Can Günyel 150200049

Introduction

The primary objective of this simulation project is to comprehensively explore the behavior of communication models, specifically Reliable Data Transfer (RDT) models and the Transmission Control Protocol (TCP). Through simulating various scenarios, the aim is to gain insights into their performance under diverse conditions. This report delves into the design, implementation, and outcomes of the simulation, shedding light on the intricacies of each model.

System Architecture

For each model, three scripts are employed: sender, receiver, and a GUI script named "RdtXmain." Both the sender and receiver scripts maintain their individual buffers. When a sender wishes to dispatch a packet, it writes to the receiver's buffer. The receiver, consistently monitoring its buffer, detects when a packet is sent and responds with an acknowledgment (ACK) if the model necessitates it. Communication between the sender, receiver, and the graphical user interface (GUI) script facilitates reporting of operations. The GUI reads from the buffers and visualizes the entire process.

To enhance system flow tracking in the simulation, additional visual messages have been added. These include package numbers in primal RDTs, as well as alerts for message and ACK receipt. While these details are typically not transmitted as messages in the real world, they provide valuable observations for simulation observers and make it easier to keep track of the process.

Files

Main Files

This Python script orchestrates the execution of three separate processes—sender, receiver, and GUI. It utilizes the subprocess module for executing these processes in parallel and threading to manage concurrency. It also clears buffers since they can have former data from previous run.

Buffers

The files "receiverbuffer.txt" and "senderbuffer.txt" serve as dedicated buffers for the respective parties involved. These files facilitate communication between the sender and receiver components. In contrast, the GUI files play a distinct role in mediating communication between the GUI and the participating parties. "gui_figure.txt" is utilized to denote the currently active function at the sender's side, “gui_file.txt" is utilized to record sent messages, creating a log of the communication events, "gui_input.txt" states user input received from the GUI during the simulation.

Sender and Receiver Files

Each communication model features a unique implementation, focusing on simulating the send and receive functionalities. In each model, the scripts continually monitor their respective buffers, updating the other party's buffer file at intervals to simulate the exchange of messages. To ensure visibility and comprehension, reported messages, both sent and received, are communicated to the GUI through the "gui_file.txt" file. During the sending and receiving processes, the scripts regularly check their "gui_input.txt" file to identify any occurrences of unusual cases, such as lost or corrupted packages. To provide real-time updates for Figure 3.8 in the textbook, each script communicates its current function to the GUI. The scripts operate within a persistent while loop,

ensuring continuous package transmission until a finish signal is received (exclusive to the TCP model).

Model File

This Python script utilizes the *Tkinter* library to construct a GUI simulating a communication system. It divides the window to 4 frames as desired. There is Reliable Data Transfer Service Model as shown in Figure 3.8 at left upper corner, state transitions at right upper corner, a text and graphic that gives information about the status of the packages at left bottom corner, input section and the call status of active functions in right bottom corner. Script constantly reads its buffers and update itself to visualize the changes in the system. Also takes input from user and writes into “gui_input.txt” buffer to inform sender and receiver parties. To indicate states and active functions switch between photos that points current states and functions. I made that switches according to packages’ status.

Simulation

Simulation is achieved through distinct Python scripts for sender and receiver, utilizing text files as buffers. This approach circumvents established TCP technologies, aligning with the project's goal of constructing TCP and primitive models from scratch without relying on external libraries.

Operational in a local environment, the system is constrained from experiencing lost or corrupted packets. To simulate these scenarios deliberately, the system is coerced into sending corrupted and lost packages. The models run as distinct scripts, with the simulation initiated by executing the "project3.py" file in codes folder (command line should be opened in codes folder since paths adjusted for that) and specifying the desired model. Delays have been introduced to certain steps to ensure a comprehensible visualization of the simulation process.

- To simulate lost packages, input "lost" and submit via "Check Input" after any "Packet X sent" message. The system randomly determines whether the packet or the ACK/NAK is lost.
- For simulating corrupted packages, input "corrupt" and submit via "Check Input" after any "Packet X sent" message.
- To conclude the TCP connection, input "finish" and submit via "Check Input" after any message. Subsequently, an additional package is sent before initiating the closing procedure.

The state transactions are visually represented with arrows, providing a clear indication of the current state for both the receiver and sender in the upper right corner. The graph in the lower-left corner moves downwards when different packages are sent, but no downward movement occurs during the time taken for the same package. This decision, influenced by the scope of the course, has been mentioned for transparency.

The final TCP state transition for the client necessitates a 30-second waiting period. This temporal delay is deliberate and is required to observe the closing procedure. Patience is recommended to witness the output during this waiting period. TCP’s function calls (right bottom corner) is simplified since it is also simplified in book. Therefore, it has handshake, established and closure procedure calls. However, it doesn’t have package sending and receiving (lost, corrupt included) calls. It still has that functionalities but not show each call.

Can Günyel 150200049

Assignment Requirements

The assignment mandates running the simulation for at least one normal transaction, one premature ACK, one lost packet, and one timeout event. It is acknowledged that certain assumptions in RDT models, such as RDT1.0 and RDT2.0, will prevent the occurrence of these events.

RDT1.0

- The conditions depicted for RDT1.0: no loss, no corruption.

Therefore, only normal transaction can be observed.

RDT2.0

- The conditions depicted for RDT2.0: no loss, corruption may occur.

Therefore, normal transaction and corrupted packet can be observed. Packet can be corrupted by “corrupt”. And for corrupted packet we’ll get a NAK message instead of ACK.

RDT3.0

- The conditions depicted for RDT3.0: both loss and corruption may occur

Therefore, normal transaction, premature ACK, lost package and timeout can be observed. Packet can be corrupted by “corrupt”. And for corrupted packet we’ll get a NAK message instead of ACK. Packet or ACK/NAK can be lost by “lost”.

TCP

- The conditions depicted for TCP: both loss and corruption may occur

Therefore, normal transaction, premature ACK, lost package and timeout can be observed as RDT3.0. At the beginning of TCP package transfer system ensure that it established a connection by three-way handshake. Then it starts to send packages. TCP package size is set to constant number 8. Therefore, each ACK will have a package number 8 greater than the packet to be acknowledged. System can be stopped by “finish”. After submitting, we’ll see closing procedure of TCP and then system will stop sending packages.

Sample Videos

Sample videos attached to Google Drive link since size of them is too much for Ninova. I shot an example videos because I thought some details might not be visible when viewed on a different monitor because there are shifts in frames with different window sizes.

- In RDT1.0 I show continuous package transfer.
- In RDT2.0 I show continuous package transfer but I sent one corrupt package and show NAK message.
- In RDT3.0 I show continuous package transfer but message numbers are state numbers in here. I sent one corrupt message and show NAK message. I also sent a packages which will timeout by both ACK lost and package lost.
- In TCP I show continuous package transfer but message flow ended with “finish” call. I show handshake procedure, then normal message transaction. Here ACK messages have a package number 8 greater than the packet to be acknowledged since message size is constant 8. Then I show corrupt and lost packages. At last I ended flow with “finish” call and show closing procedure.

Can Günyel 150200049

Results and Discussion

Since this assignment was a simulation assignment, I did not obtain any data to observe and conclude. But I better understood how the different packet sending protocols we saw in the course worked. Apart from this, I learned and practiced using buffers for the communication of the parties. And I realized that this is the very basis of the physical side of sending packets. Apart from this physical aspect, I learned what precautions can be taken in different scenarios in case of undesirable situations: corrupt or lost packages. I also gained deeper knowledge about handshake and closing procedures. Apart from these, I gained experience with GUI, but since it is not within the scope of this course or a prerequisite lecture, I do not think it should be a requirement and I hope it will not be one of the evaluation criteria.

I also want to mention that it was the most comprehensive homework I have ever been assigned but at the same time the most poorly explained one. Moreover, I couldn't get any response to my mails from both instructor and TA. I hope next time description will be more explanatory and include any example that what should the expected output like.