[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

You are required to implement a reliable transport protocol over UDP protocol. We refer to the reliable transport protocol that you will be programming in this assignment as Simple Transport Protocol (STP). STP will include most (but not all) of the features that are described in Sections 3.5.4 and 3.5.6 of the text Computer Networking (7th ed.). Examples of these features include timeout, ACK, sequence number etc. Note that these features are commonly found in many transport protocols. Therefore, this assignment will give you an opportunity to implement some of these basic features of a transport protocol. In addition, you may have wondered why the designer of the TCP/IP protocol stack includes such feature-less transport protocol as UDP. You will find in this assignment that you can design your own transport protocol and run it over UDP. This is the case for some existing multimedia delivery services in the Internet, where they have implemented their own proprietary transport protocol over UDP such as QUIC for Google Chrome.

[redacted]

[redacted]

[redacted]

1. Learning Objectives

On completing this assignment you will gain sufficient expertise in the following skills:

1. Being able to design and implement message passing protocols over IP network.
2. Detailed understanding of how reliable transport protocols such as TCP function.
3. Socket programming for UDP transport protocol.

2. Overview

As part of this assignment, you will have to implement Simple Transport Protocol (STP), a piece of software that consists of a sender and receiver component that allows reliable unidirectional data transfer. STP includes some of the features of the TCP protocols that are described in sections 3.5.4 and 3.5.6 of the textbook (7th edition). You will use your STP protocol to transfer simple text (ASCII) files (examples provided on the assignment webpage) from the sender to the receiver. You should implement STP as two separate programs: Sender and Receiver. You only have to implement unidirectional transfer of data from the Sender to the Receiver. As illustrated in Figure 1, data segments will flow from Sender to Receiver while ACK segments will flow from Receiver to Sender. Let us reiterate this, STP must be implemented on top of UDP. Do not use TCP sockets. ***If you use TCP you will not receive any marks for your assignment.***

You will find it useful to review sections 3.5.4 and 3.5.6 of the text. It may also be useful to review the basic concepts of reliable data transfer from section 3.4.

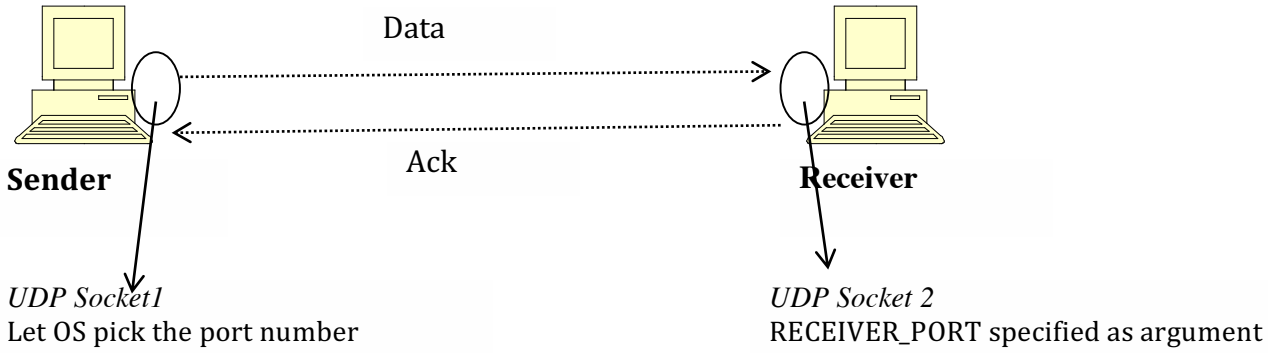

Figure 1: The basic setup of your assignment. A file is to be transferred from the Sender to the Receiver. Sender will run on the sender side while Receiver will run on the receiver side. Note that data segments will flow from the sender to receiver, while ACK segments will flow from the receiver to sender.

3. Assignment Specifications

This section gives detailed specifications of the assignment. There are two versions of this assignment, a standard version (with a total of **10 marks**) and an extended version (with a total of **12 marks** of which **2 marks are bonus marks**). The specifications for the extended version can be found in Section 5 of the specification. Note that the bonus marks may not be proportional to the amount of extra work that you will have to do. They are there to encourage you to go beyond the standard assignment.

3.1 File Names

The main code for the sender and receiver should be contained in the following files: `sender.c`, or `sender.java` or `sender.py`, and `receiver.c` or `receiver.java` or `receiver.py`. You are free to create additional files such as header files or other class files and name them as you wish.

3.2 List of features provided by the Sender and Receiver

You are required to implement the following features in the Sender and Receiver:

1. A three-way handshake (SYN, SYN+ACK, ACK) for the connection establishment. The ACK sent by the sender to conclude the three-way handshake should not contain any payload (i.e. data). See Section 3.5.6 for further details.

2. The four-segment connection termination (FIN, ACK, FIN, ACK). The Sender will initiate the connection close once the entire file has been successfully transmitted. See Section 3.5.6 for further details.

3. Sender must maintain a single-timer for timeout operation (Section 3.5.4 of the text).

4. Sender should implement all the features mentioned in Section 3.5.4 of the text, with the exception of doubling the timeout. The STP protocol must include the simplified TCP sender (Figure 3.33 of the text) and fast retransmit (pages 247-248). You will need to use a number of concepts that we have discussed in class, e.g., sequence numbers, cumulative acknowledgements, timers, buffers, etc. for implementing your protocol.

5. Receiver should implement the features mentioned in Section 3.5.4 of the text. However, you do not need to follow Table 3.2 for ACK generation. All packets should be immediately acknowledged, i.e. you do not have to implement delayed ACKs.

6. STP is a byte-stream oriented protocol. You will need to include sequence number and acknowledgement number fields in the STP header for each segment. The meaning of sequence number and acknowledgment number are the same as TCP.

7. MSS (Maximum segment size) is the maximum number of bytes of data that your STP segment can contain. In other words, MSS counts data ONLY and does NOT include header. Sender must be able to deal with different values of MSS. The value of MSS will be supplied to Sender as an input argument.

8. Another input argument for Sender is Maximum Window Size (MWS). MWS is the maximum number of un-acknowledged bytes that the Sender can have at any time. MWS counts ONLY data. Header length should NOT be counted as part of MWS.

Remarks: Note that TCP does not explicitly define a maximum window size. In TCP, the maximum number of un-acknowledged bytes is limited by the smaller of receive window and the congestion control window. Since you will not be implementing flow or congestion control, you will be limiting the number of un-acknowledged bytes by using the MWS parameter. In other words, you will need to ensure that during the lifetime of the connection, the following condition is satisfied:

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{MWS}$$

9. Even though you will use UDP since the sender and receiver will mostly be running on machines that are within close proximity of each other (e.g.: on the same Ethernet LAN or even on the same physical machine), there will be no real possibility of datagrams being dropped. In order to test the reliability of your protocol, it is imperative to introduce artificially induced packet loss and delays. For this purpose you must also implement a Packet Loss and Delay (PLD) Module as part of the

Sender program. The details for this module are explained later in the specification.

Remarks: For the standard version of the assignment, the PLD module will only need to drop packets while for extended version, the PLD module will need to drop and delay packets. For simplicity, call both of them the PLD module, even though the PLD module for the standard version does not delay packets.

10. You must use a constant timeout in your program. The value of the timeout will be supplied to Sender as an input argument. Note that, this requirement applies to the standard version of the assignment. The extended version has a different requirement.

3.3 Features excluded

There are a number of transport layer features adopted by TCP that are excluded from this assignment:

1. You do not need to implement timeout estimation unless you want to attempt the extended version of the assignment.
2. You do not need to double timeout interval unless you want to attempt the extended version of the assignment.
3. You do not need to implement any flow nor congestion control.
4. STP does not have to deal with corrupted packets. Packets will very rarely be corrupted in our test topology, if at all. In short, it is safe for you to assume that packets are only lost.

3.4 Packet header and MSS

In designing the segment header, you only need to include the fields that you think are necessary for STP. You can draw inspiration from TCP but the exact format of the STP packet header is for you to decide. The header portion can include as many fields as you think are necessary. Two important fields that will be needed are the sequence number and acknowledgement number. You will also need a number of flags for connection establishment and teardown.

The data portion must not contain more than MSS bytes of data. You must use the same STP segment format for data transfer as well as for the acknowledgements flowing back from the receiver to the sender. The only difference will be that the acknowledgement segments will not contain any data. All information that is necessary for the proper functioning of your protocol must be provided in the STP headers. You should not use any information from the header of the UDP datagram that will encapsulate the STP packets (except for port number and IP address).

3.5 Sender

This section provides details on the Sender.

For the standard version of the assignment, the Sender should accept the following eight (8) arguments (note that the last two arguments are used exclusively by the PLD module):

1. `receiver_host_ip`: the IP address of the host machine on which the Receiver is running.
2. `receiver_port`: the port number on which Receiver is expecting to receive packets from the sender.
3. `file.txt`: the name of the text file that has to be transferred from sender to receiver using

your reliable transport protocol.

4. `MWS`: the maximum window size used by your STP protocol in bytes.
5. `MSS`: Maximum Segment Size which is the maximum amount of data (in bytes) carried in each STP segment.
6. `timeout`: the value of timeout in milliseconds.

The following two arguments are used exclusively by the PLD module:

7. `pdrop`: the probability that a STP data segment which is ready to be transmitted will be dropped. This value must be between 0 and 1. For example if `pdrop` = 0.5, it means that 50% of the transmitted packets are dropped by the PLD.
8. `seed`: The seed for your random number generator. The use of seed will be explained in Section 4.5.2 of the specification.

The Sender should be initiated as follows:

If you use Java:

```
java Sender receiver_host_ip receiver_port file.txt MWS MSS timeout pdrop seed
```

If you use C:

```
./sender receiver_host_ip receiver_port file.txt MWS MSS timeout pdrop seed
```

If you use Python:

```
python sender.py receiver_host_ip receiver_port file.txt MWS MSS timeout pdrop
seed
```

Note that, you should first start the Receiver before initiating the Sender.

3.5.1 The PLD Module

The PLD module should be implemented as part of your Sender program. The function of the PLD is to emulate some of the events that can occur in the Internet such as loss of packets and delays. Even though theoretically UDP packets will get lost and delayed on their own, in our test environment these events will occur very rarely. Further to test the reliability of your STP protocol we would like to be able to control the percentage of packets being lost. As mentioned before, the PLD module for the standard version of this assignment will only drop the packet.

The following describes the sequence of steps that the PLD should perform on receiving a STP segment:

1. If the STP segment is for connection establishment or teardown, then pass the segment to UDP, do not drop it.

   Remark: In order to reduce the complexity of connection setup, the connection establishment and teardown segments from the Sender can bypass the PLD module and will not be dropped.

2. If the STP segment is not for connection establishment or teardown, the PLD must do one of the following:

   (a) with probability `pdrop` drop the datagram.

   (b) With probability `(1-pdrop)`, forward the datagram.

   To implement this simply generate a random number between 0 and 1. If the chosen number is greater than `pdrop` transmit the packet, else the packet is dropped.

Remark: The file PingServer.java in Lab Exercise 2 contains an example of randomly dropping packets.

Once the PLD is ready to transmit a STP segment, the Sender should encapsulate the STP segment in a UDP datagram (i.e. create a UDP datagram with the STP segment as the payload). It should then transmit this datagram to the Receiver through the UDP socket created earlier. (Use the RECEIVER_HOST_IP and RECEIVER_PORT as the destination IP address and port number respectively). Once the entire text file has been transmitted reliably (i.e. the sender window is empty and the final ACK is received) the Sender can close the UDP socket and terminate the program.

Note that the ACK segments from the receiver must completely bypass the PLD modules. In other words, ACK segments are never lost.

3.5.2 Seed for random number generators

You will be asked to run your Sender and Receiver pair to show us that they are running correctly, see Section 8 of the specification for the experiments that you need to conduct. In order for us to check your results, we will be asking you to initialise your random number generator with a specific seed in Section 8 so that we can repeat your experiments.

Extra: If you have not learnt about the principles behind random number generators, you need to know that random numbers are in fact generated by a deterministic formula by a computer program. Therefore, strictly speaking, random number generators are called pseudo-random number generators because the numbers are not truly random. The deterministic formula for random number generation in Python, Java and C uses an input parameter called a *seed*. If the same seed is used, then the same sequence of random numbers will be produced.

The following code fragment in Python, Java and C will generate random numbers between 0 and 1 using a supplied seed.

1. In Python, you initialise a random number generator (assuming the seed is 50) by using `random.seed(50);`. After that you can generate a random floating point number between (0,1) by using `random.random();`

2. In Java, you initalise a random number generator (assuming the seed is 50) by using `Random random = new Random(50);`. After that, you can generate a random floating point number between (0,1) by using `float x = random.nextFloat();`

3. In C, you initalise a random number generator (assuming the seed is 50) by using `srand(50);`. After that, you can generate a random floating point number between (0,1) by using `float x = rand()/((float)(RAND_MAX)+1);` Note that, `RAND_MAX` is the maximum value returned by the `rand()` function.

You will find that if you specify different seeds, a different sequence of pseudo-random numbers will be produced.

3.5.3 Additional requirements for Sender

Your Sender will receive acknowledgements from the Receiver through the same socket, which the sender uses to transmit data. The Sender must first extract the STP acknowledgement from the UDP datagram that it receives and then process it as per the operation of your STP protocol. The format of the acknowledgement segments should be exactly the same as the data segments except that they should not contain any data. Note that these acknowledgements should bypass the PLD module.

The sender should maintain a log file titled `Sender_log.txt` where it records the information about each segment that it sends and receives. Information about dropped segments (and delayed segments in case of the extended assignment) packets should also be included. Start each entry on a new line. The format should be as follows:

```
<snd/rcv/drop> <time> <type of packet> <seq-number> <number-of-bytes> <ack-number>
```

where <type of packet> could be S (SYN), A (ACK), F (FIN) and D (Data)

So for example, the following shows the log file for a Sender that transmits 112 bytes of data. The MSS used here is 56 bytes and the timeout interval is 100msec. Notice that the second data packet is dropped and is hence retransmitted after a timeout interval of 100msec.

```
snd  34.335 S   121 0  0
rcv  34.4   SA  154 0  122
snd  34.54  A   122 0  155
snd  34.57  D   122 56 155
drop 34.67  D   178 56 155
rcv  36.56  A   155 0  178
snd  134.67 D   178 56 155
rcv  137.65 A   155 0  234
snd  138.76 F   234 0  155
rcv  140.23 FA  155 0  235
snd  141.11 A   235 0  156
```

Once the entire file has been transmitted reliably the Sender should initiate the connection closure process by sending a FIN segment (refer to Section 3.5.6 of the text). The Sender should also print the following statistics at the end of the log file (i.e. `Sender_log.txt`):

- Amount of (original) Data Transferred (in bytes)
- Number of Data Segments Sent (excluding retransmissions)
- Number of (all) Packets Dropped (by the PLD module)
- Number of (all) Packets Delayed (for the extended assignment only)
- Number of Retransmitted Segments
- Number of Duplicate Acknowledgements received

3.6 Receiver

The Receiver should accept the following two arguments:

1. `receiver_port`: the port number on which the Receiver will open a UDP socket for receiving datagrams from the Sender.
2. `file.txt`: the name of the text file into which the text sent by the sender should be stored (this is the file that is being transferred from sender to receiver).

The Receiver should be initiated as follows:

If you use Java:

```
java Receiver receiver_port file.txt
```

If you use C:

```
./receiver receiver_port file.txt
```

If you use Python:

```
python receiver.py receiver_port file.txt
```

Note that, you should first start the Receiver before initiating the Server.

The Receiver should generate an ACK immediately after receiving a data segment. This is the only ACK generation rule you need. You do **not** need to follow Table 3.2 of the text. In other words, you do no have to implement delayed ACKs. The format of the acknowledgement segment must be exactly similar to the STP data segment. It should however not contain any payload.

The receiver is expected to buffer out-of-order arrival packets.

The receiver should first open a UDP listening socket on `receiver_port` and then wait for segments to arrive from the Sender. The first segment to be sent by the Sender is a SYN segment and the receiver is expected to reply a SYNACK segment.

After the completion of the three-way handshake, the receiver should create a new text file called `file.txt`. All incoming data should be stored in this file. The Receiver should first extract the STP packet from the arriving UDP datagrams and then extract the data (i.e. payload) from the STP packet. Note that, the Receiver is allowed to examine the header of the UDP datagram that encapsulates the STP Packet to determine the UDP port and IP address that the Sender is using.

The data should be written into `file.txt`. At the end of the transfer, the Receiver should have a duplicate of the text file sent by the Sender. You can verify this by using the diff command on a Linux machine (`diff file1.txt file2.txt`).

The Receiver should also maintain a log file titled `Receiver_log.txt` where it records the information about each segment that it sends and receives. The format should be exactly similar to the sender log file as outlined in the Sender specification.

The Receiver should terminate after the connection closure procedure initiated by the sender concludes. The Receiver should also print the following statistics at the end of the log file (i.e. `Receiver_log.txt`):

- Amount of (original) Data Received (in bytes) – do not include retransmitted data
- Number of (original) Data Segments Received
- Number of duplicate segments received (if any)

3.7 Overall structure

The overall structure of your protocol will be similar to that shown in Figure 2. Note in particular that the PLD module is only required at the Sender.

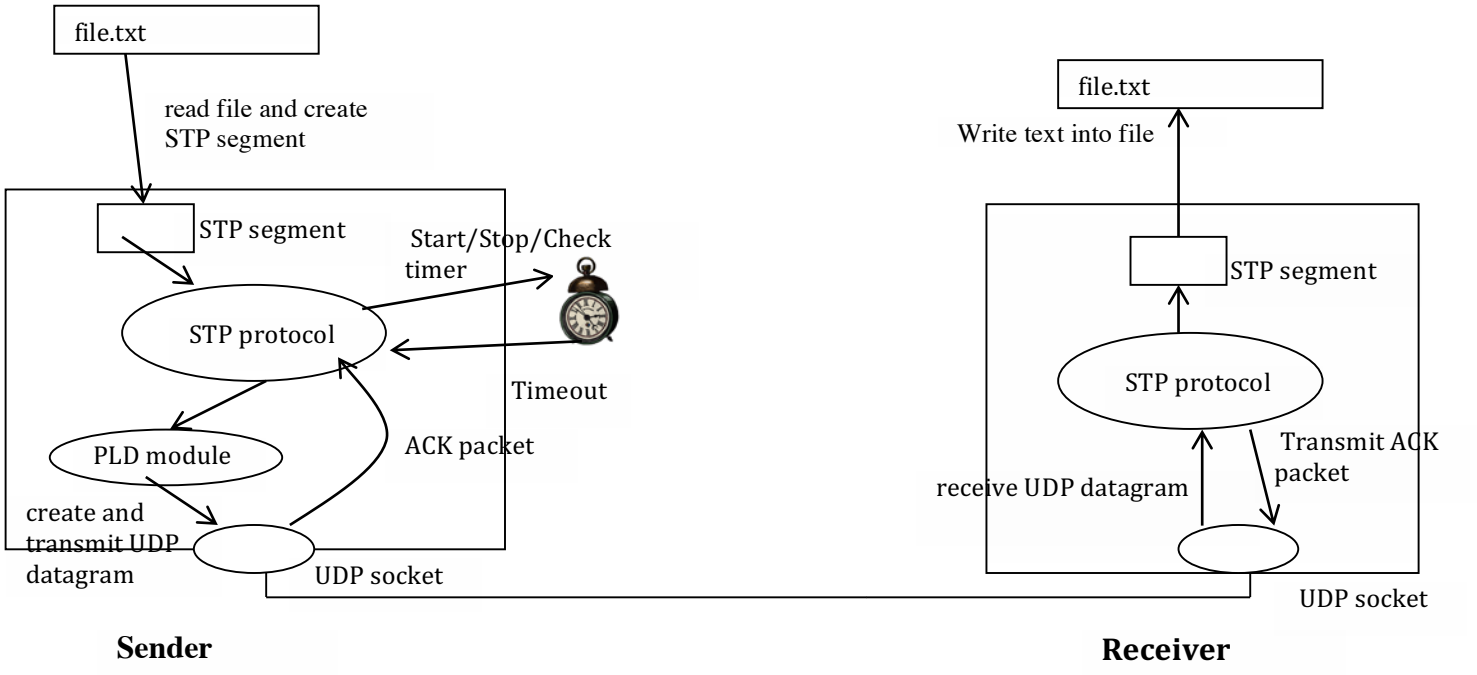

Figure 2: The overall structure of your assignment

4. Extended Version

The extended version of the assignment differs from the standard version in the following aspects:

1. You are required to implement round-trip-time estimation in Section 3.5.3 of the text.
2. The timeout is not a constant value but is given by the formula on page 271 of the text.
3. The PLD module will need to delay packets in addition to dropping packets. Specifically, the PLD module will take in two addition parameters `pdelay` and `MaxDelay`, in addition to `pdrop`. The PLD module will work as follows:
   (a) For each packet passing through the PLD module (with the exception of connection setup packets), the packet is dropped with probability `pdrop`, or is not dropped with probability `1 - pdrop`.
   (b) For those packets that are not dropped, there is a probability of `pdelay` that the packet will be delayed by anywhere between 0 to `MaxDelay` seconds. In other words, out of all the packets that are not dropped, a portion of them (specified by `pdelay`) will be delayed and the amount of the delay that is experienced by the packet is in the interval `[0, MaxDelay]` with a uniform distribution.

4.1 Extended Version of Sender

For the extended version of the assignment, the Sender should accept the following ten (10) arguments (note that the last four arguments are used exclusively by the PLD module):

1. `receiver_host_ip`: the IP address of the host machine on which the Receiver is running.
2. `receiver_port`: the port number on which Receiver is expecting to receive packets from the sender.
3. `file.txt`: the name of the text file that has to be transferred from sender to receiver using

your reliable transport protocol.

4. `MWS`: the maximum window size used by your STP protocol in bytes.
5. `MSS`: Maximum Segment Size which is the maximum amount of data (in bytes) carried in each STP segment.
6. `gamma`: See Section 8 of the specification for the meaning of this parameter.

The following four arguments are used exclusively by the PLD module:

7. `pdrop`: the probability that a data segment which is ready to be transmitted will be dropped. This value must be between 0 and 1. For example if `pdrop` = 0.5, it means that 50% of the transmitted packets are dropped by the PLD.
8. `pdelay`: the probability that a data segment which is not dropped will be delayed. This value must also be between 0 and 1.
9. `MaxDelay`: The maximum delay (in milliseconds) experienced by those data segments that are delayed.
10. `seed`: The seed for your random number generator. The use of seed was explained in Section 4.5.2 of the specification.

5. Additional Notes

- This is NOT group assignment. You are expected to work on this individually.
- **Tips on getting started**: The best way to tackle a complex implementation task is to do it in stages. A good starting point is to implement the file transfer using the simpler alternating-bit (stop-and-wait) protocol (version rdt3.0 from the textbook). First, make sure that your program works without implementing the PLD module. Next, implement the packet drop functionality of the PLD and test your protocol. Once you can verify that this works, extend your code to handle transmission of a window of packets (i.e., MWS). Send a window of packets and wait for all acknowledgements to come back before sending another window worth of data. As before, test the no loss case first. Then, extend your program to handle packet losses. Once you have the complete STP protocol implemented run comprehensive tests to ensure that your program works correctly.
- **Language and Platform**: You are free to use C, JAVA or Python to implement this assignment. Please choose a language that you are comfortable with. The programs will be tested on CSE Linux machines. So please make sure that your entire application runs correctly on these machines (i.e. your lab computers). ***This is especially important if you plan to develop and test the programs on your personal computers (which may possibly use a different OS or version or IDE).*** Note that CSE machines support the following: **gcc version 4.9.2, Java 1.7, Python 2.7, 2.8 and 3. If you are using Python, please clearly mention in your report which version of Python we should use to test your code.** You may only use the basic socket programming APIs providing in your programming language of choice. You may not use any special ready-to-use libraries or APIs that implement certain functions of the spec for you.
- You are free to design your own format and data structure for the messages. Just make sure your program handles these messages appropriately.
- You are encouraged to use the designated OpenLearning page for assignment 1 to ask questions and to discuss different approaches to solve the problem. However, you should **not** post your solution or any code fragments on the forum.

[redacted]

[redacted]

7. Report

As mentioned earlier, you should submit a small report, `report.pdf` (no more than 5 pages), plus appendix section (appendix need not be counted in the 5 page limit). Your report must contain the following:

1. A brief discussion of how you have implemented the STP protocol. Provide a list of features that you have successfully implemented. In case you have not been able to get certain features of STP working, you should also mention that in your report.
2. A detailed diagram of your STP header and a quick explanation of all fields (similar to the diagrams that we have used in the lectures to understand TCP/UDP headers).
3. For the *standard version of the assignment*, answer the following questions:

   (a) Use the following parameter setting: `pdrop` = 0.1, MWS = 500 bytes, MSS = 50 bytes, `seed` = 300. Explain how you determine a suitable value for timeout. Note that you may need to experiment with different timeout values to answer this question. Justify your answer.

   With the timeout value that you have selected, run an experiment with your STP programs transferring the file test1.txt (available on the assignment webpage). Show the sequence of STP packets that are observed at the receiver. It is sufficient to just indicate the sequence numbers of the STP packets that have arrived. You do not have to indicate the payload contained in the STP packets (i.e. the text). Run an additional experiment with `pdrop` = 0.3, transferring the same file (test1.txt). In your report, discuss the resulting packet sequences of both experiments indicating where dropping occurred. Also, in the appendix section show the packet sequences of all the experiments.

   (b) Let `Tcurrent` represent the timeout value that you have chosen in part (a). Set `pdrop` = 0.1, MWS = 500 bytes, MSS = 50 bytes, `seed` = 300 and run three experiments with the following different timeout values:

   `i. Tcurrent`

   `ii. 4 × Tcurrent`

   `iii. Tcurrent/4`

   and transfer the file test2.txt (available on the assignment webpage) using STP. Show a table that indicates how many STP packets were transmitted (this should include retransmissions) in total and how long the overall transfer took. Discuss the results.

4. For the *extended version of the assignment*, answer the following questions:

   (a) Run your protocol using `pdrop` = 0.1, MWS = 500 bytes, MSS = 50 bytes, `seed` = 100, `pdelay` = 0, `MaxDelay` = 800 (note: since `pdelay` = 0, the value of MaxDelay is not important), `gamma` = 4, and transfer the file test1.txt (available on the assignment webpage). Show the sequence of STP packets that are observed at the receiver. It is sufficient to just indicate the sequence numbers of the STP packets that have arrived. You

do not have to indicate the payload contained in the STP packets (i.e. the text). Run an additional experiment with `pdrop` = 0.3, transferring the same file (test1.txt). In your report, discuss the resulting packet sequences of both experiments indicating where dropping occurred. Also, in the appendix section show the packet sequences of all the experiments.

(b) The timeout for TCP is given by the following formula:

```
TimeoutInterval = EstimatedRTT + 4 DevRTT
```

Instead of using a fixed multiplier of 4, you will use a parameter gamma to study the effect of this multiplier on the performance. In the extended version, the timeout is given by:

```
TimeoutInterval = EstimatedRTT + gamma DevRTT
```

where `gamma` will be supplied to the program as an input argument, see Section 5.1.

Set `pdrop` = 0.5, MWS = 500 bytes, MSS = 50 bytes, `seed` = 300, `pdelay` = 0.2, `MaxDelay` = 1000 and run three experiments with the following different `gamma` values:

i. `gamma = 2`

ii. `gamma = 4`

iii. `gamma = 9`

and transfer the file test2.txt using STP. Show a table that indicates how many STP packets were transmitted (this should include retransmissions) in total and how long the overall transfer took. Discuss the results.

[redacted]

[redacted]

9. Marking Policy

You should test your program rigorously before submitting your code. Your code will be marked using the following criteria:

1. We will first run STP with the drop probability set to zero and the window size set to 1 MSS. This should result in a simple stop-and-wait protocol. We will transfer a sample text file (similar to those on the assignment webpage) using STP. We will then test if the stop-and-wait version can tolerate packet loss, by varying the drop probability. In all tests we will also check your log files to verify the functioning of the PLD module and the reliability of STP.
2. Successful operation of the STP protocol. This will involve a number of tests as indicated below:
    (a) Initially we will set the drop probability (`pdrop`) for the PLD to zero, and transfer a file using STP.

    (b) We will then test how reliable your protocol is by gradually increasing the drop probability (`pdrop`). Your protocol should be able to deal with lost packets successfully.

    (c) In the above tests we will also check the log files created by your Sender, Receiver to verify the functioning of your programs.

    (d) We will thoroughly test the operation for a wide range of parameters: `MWS`, `pdrop`, `timeout`, etc.

3. Your report, which includes a description of how you implemented the programs and the answers to the prescribed questions (as described in Section 8 of the specification).

Note that, we will verify that the description in your report conforms with the actual implementations in the programs. We will also verify the experiments that you have run for answering the questions. If we find that your report does not conform to the programs that you have submitted you will NOT receive any marks for your report.

For the extended version, we will conduct additional tests to see how your program can cope with different `pdelay` and `MaxDelay`. Note that it is possible that packets are re-ordered for the extended version of the assignment. We will see how your program deals with the re-ordering of packets. We will also verify your timeout estimation by using various values of `pdelay` and `MaxDelay`. The maximum bonus mark that you can get for the extended assignment is **2 marks**.