Introduction to Computer Networks

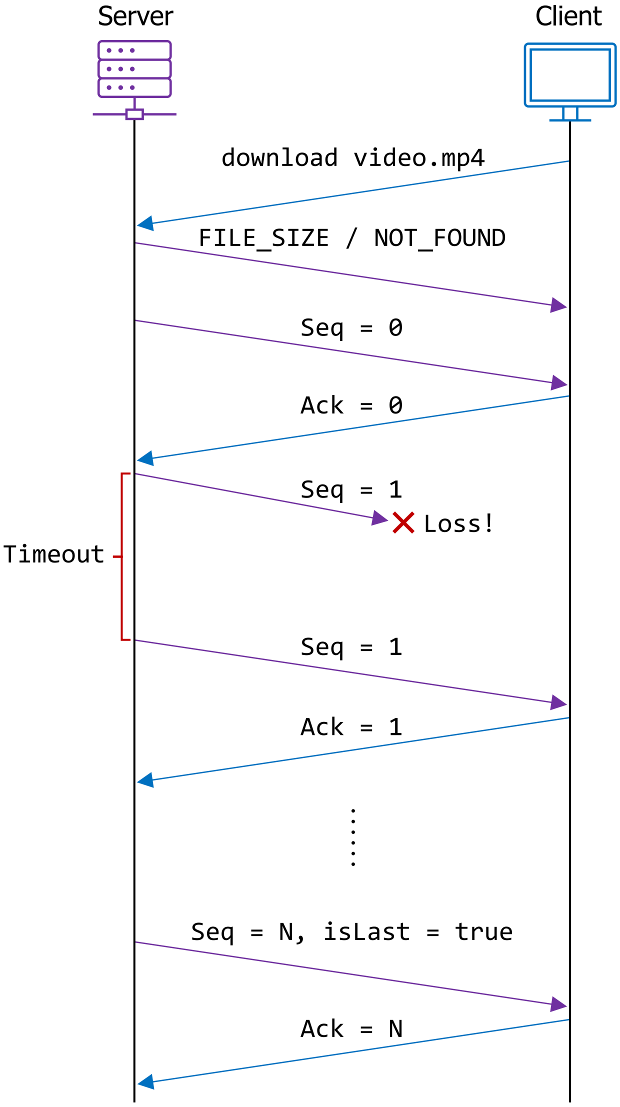
Lab 3: Linux Socket Programming II

1. Description

Implement the stop-and-wait mechanism by Linux socketing programming at both the client side and the server side. The client should be able to download a file from the server using this mechanism over a UDP socket. The main objective is to have hands-on experience in implementing Automatic Repeat-request (ARQ).

Extra Bonus: You can earn an additional bonus by implementing the selective repeat mechanism in addition to the stop-and-wait mechanism.

2. Requirements: Stop-and-Wait (100%)



(1) Server

- Create a UDP socket on port `7777`. The server awaits requests from clients by listening to the port.
- When the server receives a download request, if the file exists, the server should respond with the file size as "`FILE_SIZE={size}`"; otherwise, it should reply with "`NOT_FOUND`".
- If the specified file exists, the server transmits that file to the client by the **stop-and-wait** mechanism.
- The payload size of each packet is fixed at 1024 bytes, including the last packet. The actual data of the last packet is `{size}` *mod* 1024 bytes, but padding (all zeros) should be added to the last packet such that the length of the last packet is equal to 1024 bytes.
- In case an acknowledgment (ACK) is not received within 100 milliseconds, the missing packet should be retransmitted (`#define TIMEOUT 100`).

(2) Client

- Create a UDP socket and specify the server's IP address and port.
- Allow the user to use the command "`download {filename}`" to request a file from the server.
- After receiving confirmation of the file's existence (`FILE_SIZE={size}`) in the response, the client can proceed to start receiving the file.
- To simulate packet loss, the client intentionally disregards each received packet with a 30% probability (`#define LOSS_RATE 0.3`).
- Upon successfully receiving a packet, the client should send an acknowledgment (ACK) and append the received data to the file if the associated sequence number is valid.
- Save the file with the prefix "`download_`" and name it as "`download_{filename}`".

(3) Build

- This project should include a Makefile, and the build process should be performed using the "make" command. Compile to produce two executable files named "client" and "server".

3. Extra Bonus: Selective Repeat (+20%)

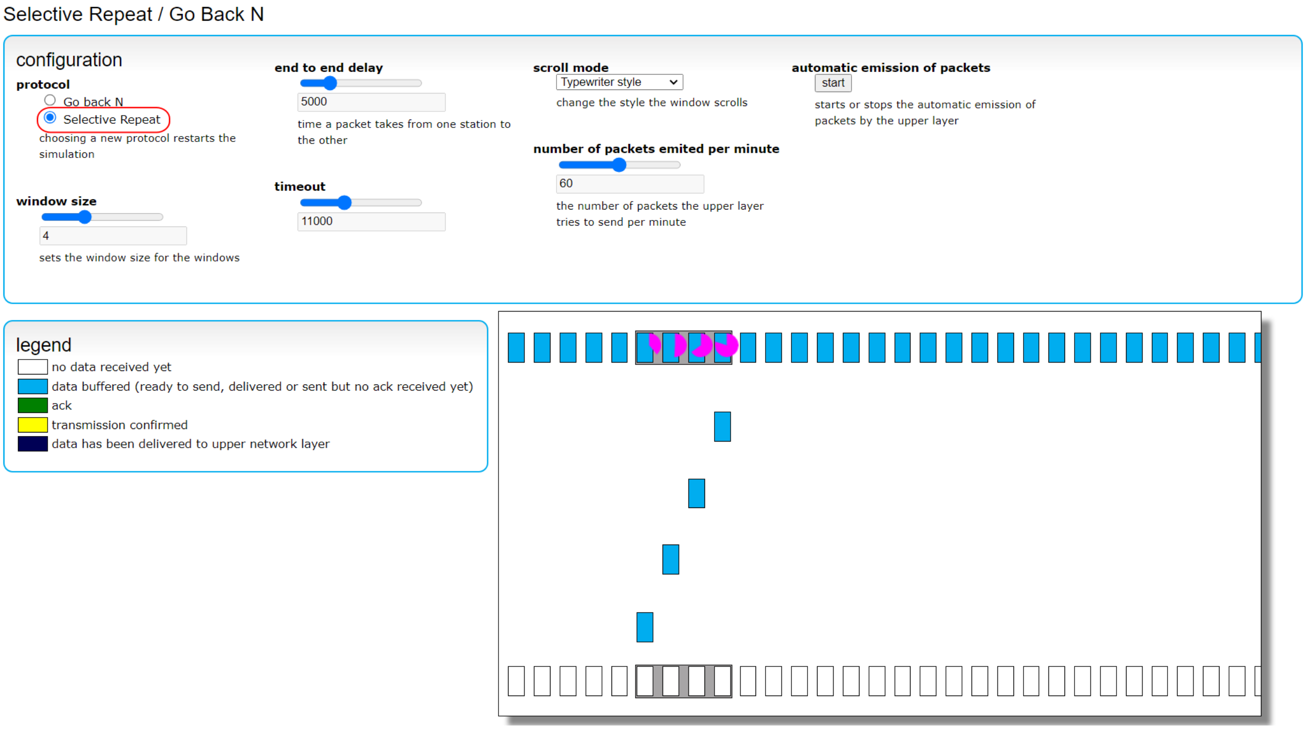
To implement the selective repeat mechanism, begin by defining the window size (e.g., `#define WINDOW_SIZE 4`). Then, in the context of this mechanism, effective synchronization between threads becomes crucial. You can use mutex to organize critical sections, ensuring that only one critical section is executed at any given time. For instance, you can establish critical sections for sending packets, monitoring ACK timeout, and handling ACK packets, respectively.

The challenging aspect lies in implementing the movement of a sliding window and managing multiple ACK timers. In cases where ACKs are not received in a timely manner, retransmission is required only for the packets corresponding to specific sequence numbers.

The original range of the sequence number would be constrained by space limitations and cyclically reused, such as in the case of 2 bits: 0, 1, 2, 3, 0, 1... For ease of implementation, you don't need to consider this. Simply use 0, 1, 2, 3, 4, 5... directly.

You can view the visualization of selective repeat on the website below. In the top-left corner, select the protocol as [Selective Repeat], and adjust parameters such as the window size as needed. Click [start] to run the simulation. During the packet transmission process, you can click on packets to simulate packet loss, allowing you to observe the mechanism of selective repeat.

- https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/



Selective Repeat / Go Back N

configuration

protocol

○ Go back N
● Selective Repeat

choosing a new protocol restarts the simulation

window size

4

sets the window size for the windows

end to end delay

5000

time a packet takes from one station to the other

timeout

11000

scroll mode

Typewriter style

change the style the window scrolls

number of packets emited per minute

60

the number of packets the upper layer tries to send per minute

automatic emission of packets

start

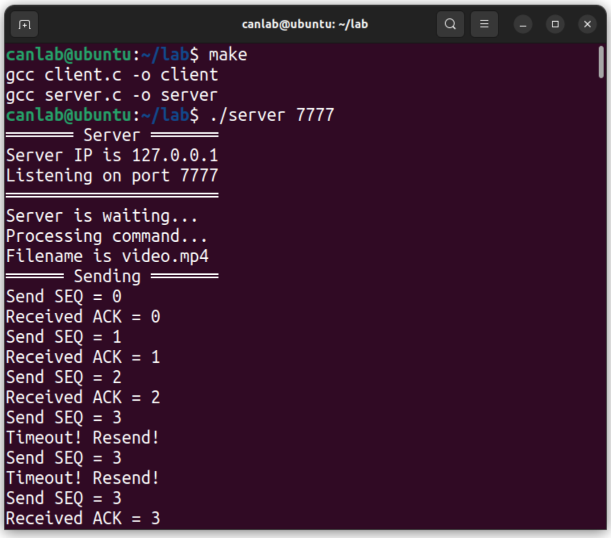
starts or stops the automatic emission of packets by the upper layer

legend

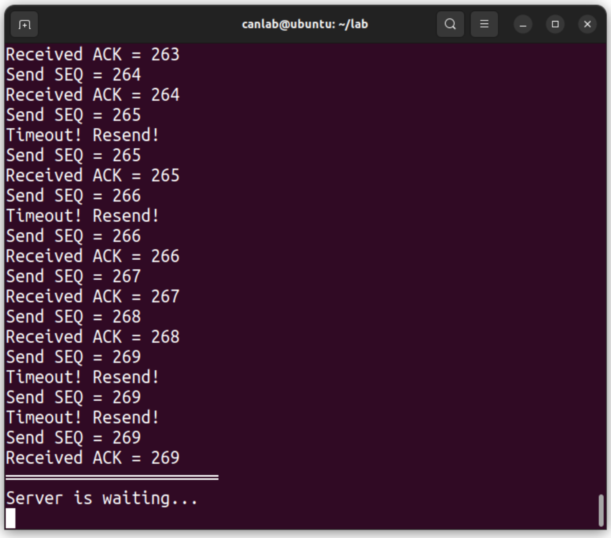
□ no data received yet
🟦 data buffered (ready to send, delivered or sent but no ack received yet)
🟩 ack
🟨 transmission confirmed
■ data has been delivered to upper network layer

4. Example

Server

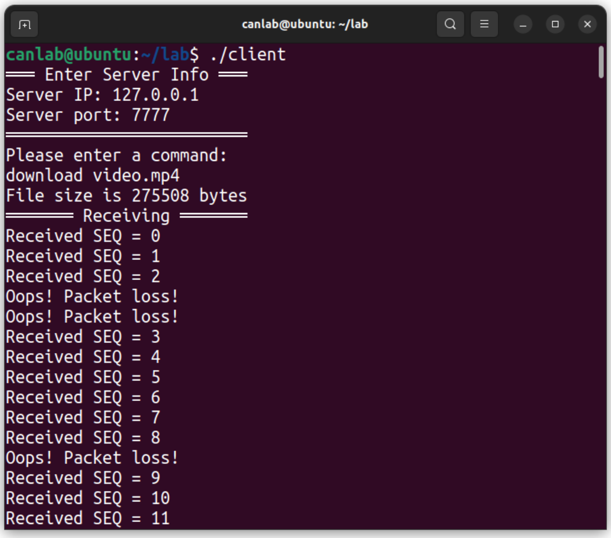


```
canlab@ubuntu:~/lab$ make
gcc client.c -o client
gcc server.c -o server
canlab@ubuntu:~/lab$ ./server 7777
══════ Server ══════
Server IP is 127.0.0.1
Listening on port 7777
══════════════════════
Server is waiting...
Processing command...
Filename is video.mp4
═════ Sending ═════
Send SEQ = 0
Received ACK = 0
Send SEQ = 1
Received ACK = 1
Send SEQ = 2
Received ACK = 2
Send SEQ = 3
Timeout! Resend!
Send SEQ = 3
Timeout! Resend!
Send SEQ = 3
Received ACK = 3
```

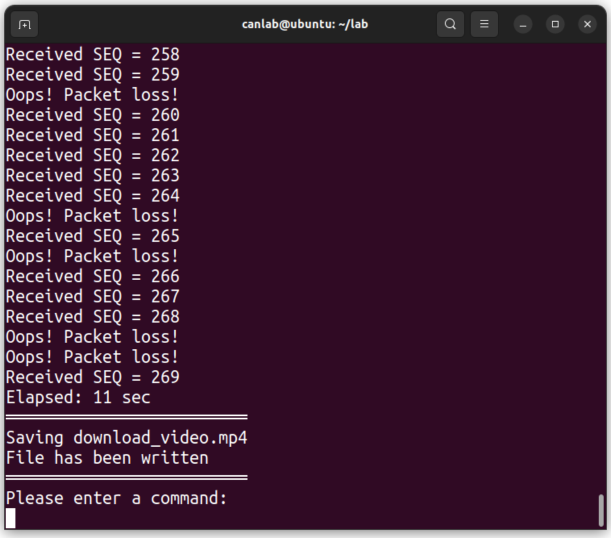


```
Received ACK = 263
Send SEQ = 264
Received ACK = 264
Send SEQ = 265
Timeout! Resend!
Send SEQ = 265
Received ACK = 265
Send SEQ = 266
Timeout! Resend!
Send SEQ = 266
Received ACK = 266
Send SEQ = 267
Received ACK = 267
Send SEQ = 268
Received ACK = 268
Send SEQ = 269
Timeout! Resend!
Send SEQ = 269
Timeout! Resend!
Send SEQ = 269
Received ACK = 269
══════════════════════
Server is waiting...
```

client



```
canlab@ubuntu:~/lab$ ./client
══ Enter Server Info ══
Server IP: 127.0.0.1
Server port: 7777
═════════════════════════
Please enter a command:
download video.mp4
File size is 275508 bytes
═══════ Receiving ═══════
Received SEQ = 0
Received SEQ = 1
Received SEQ = 2
Oops! Packet loss!
Oops! Packet loss!
Received SEQ = 3
Received SEQ = 4
Received SEQ = 5
Received SEQ = 6
Received SEQ = 7
Received SEQ = 8
Oops! Packet loss!
Received SEQ = 9
Received SEQ = 10
Received SEQ = 11
```



```
Received SEQ = 258
Received SEQ = 259
Oops! Packet loss!
Received SEQ = 260
Received SEQ = 261
Received SEQ = 262
Received SEQ = 263
Received SEQ = 264
Oops! Packet loss!
Received SEQ = 265
Oops! Packet loss!
Received SEQ = 266
Received SEQ = 267
Received SEQ = 268
Oops! Packet loss!
Oops! Packet loss!
Received SEQ = 269
Elapsed: 11 sec
═════════════════════════
Saving download_video.mp4
File has been written
═════════════════════════
Please enter a command:
```

5. Code Template

To facilitate your implementation, we have provided basic code templates along with the Makefile. You can gradually follow the hints to complete this project based on the template. Feel free to modify the code style; you are not restricted to the template. However, if you choose to write the code from scratch by yourself, be sure to confirm that you meet the requirements we have specified.

6. Make and Makefile

“Make” is a build automation tool that manages the compilation and linking of source code into executable programs. It uses a set of rules defined in a Makefile to determine how to build the software. The Makefile contains dependencies and commands for each target, ensuring that only necessary components are recompiled, saving time and resources.

Here’s a straightforward example of a Makefile. It’s worth noting that in practical applications, Makefiles are often generated with tools like CMake, which makes them more structured and portable.

Makefile

```makefile
all: client server

client: lab.h client.c
	gcc client.c -o client
# ↑ a tab character, not 4 spaces

server: lab.h server.c
	gcc server.c -o server

clean:
	rm -f client server
```

Terminal

```
$ make
gcc client.c -o client
gcc server.c -o server

$ make clean
rm -f client server
```

Note: Makefiles must be indented using TABs and not spaces!

- Makefile tutorial: https://makefiletutorial.com/

7. File Validation

For a reliable transmission mechanism, the received file should be identical to the original file. In Unix and Linux, the “cmp” command can be employed to compare whether two files are the same.

If discrepancies are identified between the files, you can install and use “colordiff” along with “xxd” to pinpoint the specific bytes where the differences occur. This information can assist you in debugging your C code.

Terminal

```
$ cmp -s video.mp4 download_video.mp4 && echo "Same!" || echo "Different!"
Same!

$ sudo apt update && sudo apt install colordiff
$ colordiff -y <(xxd video.mp4) <(xxd download_video.mp4)
```

If you prefer to use a hex viewer with a GUI interface, using EmEditor or Notepad++ with hex editor plugin (video) is also an option.

8. Submission

(a) Please provide a readme.pdf file to show what functionalities your program has.

- For example, is it able to be built by the “Make” tool? Does it meet all requirements? How do you handle the stop-and-wait mechanism? What have you learned from this lab?
- If you can run your C program, please provide screenshots to show how it works (similarly to our example mentioned above).

(b) Compress/zip all the C source files, readme.pdf and related files ***directly*** into a single file named studentID_lab3.zip (e.g., 111012345_lab3.zip) ***without including the parent directory***. The content of the zip archive should look something like:

```
├── (my_module.h / my_module.c, if applicable)
├── lab.h
├── client.c
├── server.c
├── Makefile
├── video.mp4
└── readme.pdf
```

Do ***NOT*** include any parent folder (e.g. 111012345_lab3) in the zip archive.

(c) If you have implemented selective repeat mechanism, compress/zip the associated files into studentID_bonus.zip.

(d) Upload your zip file(s) to eeclass.

(e) Discussion is encouraged; however, plagiarism is not allowed. We will use tools like Moss for similarity comparison. If plagiarism is detected, 0 points will be given.

(f) If you have referred to any books or online materials, please indicate the source in the readme.pdf to avoid from being mistaken for plagiarism. For example, you can add a “Reference” section:

Reference
[1] *How to do socket programming in C*, https://example.com/
[2] …

(g) Submit your assignment by the deadline. Late submissions will not be accepted, and a score of 0 will be assigned.