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

# 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\_STOP with skip\_ack=True

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEST ID | Scenario | Test Step | Expected Result | Actual Outcome |
| SKIP\_ACK\_SAW | Verify that retransmission occurs when first ACK message is skipped | 1. Run the application.py program in server/receiver mode with the optional flag -t skip\_ack 2. Run the application.py in client/sender mode with the same reliable method as the server/receiver. 3. Verify that the receiver skips sending the first ACK message. 4. Verify that the sender retransmits the packet | * The server/receiver should skip sending the first ACK message. * The client/sender will retransmit the previously sent packet because of timeout | * The sender retrasmitted the packet as expected after timeour |

# 5. Conclusions

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

# 6. References