**Portfolio 2: DATA2410 Reliable Transport Protocol (DRTP)**

This portfolio will count towards your final grade (40%). It's a group project - please choose your own group members (min=3 and max=4). Your submission will be marked with respect to how well you have fulfilled the requirements listed below.

**Prerequisites**

We have already covered the required concepts in our lectures, labs, and mandatory assignments which you need for this portfolio. Should you be missing lectures and lab sessions, it is your own responsibility to catch up to the competence level that you are supposed to get it from the lectures and lab sessions. Also, try to make good use of the lectures and lab sessions because we will cover some important concepts there.

**Overview**

You will implement a simple transport protocol - DATA2410 Reliable Transport Protocol (DRTP) that provides reliable data delivery on top of UDP. Your protocol will ensure that data is reliably delivered in-order without missing data or duplicates.

In this portfolio, you will write code that will reliably transfer file between two nodes in the network. You MUST implement two programs:

1. DRTP that will provide reliable connection over UDP
2. A file transfer application:
   1. A simple file transfer client
   2. A simple file transfer server

**Implement a reliable data transfer protocol (DRTP)**

You must use UDP as the transport protocol and add reliability on top of that. You are not supposed to use any other transport protocol which already provides reliability (e.g., TCP). You must construct the packets and acknowledgements. You must establish the connection and gracefully close when the transfer is finished.

**A file transfer application**

You will implement a file transfer application (application.py) where you will use -c to invoke a client and -s to invoke a server.

**A simple file transfer server**

A simple file transfer server will receive a file from a client through its DRTP/UDP socket and write it to the file system. The file name and the port numbers on which the server listens are given as command line arguments. NOTE: the port number of the server must also be the port number used by the client.

The server/receiver can be invoked with:

python3 application.py -s -f Photo.jpg -m gbn

If I want to skip an ack to trigger retransmission at the sender-side:

python3 application.py -s -f Photo.jpg -m gbn -t skipack

**A simple file transfer client**

A client reads a file from the computer and sends it over DRTP/UDP. The file name, server address and port number are given as command line arguments.

It is possible to transfer two source files at the same time. For simplicity, we'll only allow one transfer.

The sender can be invoked with:

python3 application.py -c -f Photo.jpg -r gbn

If I want to test duplicate/reordering/packet\_loss scenario:

python3 application.py -c -f Photo.jpg -r gbn -t loss

**Details**

A sender adds a header in front of the application data before it sends the data over UDP. The header length is 12 bytes and application data is 1460 bytes. A sender therefore sends 1472 bytes of data to a receiver, including the custom DRTP header. The header contains a sequence number (packet sequence number), an acknowledgment number (packet acknowledgment number), flags (only 4 bits are used for connection establishments, teardown, and acknowledgment packets), and receiver window (window). Receiver window is advertised by the receiver for flow control, set it to 64. The DRTP header looks like this:

0 1 2 3

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Sequence Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Acknowledgment Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Flags | Window |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|<------------------------------------------------------------->|

bits(32) / bytes(4)

protocol "Sequence Number:32 bits, Acknowledgment Number:32, Flags:16 ,Window:16.

Together with the header, your application data that you will send over UDP:

0 1 2 3

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Sequence Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Acknowledgment Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Flags | Window |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

+ Message Body +

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

If a DRTP packet contains no data, then it is only an acknowledgment packet ("ACK"). It confirms having received all packets up to the specified sequence number. An acknowledgement packet (only 12 bytes) is sent by the receiver with the acknowledgment number and set the ACK flag bit in the Flags field.

Format of the flag fields:

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |S|A|F|R|

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

S = SYN flag, A= ACK flag, F=FIN flag and R=Reset flag

**Connection establishment and tear down**

A sender initiates a three-way handshake with the receiver (similar to TCP) to establish a reliable connection. A sender first sends and empty packet with the syn flag ON, a server then responds with packet with SYN and ACK flags set and establishes the connection. Upon receiving syn-ack packet, a sender sends ack to the server.

When the transfer is finished the sender sends a FIN packet. The receiver sends an ACK when it receives the FIN packet and closes the connection. When the sender receives the ACK, it also gracefully closes the connection.

**Reliability functions**

You will implement the following three reliability functions and users will be able to choose them from the command line argument (using -r):

1. A stop and wait protocol (stop\_and\_wait()): The sender sends a packet, then waits for an ack confirming that packet. If an ack is arrived, it sends a new packet. If an ack does not arrive, it waits for timeout (fixed value: 500ms, use socket.settimeout) and then resends the packet. If sender receives a duplicate ACK (ack of the last received sequence), it resends the packet.
2. Go-Back-N (GBN()): sender implements the Go-Back-N strategy using a fixed window size of 5 packets to transfer data. The sequence numbers represent packets, i.e. packet 1 is numbered 1, packet 2 is numbered 2 and so on. If no ACK packet is received within a given timeout (choose a default value: 500ms, use socket.settimeout() function), all packets that have not previously been acknowledged are assumed to be lost and they are retransmitted. A receiver passes on data in order and if packets arrive at the receiver in the wrong order, this indicates packet loss or reordering in the network. The DRTP receiver should in such cases not acknowledge anything and may discard these packets.
3. Selective-Repeat (SR()): Rather than throwing away packets that arrive in the wrong order, put the packets in the correct place in the receive buffer. Combine both GBN and SR to optimise the performance.

See textbook and lecture slides for more details.

**Discussion:**

Test your code in mininet using `simple-topo.py

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.
2. Write a test case to skip an ack - this will trigger retransmission. Test with all three reliable functions.
3. 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.
4. Use your artificial testcases to show the efficacy of your solution.

**Submission**

You must submit **yourgroup\_studentids\_portfolio2.zip** through the Inspera exam system. You will get access to inspera before the deadline. You must add your group members on the Inspera system.

Your zip file should contain:

1. Source code of final transfer application (application.py) and DRTP
   * where the code is well commented. Document all the variables and definitions. For each function in the program, you must document the following:
     + what are the arguments.
     + What the function does.
     + What input and output parameters mean and how they are used.
     + What the function returns.
     + Correctly handling exceptions.

Here is an example:

# Description:

# checks if the port and addresses are in the correct format

# Arguments:

# ip: holds the ip address of the server

# port: port number of the server

# Use of other input and output parameters in the function

# checks dotted decimal notation and port ranges

# Returns: .... and why?

#

1. A report with your names, IDs and title. Your document must be submitted in the pdf format.
   * read the instructions (see below)
2. A README file: info on how to run application.py with the flags

Example final structure of your folder:

├── README

├── src

│   ├── code

├── studentids\_portfolio2.pdf

**NOTE** I strongly recommend that you confirm that the archive you uploaded contains what it should by downloading it and unpacking it again after submission. If you deliver the wrong files, there is nothing I can do about it when I check. We also recommend you upload draft versions as you work, to ensure you have a deliverable in the system should you experience last minute technical difficulties etc.

**Project report**

You also need to submit a project report in the pdf format. **NOTE**: I won't accept word or any other format. The report must have a title page (your names, IDs and the title of your work), table of contents, and the following sections:

1. Introduction
2. Background
3. Implementation
4. Discussion
5. Conclusions
6. References (if any: mention sources)

The report cannot exceed 20 pages, including the list of references. The page format must be A4 with 2 cm margins, single spacing and Arial, Calibri, Times New Roman or similar 11-point font.

**Grading**

* Implementation of DRTP - 75%
* Project report - 25%
* **BONUS** 5%
  + Rather than using artifical testcases, use tc-netem to emulate packet loss, reordering and duplicate packets to show the efficacy of your code. Add the results in your discussion section.
* **BONUS** 5%
  + Rather than using a fixed value for the timeout, if your code can calculate the per-packet roundtrip time and set the timeout to 4RTTs, you will get 5% extra points.

**Deadline**

The deadline for submitting this exam is **May 16 2023 at 12:00 PM** Oslo local time.

This is a HARD deadline, failure to deliver on time will result in fail in this part-exam.

I am not going to handle any enquiries regarding medical extensions and so forth, you must contact the study administration directly.

Follow the submission guidelines. Do not make your respository public, and *do not copy*.