PORTFOLIO 2

DATA2410 – OSLO metropolitan university

FAHMI HAYBE MOHAMMED / s362106, BEHDAD NIKKHAH / s362085, TERESA PHAM / s345368, JAN NICOLE RUBIC YAO / s362049

2023

TABLE OF CONTENTS

[1. Introduction 2](#_Toc134206825)

[2. Background 3](#_Toc134206826)

[3. Implementation 3](#_Toc134206827)

[4. Discussion 3](#_Toc134206828)

[5. Conclusions 5](#_Toc134206829)

[6. References 5](#_Toc134206830)

# 1. Introduction

An introduction should tell the reader why this work is interesting.

This report contains an implementation of a simple transport protocol (DRTP) - Data Reliable Transport Protocol. This program is built using python and socket module with UDP to transfer data. Given that UDP is an unreliable protocol, DRTP will ensure a reliable connection. The network tool Mininet is also an important tool to build a virtual network regarding the simple topology. This will be useful to evaluate the virtual network resources and performance when using DRTP. (Include the bonus as an implemented work)

The test would be done by the custom written code to transfer a file, and to measure the throughput using different RTTs (round-trip-time). Different various functions will be used to trigger retransmission of packets either due to loss or being skipped. The performed tests should reveal if the code is handling potential losses, reordering and duplicates.

The approach to solve the problem is to implement a (three-way handshake) between the client and server as a way of establishing a connection. Each reliable function has its own client and server methods which results in a more maintainable and flexible code. This makes it easier to alter the different reliability functions as needed.

Include limitations here!!!

This document is structured and divided into five different segments, including the list of references at the end of the document. The first chapter would start by presenting the background, containing appropriate theoretical instructions. The next chapter is the implementation of the custom/customized written code, and what sort of functions are used. The discussion of the different tests performed would be discussed in the next following chapter, also following the result of the tests. The last chapters consist of a conclusion and a list of sources.

# 2. Background

Your background section should include the appropriate theoretical background that your reader should know before delving into the details. Explain reliability, stop and wait, go back n and selective repeat

Reliable data delivery is crucial in computer networking to make sure that data is being transmitted correctly and error-free. Transport protocols help in achieving this as they sit on top of the network layer providing flow control, error detection and correction among other services. A common issue which transport protocols try to solve is packet loss or corruption during transmission. In this project, we implemented three reliability functions: stop-and-wait, go-back-n, and selective repeat.

Stop-and-wait is the simplest of the three reliability functions implemented in this project and is a commonly used protocol which ensures reliable data delivery. It does so by transmitting packets one by one and waiting for an acknowledgement (ACK) from the receiver before transmitting the next packet. In the case of a timeout where no ACK is received after a certain amount of time, the sender retransmits the packet.

Go-back-n applies a sliding window protocol where the sender transmits a certain number of packets. With this protocol, the receiver only accepts packets if they are in order and discards those which are not. Similar to the stop-and-wait protocol, when a timeout occurs the sender retransmits the packets after the last acknowledged in-order packet.

Additionally, selective repeat also applies a sliding window protocol. However, as opposed to retransmitting all the packets, the sender will only retransmit the lost packet(s) within the specified window. This means that the receiver accepts all the packets even if they are out of order, given that they are within the window. The sender waits until all the packets within the window have been acknowledged before transmitting the next set of packets.

# 3. Implementation

Add your implementation details here - with code snippets, diagrams and other details.

In this Portfolio, we are to implement a simple transport protocol – DATA2410 Reliable Transport Protocol (DRTP) that provides reliable data delivery on top of UDP. Our Protocol will ensure that data is reliably delivered in-order without missing data or duplicates. To test our custom protocol, we also had to implement a file transfer application program that uses DRTP/UDP protocol.

In our custom protocol, we will implement three reliability functions that users can choose from the command line argument. The DRTP protocol code is divided into two functions for each reliable method. A receive function and a send function. Additionally, there are common functions that all three reliable methods use, such as functions for establishing connection.

The Stop\_And\_wait reliable method is divided into functions **“RECV\_SAW()”** and **“SEND\_SAW()”**. The former is responsible for receiving data packets sent by the sender and sending ACK packets to confirm the receipt of each packet. The **“RECV\_SAW()”** starts with a handshake with the sender and initializes the expected sequence number and waits for data packets to arrive. Upon receiving a packet, the function first checks the sequence number. If it is the expected sequence number, the function sed an ACK message to confirm receipt, increments the expected sequence number, and concatenates the received payload data. If the received packet has an incorrect sequence number, the function sends a duplicate ACK message or discard the out-of-order packet. If the FIN flag is set, the function sends an ACK message, closes the socket, and returns the received data. The **“SEND\_SAW()”** function begins with a handshake and sets the sequence number variable to just one. Once in a loop, it sends a data packet (up to 1460 bytes) with the current sequence number and waits for the recipient to respond with an ACK message. The sequence number is updated, and the next packet is sent if the ACK message is valid. It resends the previous packet with the prior sequence number if the ACK message is a duplicate. While awaiting an ACK message, if there is a timeout, the packet is resent with the current sequence number. When there is no more data to send, the function closes the connection.

The Go\_Back\_N reliable method consists of **“RECV\_GBN()”** and **“SEND\_GBN()”**. The former first performs the three-way handshake by calling **“handle\_handshake()”**. Then, it initializes variables such as **“expected\_sequence\_number”** and **“received\_data”** before it waits for data packets to arrive. When a packet arrives in-order, it sends back an ACK message. If SYN message is received, the function calls **“handle\_handshake()”** to redo the three-way handshake. If the received packet has correct sequence number and is not a FIN message, the function concatenates the data to the **“received\_data”** variable and sends back an ACK message. If FIN message is received, the function sends back an ACK message, closes the socket, and return the received data. Otherwise, if a packet arrives with wrong sequence number and FIN flag is not set, it discards the packet.

The **“SEND\_GBN()”** initiates the three-way handshake by calling **“initiate\_handshake()”** and initializes variables such as next sequence number, base sequence number, unacknowledged packets, data offset, and whether FIN message has been sent. This function sends data packets as long as the number of sent packets is less than the window size. Payload for each packet is calculated by slicing it from the data. **“SEND\_GBN()”** then sends the packet, adds it to unacknowledged packets, and increments the next sequence number and data offset. The functions waits for an ACK message for the sent packet if a FIN message has not yet been sent. If an ACK message is received with the correct acknowledge number, the function moves the base sequence number for the window and updates the unacknowledged packets list. If no ACK message arrives, “SEND\_GBN” resends all unacknowledged packets in the window. The function waits for an ACK message for the FIN message if one has been sent before exiting the loop if one has.

# 4. Discussion

Test cases here and show how you handle losses, reordering and duplicate packets.

1. Run the file transfer application with stop-and-wait reliable protocol, GBN with window sizes 5, 10, 15, and GBN-SR with window sizes 5, 10, 15 using RTTs 25, 50 and 100ms. Calculate

throughput values for all these cases and explain your results.

|  |  |  |
| --- | --- | --- |
| Stop\_and\_wait | RTT | Output |
| Test 1 | 25ms | Successful received the file (saved in test-1-saw) |
| Test 2 | 50ms | Successful received the file (saved in test-2-saw) |
| Test 3 | 100ms | Successful received the file (saved in test-3-saw) |

1. Write a test case to skip an ack - this will trigger retransmission. Test with all three reliable functions.
2. Write a test case to skip a sequence number to show the out-of-order delivery effect. This will also trigger retransmission. Test with GBN, and SR. Report your results.
3. Use your artificial testcases to show the efficacy of your solution. Your solution should be able to handle losses, reordering and duplicates.

Test Case Name: Test RECV\_SAW with skip\_ack=True

Test Case Description: Test the RECV\_SAW function for skipping the first acknowledgment message, which should trigger retransmission at the sender side.

## Test Case 1

Check that retransmission is triggered when a receiver skip sending an ACK message in all reliable functions.

### Steps

1. Run the application.py in server/receiver mode with the optional flag -t skip\_ack with one of the reliable functions.
2. Run the application.py in client/sender mode with the same reliable method as the server/receiver.

### Expected Results

The client/sender should retransmit the previously sent packet because of timeout in Stop\_And\_Wait reliable method. In Go\_Back\_N, it should retransmit all packets in the window. Meanwhile in Selective Repeat, it should retransmit only packets not ACK’ed by the receiver.

### Actual Results

All retransmission happened as expected.

## Test Case 2

Check the out-of-order delivery effect when a sequence number is skipped. It should also trigger retransmission.

### Steps

1. Run the application.py in server/receiver mode with either Go\_Back\_N or Selective Repeat reliable function.
2. Run the application.py in client/sender mode with the optional flag -t loss with the same reliable method as the server/receiver.

### Expected Results

When the server/receiver receives the packets where a sequence number is skipped in Go\_Back\_N reliable function, it will discard out-of-order packets. Timeout will occur on the client/sender side, triggering retransmission of all packets in the window, including the skipped packet. In Selective Repeat, timeout will occur only for the unacknowledged packet, resulting in a retransmission.

### Actual Results

The program behaved as expected.

# 5. Conclusions

A summary of what you have done, including key results.

# 6. References

Islam, Safiqul. (2023). The Transport Layer [Lecture slides], DATA2410: Networking and Cloud Computing. Oslo Metropolitan University.