 0 [ACK] (attempt 1)
Chunk 1 [ACK] (attempt 1)
...
Chunk 6 [ACK] (attempt 1)
Chunk 7 [ERR] (attempt 1/10)
...
Chunk 7 [ERR] (attempt 8/10)
Chunk 7 [ACK] (attempt 9)
Chunk 8 [ACK] (attempt 1)
Finalized: Packets sent in 13.0 ms.
```

3 Longitudinal Redundancy Check (LRC)

The Longitudinal Redundancy Check (LRC) is also known as 2-D parity check. In this method, data which the user wants to send is organised into a matrix of rows and columns. In order to detect an error, a redundant bit is added to each row and a whole row is added to the whole block and this block is transmitted to receiver. The receiver uses these redundant bits to detect an error. In this task, we use even parity and each row consists of a single 7-bit ASCII character, which sums to 8-bit with an added most-significant even parity bit, so that the total number of ’1’s in the 8-bits are always an even number. The added LRC row is calculated by the even parity of the columns of the data block. It should be noted, that while the rows are always of length 8, the length of the columns can vary and depends on the numbers of characters send. An example of the LRC check with a burst error is shown in Figure 1.

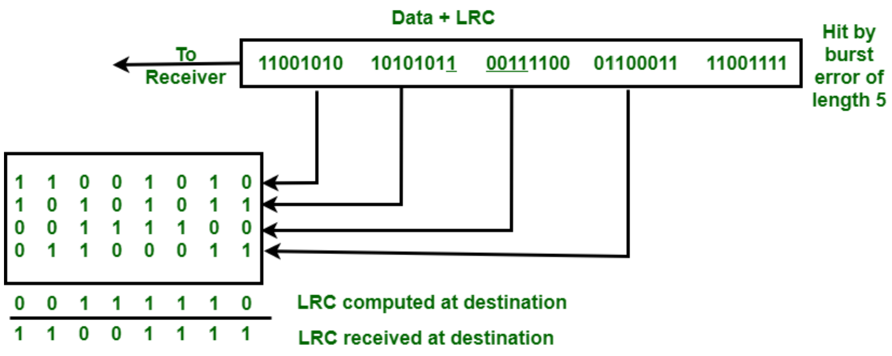
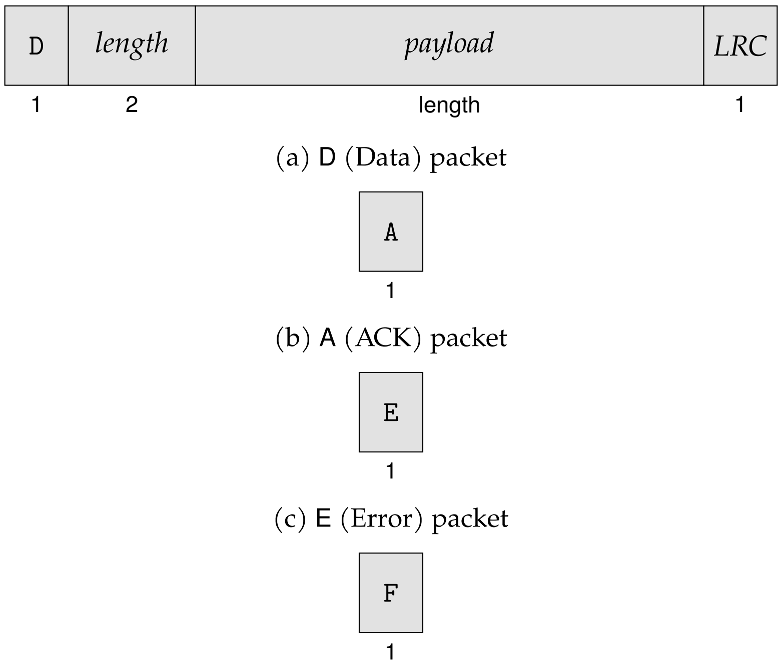


Figure 1: LRC check visualization with burst error.

4 Protocol Format

All packets are transported over UDP. Multi-byte integers are encoded in **big-endian**.



(a) D (Data) packet

(b) A (ACK) packet

(c) E (Error) packet

(d) F (Finalize) packet

Figure 2: Message formats for the protocol.

5 Follow-Up

Try to answer the following questions:

1. If there is no LRC error, is the uploaded file in this setting always correct? Why not?
2. Comparing LRC to CRC (Cyclic Redundancy Check). What improvements does CRC provide and where does it fail?

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!