Computer Network project's work report

Project number 2

Author: Amir Hamidi

Student Number: 9907794

January 2022

Table of Contents

Introduction

In this project, we have implemented the most advanced version of stop and wait (SAW) protocol (over UDP). this version of stop and wait can handle segment lost and segment corruption. Since this protocol is completely defined in the text book, we point out only the important details in implementation of this protocol:

Segments and Acks Structure

1. We have considered 1-byte sequence number in order to have a byte base environment. In this 1-byte sequence number, the rightmost bit indicates the sequence number and the other 7, are 0.

2. 16-bit as segment indicator, which is 0101010101010101

3. 16-bit as ack indicator, which is 1010101010101010

4. 16-bit checksum in both segments and acks

5. a varying maximum-segment-size and probability-of-losing-a-packet which can be set from arguments(for easier execution, these values are also hard coded (to default values) and you can choose either to enter the argument values or not).

Stop and Wait Specification

1. We have used timers for handling packet loss.

2. we have written a function that calculates the checksum in order to check whether a receiving segment/ack is corrupted or not.

3. we have used sequence number to control the packet order.

4. we have used segment and Ack indicator in order to identify each packet as Ack or data segment.

In this project we have **only** implemented the **STOP AND WAIT** protocol.

Code Explanation

Unlike its name, we don't want to get into details of coding in this section as it is not the goal of this work report. but here we notice some of the most important aspects of overall codes and files that has been written and created for this project.

Files (.py)

Sender_SAW.py: this file is playing the role of the sender in SAW protocol.

Receiver_SAW.py: this file is playing the role of the receiver in SAW protocol.

Function.py: this file contains the necessary functions for implementing this project. checksum_calculator, byte2str and str2byte are some of the most important functions of this projects.

Text_Maker.py: using the text_maker function, this file can write *whatever we want,* for *how many times we want*, in *any where we want*.

Plotter.py: this file is used in order to plot the figures of the simulations.

TXTs Folder

sent.txt: this file is the file to be transmitted by the Sender_SAW.py. (like before, this file is the default value to be transmitted; but you can also use argument values to choose the file you want to be transmitted)

received.txt: this file is what we have received by the Receiver_SAW.py

results.txt: since each simulation takes a lot of time, instead of plotting the figures immediately after the end of simulation, we record each results of the simulation in results.txt and use this file and plotter.py in order to plot the figures.

Variables

end_check: this variable is defined in order to lets the receiver know that the transmitting is over and it can now write the receiving message into the received.txt file.

sequence_number: this variable is the sequence number of the current segment.

p_squence_nubmer: this variable is the sequence number of the previous segment.

auxiliary_byte: this variable helps us calculating the checksum value. Since checksum is 16 bits, we need an even number of bytes to calculate the checksum. This auxiliary_byte is used in case of odd number of bytes.

Almost all other variables are fully represented by their name and need no more description.

We have also left some notes in Sender_SAW.py and Receiver_SAW.py and therefore these files are completely comprehensive (despite they are python codes).

Walkthrough for Running the Project

In order to run the project, first we should turn on the receiver(Running the Receiver_SAW.py file) and then turn on the sender. (run the Sender_SAW.py file)

if you turn on the sender first, a message pops up saying that the receiver is turned off. in this case you should turn on the receiver and press enter in the sender side(where you have run the Sender_SAW.py).

As it was mentioned before, you can choose the maximum segment size, and file you would like to send with argument values in the sending side, and probability of the segment loss in the receiver side. Here's a simple example of how deliver the maximum segment size and target file to the sender, in the terminal:

python .\Sender_SAW.py 1250 TXTs/sent.txt

where 1250 is the maximum segment size and TXTs/sent.txt shows that the file **sent.txt** in the folder **TXTs**, is the file we would like to send.

After running both files with desired parameters, the sender will send the file to the receiver 5 times and measures the elapsed time it takes to send the whole file

for each iteration. it then calculates the average of these five values and add this average time to the results.txt file.

it is important to note that the receiver will not be turned off after that 5 times so that we can run the Sender_SAW.py over and over again with new configuration (for example new **maximum segment size**), without needing to turn on the receiver each time.

After running the files for each **probability of segment loss** and **maximum segment size**, we can use plotter.py to draw the figures (these figures are illustrated in the next section, **Simulation Results**).

NOTE: Besides this work report, there is Presentation, video file, right next to this PDF. In case of any confusion with reading this work report, feel free to check this video file as I have run the entire project by myself.

Simulation Results

In this part we have simulated the transmission of a fixed-size file(as mentioned before sent.txt) and receiving it by a receiver while changing the **maximum segment size** and **probability of a segment loss**.

Simulation Notes

Simulations are completely according to the **Simulation and Results** part of the **project description**. (except the line with *This format*)

Transmitted file contained the string "**Hello world**", 200000 times and its size was about 2.344 MB. This file was created by Text_Maker.py.

Maximum Segment Size

The following figure shows the elapsed time for sending the **sent.txt** file vs the maximum segment size

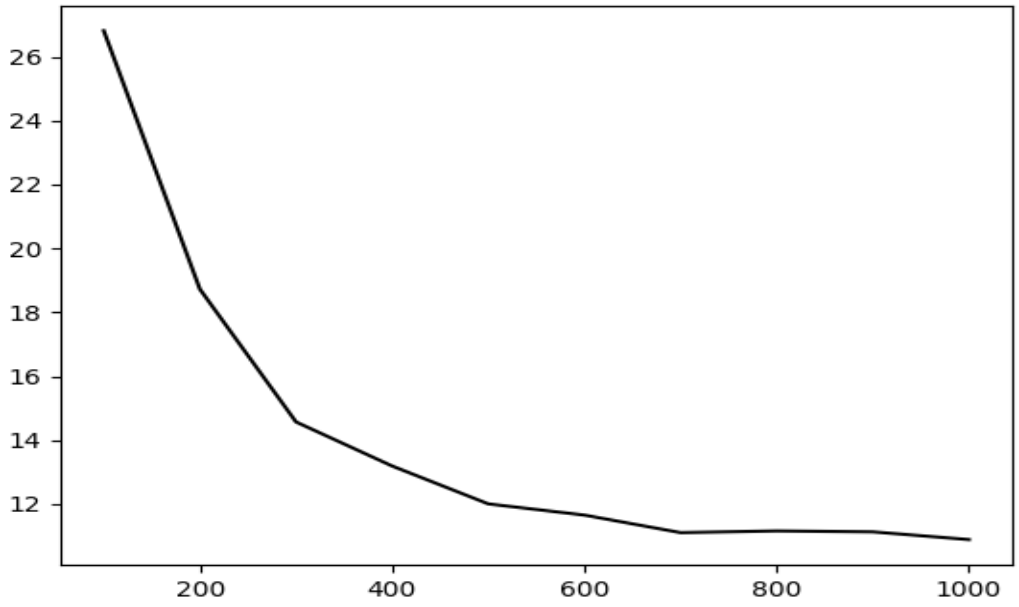

FIGURE 1: MAXIMUM SEGMENT SIZE FROM 100 TO 1000(SAW)

As it can be seen, by increasing the maximum segment size, the elapsed time reduces. Of course, this is obvious because by increasing the number the maximums segment size, the total number of transmission and hence the total elapsed time will reduce.

The segment loss probability was 0.01 during this simulation.

Segment Loss Probability

The following figure shows the elapsed time for sending the **sent.txt** file vs the probability of the segment loss

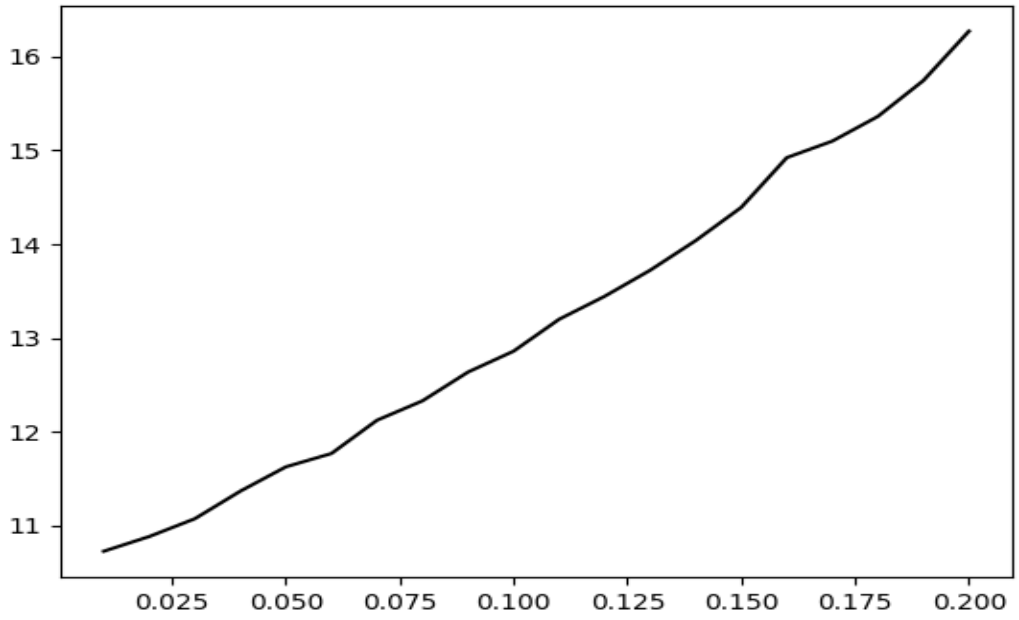

FIGURE 2:SEGMENT LOSS PROBABILITY FROM 0.01 TO 0.2 (SAW)

As it can be seen, by increasing the segment loss probability, the elapsed time increases. That's because with greater probability of loss, we lose more segment and we need to resend more segment. this means we need more time to send the whole message.

the maximum segment size was 1000 bytes during this simulation.