ARIEL UNIVERSITY

School of Computer Science

Computer Networks, Winter 2025

Assignment 3 – The Transport Layer

This assignment is to be done up to pairs, with exceptions approved by emailing the course coordinator.
You must upload all the necessary files in a ZIP file named after both of the student's IDs. For example, $123456789\_987654321.zip$.

1) Please ensure that you submit your assignment on or before the deadline indicated in the submission box.

2) All assignment files, including code, Wireshark .pcap files, and a PDF with necessary screenshots and descriptions, must be submitted in a ZIP file.

3) You may use any reference material available on the course Moodle or provided during exercises.

4) While you're free to refer to any online resources, it's important to avoid copying entire code blocks from websites, including those found on GitHub. If a student is caught doing so, they will fail the assignment with a score of 0. Additionally, please make sure to document all websites used to complete the assignment in the PDF.

5) Your code should be well structured and designed. To ensure clarity, include helpful comments in the code and use meaningful variable names.

6) **The assignment code must be written in python only.**

7) The assignment is individual and should not be assisted by anyone outside or inside the university. You can seek help during reception hours from the course staff or ask a question in the course forum. It is forbidden to share code sections, upload solutions, or parts of solutions on any Internet website or communication platform.

8) **If you use any AI, please add the prompts. AI should/may help but not solve**

Good Luck!

ARIEL UNIVERSITY

Requirements

In this assignment, you will implement a simplified version of a reliable, ordered data transfer protocol, inspired by the core principles of TCP.
You will write two programs, **a client and a server**, communicating over a network connection. Your code should be in Python.

The actual establishment of this connection will be handled by standard **TCP sockets** behind the scenes, so from your perspective, once the socket connection is established, you can send and receive data just as you would with any stream-based communication. Your focus is on implementing the logic of message segmentation, a sliding window, immediate server acknowledgments, and timeout-based retransmissions.

The assignment begins with a server listening to clients' requests, and a client initiating a TCP connection to the server.
Once the connection is set up, the client asks the server for the maximum size of a single message the server is willing to handle. This is done by sending a message from the client to the server requesting the maximum message size. The value will be passed either **as input from the user; or from a text input file** whose structure is detailed in the last section of the assignment. You must **provide support for both options**. Upon receiving this request, the server responds with a number representing the largest allowed message size in bytes.

The client aims to send a text message to the server. From now on, the client must respect this limit and never send messages larger than what the server specified. For example, suppose the server says the maximum message size is 20 bytes and the client must send 80 bytes of data. In that case, the client will divide its data into four messages of 20 bytes each, labeled M0, M1, M2, and M3, with increasing sequence numbers starting from zero for the first message.

With the maximum message size determined, the client's next task is to send the data reliably and in order. To do this, the client employs **a sliding window approach**. The sliding window determines how many messages can be sent **without waiting** for

acknowledgments. The window size will also be passed as input from the user or from a text input file whose structure is detailed in the last section of the assignment. You **must provide support for both options**. The window's size remains fixed throughout the communications between the client and the server (no dynamic windowing).

In this assignment, **only the server sends acknowledgments**. The server **acknowledges each message as soon as it is received**. This acknowledgment states the highest contiguous sequence number of messages it has received so far, starting from zero. If the server receives M0 and later M2 but not M1, it cannot advance the contiguous sequence beyond M0. When it received M0, it would have sent ACK0. When it receives M2 out of order, it stores it but sends another ACK0 since M1 is missing, and it cannot declare that messages beyond M0 are consecutively received. Once M1 arrives, the server can send an acknowledgment that includes M2, for example, ACK2, indicating that it now has M0, M1, and M2 all in order.
This acknowledgment behavior lets the client know which messages have been received continuously. As soon as the client gets ACK0, it knows M0 was received and slides its window forward.

The client maintains a single timer associated with the oldest unacknowledged message in the current window. If the timer on the client side expires, the client resends all the messages that remained without acknowledgment.
The client initializes a timer when he sends the first message of the current window. If the timer expires before the client receives an acknowledgment for that message, the client assumes that the message or its acknowledgment was lost.
The timeout value will be passed either as input from the user or from a text input file whose structure is detailed in the last section of the assignment. You must provide support for both options.

After retransmission, the timer for that same first message is restarted in the current window. Once the first message in the current window is finally acknowledged, the timer either stops if there are no other outstanding messages or moves to the next oldest unacknowledged message if more remain (i.e., the first message in the current window).

ARIEL UNIVERSITY

This timer and retransmission logic ensures the eventual delivery of all messages, mimicking the basic reliability mechanism of TCP in a simplified form.

Additional notes

Your code should support the source of the following variables either from the user's input or from a .txt file.

The structure of the file is as follows:
message: a string
maximum_msg_size: a string representing the number of bytes as integer.
window_size: a string representing the number of messages in a window as an integer
timeout: a string representing the number of seconds before timeout as an integer

Example input from file:
message:" This is a test message"
maximum_msg_size:400
window_size:4
timeout:5

Be advised that your code must work correctly when parameters' values result in multiple messages.

Good Luck!