COMP3331/9331 Computer Networks and Applications
Assignment 2 for Session 1, 2012

Version 1
Due: 11:59pm Friday, 1 June 2012 (Week 13)

Updates to the assignment, including any corrections and clarifications, will be posted on the subject website. Please make sure that you check the subject website regularly for updates.

1 Goal and learning objectives

For this assignment, you will be asked to implement a reliable transport protocol over the UDP protocol. We will refer to the reliable transport protocol that you will be programming in this assignment as My Transport Protocol (MTP). MTP 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 (5th 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 multi-media delivery services in the Internet, where they have implemented their own proprietary transport protocol over UDP.

Note that it is mandatory that you implement MTP over UDP. Do not use TCP sockets. You will not receive any mark for this assignment if you use TCP socket.

1.1 Learning Objectives

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

1. Detailed understanding of how reliable transport protocols function
2. Socket programming for UDP transport protocol
3. Protocol and message design

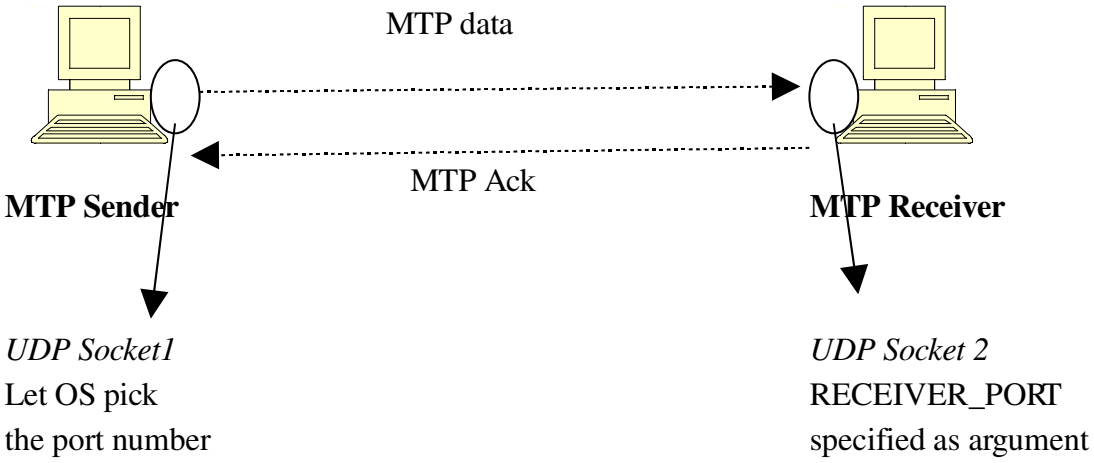

Figure 1: The basic set-up of your assignment. A file is to be transferred from the sender to the receiver. MTP_Sender will run on the sender side while MTP_Receiver will run on the receiver side. Note that data will flow from the sender to receiver, while ACK packets will flow from the receiver to sender.

2 Overview

As part of this assignment, you will have to implement My Transport Protocol (MTP), a piece of software that consists of a sender and receiver component that allows reliable unidirectional data transfer. MTP includes some of the features of the TCP protocols that are described in sections 3.5.4 and 3.5.6 of the textbook (5th edition). You will use your MTP protocol to transfer simple text files (provided on the assignment webpage) from the sender to the receiver. You should implement MTP as two separate programs: MTP_Sender and MTP_Receiver. You only have to implement unidirectional transfer of data from the sender to the receiver. As illustrated in Figure 1, data will flow from MTP_Sender to MTP_Receiver while ACK packets will flow from MTP_Receiver to MTP_Sender. Let us re-iterate this, MTP 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. A scanned version of section 3.5 of the text is available from the assignment website. Since this is copyright protected materials, you can only access it with your CSE account name and password.

3 Assignment specifications

This section gives detailed specifications of the assignment. There are two versions of this assignment, a standard version and an extended version. The specifications for the extended version can be found in Section 4.

3.1 File names

The main code for the sender and receiver should be contained in the following files: mtp_sender.c, mtp_sender.cc or mtp_sender.java, and mtp_receiver.c, mtp_receiver.cc or mtp_receiver.java.

3.2 List of features provided by the MTP_Sender and MTP_Receiver

You are required to implement the following features in the MTP_Sender and MTP_Receiver:

1. A threeway handshake for the connection establishment (Section 3.5.6 of the text)
2. MTP_Sender must maintain a single-timer for timeout operation (Section 3.5.4 of the text)
3. MTP_Sender should implement all the features mentioned in Section 3.5.4 of the text, with the exception of doubling timeout. The MTP protocol must include the simplified TCP sender (Figure 3.33 of the text) and fast retransmit (page 285). You will need to use a number of concepts that we have discussed in class, e.g., sequence numbers, cumulative acknowledgements, timers, send buffers, etc for implementing your protocol.
4. MTP_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.
5. MTP is a byte-stream oriented protocol. You will need to include sequence number and acknowledgement number fields in the MTP header. The meaning of sequence number and acknowledgment number are the same as TCP.
6. MTP_Sender must be able to deal with different maximum segment size (MSS). The value of MSS will be supplied to MTP_Sender as an input argument.
7. An input argument for MTP_Sender is Maximum Window Size (MWS). MWS is the maximum number of un-acknowledged bytes that the MTP_Sender can have at any time.

   Remarks: Note that TCP does not explicitly define a maximum window size. In TCP, the maximum number of un-acknowledeged 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:

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

8. 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), 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 MTP_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, I have chosen to call both of them the PLD module though the PLD module for the standard version does not delay packets.

9. You can use a constant timeout in your program. The value of the timeout will be supplied to MTP_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 features that are **excluded** from this assignment:

1. You do not need to implement time-out estimation unless you want to attempt the extended version of the assignment.
2. You do not need to double time-out 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. Your MTP should not worry about packets being corrupted. Packets will very rarely be corrupted, if at all. Further since you are using UDP, it will provide elementary error control by virtue of its checksum mechanism. In short, it is safe for you to assume that packets are only lost.

3.4 Packet header and MSS

In designing the packet header, you only need to include the fields that you need for MTP. The exact format of the MTP packet header is for you to decide. The header portion can contain 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.

The data portion must not contain more than MSS bytes of data. You must use the same MTP packet 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 packets will not contain any data. All information that is necessary for the proper functioning of your protocol must be provided in the MTP headers. You should not use any information from the header of the UDP datagram that will encapsulate the MTP packets.

3.5 MTP_Sender

This section provides details on the MTP_Sender.

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

1. RECEIVER_HOST_IP: the IP address of the host machine on which the MTP_Receiver is running.
2. RECEIVER_PORT: the port number on which MTP_Receiver is expecting a packet 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 MTP protocol in bytes
5. MSS: Maximum Segment Size which is the maximum amount of data (in bytes) carried in each MTP segment.
6. `timeout`: the value of timeout in milli-seconds

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

7. `pdrop`: the probability that a MTP packet 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 3.5.2.

3.5.1 The PLD Module

The PLD module should be implemented as part of your MTP_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 MTP 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 MTP segment

1. If the MTP segment is for connection establishment, pass segment to UDP, do not drop it.

Remarks: In order to reduce the complexity of connection setup, the connection establishment segments from the MTP_Sender can by-pass the PLD module and will not be dropped.

2. If the MTP segment is not for connection establishment, 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.
   Remarks: The file PingServer.java for the lab in Week 4 contains an example of randomly dropping packets.

Once the PLD is ready to transmit a MTP segment, the MTP_Sender should encapsulate the MTP segment in a UDP datagram (i.e. create a UDP datagram with the MTP segment as the payload). It should then transmit this datagram to the MTP_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 MTP_Sender can close the UDP socket and terminate the program.

3.5.2 Seed for random number generators

You will be asked to run your MTP_Sender and MTP_Receiver pair to show us that they are running correctly, see Section 7 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 7 so that we can repeat your experiments.

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 both 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 Java and C will generate random numbers between 0 and 1 using a supplied seed.

1. 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();`

2. 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);`

There is also a sample C program `rand_seed_demo.c` on the assignment web page which generates a sequence of random numbers for the seed specified in the input argument. You will find that if you specify different seeds, a different sequence of random numbers will be produced.

3.5.3 Additional requirements for MTP_Sender

The first packet that is sent by the MTP_Sender is the SYN packet. This is used for connection setup.

The MTP_Sender should also maintain a log file titled mtp_sender_log.txt in which it should record all events along with the time at which they occur. The format for these logs should be: Time, Event, Packet Details where Time is your local system time (at an appropriate resolution), Event is some event that occurred at that time such as transmitted packet with sequence number x, dropped packet with sequence number y, received an ACK with sequence number z, etc. The Packet Details should contain the details of the corresponding MTP packet such as the headers and the data portion (if it is a data packet). Do not record any UDP headers. Please start each entry on a new line. The exact format of how this is represented is for you to decide. A good place within the program to record this information is immediately following the occurrence of each event.

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

Once the entire file has been transmitted reliably the MTP_Sender should close the UDP socket.

3.6 MTP_Receiver

The MTP_Receiver should accept the following two arguments:

1. RECEIVER_PORT: the port number on which the MTP_Receiver will open a UDP socket for receiving datagrams from the MTP_Receiver.

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 MTP_Receiver should generate an ACK immediately after receiving a packet. This is the only ACK generation rule you need. You do **not** need to follow Table 3.2 of the text.

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 simply wait for datagrams to arrive from the MTP_Sender. The first packet to be sent by the MTP_Sender is a SYN segment and the receiver is expected to reply a SYNACK segment.

The receiver should then create a new text file called file.txt. All incoming data should be stored in this file. The MTP_Receiver should first extract the MTP packet from the arriving UDP datagrams and then extract the data from the MTP packet. At the end of the transfer, the MTP_Receiver should have a duplicate of the text file sent by the MTP_Sender.

The receiver should also maintain a log file titled mtp_receiver_log.txt in which it should record all events along with the time at which they occur. The format for these logs should be: Time, Event, Packet Details where Time is your local system time (at an appropriate resolution), Event is some event that occurred at that time such as received packet with sequence number `x`, transmitted an ACK with sequence number `z`, etc. The Packet Details should contain the details of the corresponding MTP packet such as the headers and the data portion (if it is a data packet). Do not record any UDP headers. Please start each entry on a new line. The exact format of how this is represented is for you to decide. A good place within the program to record this information is immediately following the arrival of the MTP packet and just prior to the transmission of an acknowledgement packet.

Your MTP_Receiver must send back acknowledgements to the MTP_Sender as per the functioning of your MTP protocol. The format of the acknowledgement packet must be exactly similar to the MTP data packet. It should however not contain any payload.

Since the receiver is not sure about when the last packet will arrive, do not close the UDP socket. The receiver program can only be terminated by typing CTRL-C.

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 MTP_Sender.

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 p.279 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

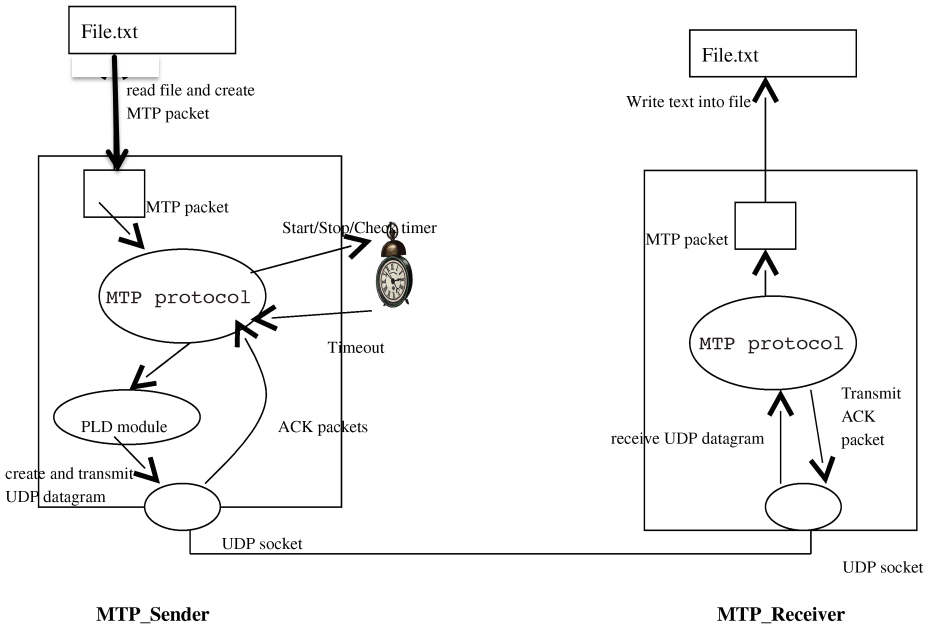

Figure 2: The overall structure of your assignment.

`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 MTP_Sender

For the extended version of the assignment, the MTP_Sender should accept the following ten (10) arguments (note that the last 3 argument are used exclusively by the PLD module):

1. RECEIVER_HOST_IP: the IP address of the host machine on which the MTP_Receiver is running.
2. RECEIVER_PORT: the port number on which MTP_Receiver is expecting a packet 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 MTP protocol in bytes
5. MSS: Maximum Segment Size which is the maximum amount of data (in bytes) carried in each MTP segment.

6. `gamma`: See Section 7 for the meaning of this parameter

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

7. `pdrop`: the probability that a MTP packet 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 an incoming packet in the forward direction (data only not ack) 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 packets that are delayed.

10. `seed`: The seed for random number generation.

5 Additional Notes

- This is not a group assignment. You are expected to work on this individually.
- How to Start: Sample client and server programs have been uploaded to the Assignment webpage. The textbook contains code for a simple UDP and TCP client server application, which is a good place to start (Section 2.8).
- Language and Platform: You are free to use either C or JAVA 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).
- 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 course discussion forum to ask questions and to discuss different approaches to solve the problem. However, you should **not** post your solution nor any code fragment on the forum.

6 Assignment Submission

Please ensure that you use the mandated file name. You may of course have additional header files and/or helper files. If you are using C, then you MUST submit a makefile/script along with your code (not necessary with Java). In addition you should submit a small report, report.pdf (no more than 3 pages) describing the program design, a brief description of how your system works

and your message design. Also discuss any design tradeoffs considered and made. Describe possible improvements and extensions to your program and indicate how you could realise them. If your program does not work under any particular circumstances please report this here. Also indicate any segments of code that you have borrowed from the Web or other books.

You are required to submit your source code and report.pdf. You can submit your assignment using the give command in an xterm from any CSE machine. Make sure you're in the same directory as your code and report, then do the following:

1. Type tar -cvf assign.tar filenames
   e.g. tar -cvf assign.tar *.java report.pdf

2. When you're ready to submit, at the bash prompt type 3331

3. Next, type: give cs3331 assignment2 assign.tar (You should receive a message stating the result of your submission).

Important notes

- The system will only accept assign.tar submission name. All other names will be rejected.

- Ensure that your program/s are tested in CSE Unix machine before submission. In the past, there were cases where students had problems in compiling and running their program during the actual demo. To avoid any disruption, please ensure that you test your program in CSE Unix-based machine before submitting the assignment.

You can submit as many time before the deadline. A later submission will override the earlier submission, so make sure you submit the correct file. Do not leave until the last moment to submit, as there may be technical or communications error and you will not have time to rectify it.

Late Submission Penalty: Late penalty will be applied as follows:

- 1 day after deadline: 10% reduction

- 2 days after deadline: 20% reduction

- 3 days after deadline: 30% reduction

- 4 days after deadline: 40% reduction

- 5 or more days late: NOT accepted

NOTE: The above penalty is applied to your final total. For example, if you submit your assignment 1 day late and your score on the assignment is 30, then your final mark will be 30 3 (10% penalty) = 27.

7 Report

In addition you should submit a small report, report.pdf (no more than 5 pages), plus appendix section. Your report must contain the following

1. A brief discussion of how you have implemented the MTP protocol. Provide a list of features that you have successfully implemented. In case you have not been able to get certain features of MTP working, you should also mention that in your report.

2. A detailed figure of your MTP header and a quick explanation of all fields.

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 MTP programs transferring the file test1.txt (available on the assignment webpage). Show the sequence of MTP packets that are observed at the receiver. It is sufficient to just indicate the sequence numbers of the MTP packets that have arrived. You do not have to indicate the payload contained in the MTP 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 using MTP. Show a table that indicates how many MTP 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 MTP packets that are observed at the receiver. It is sufficient to just indicate the sequence numbers of the MTP packets

that have arrived. You do not have to indicate the payload contained in the MTP 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 4.1. 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 MTP. Show a table that indicates how many MTP packets were transmitted (this should include retransmissions) in total and how long the overall transfer took. Discuss the results.

8 Plagiarism

You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Do not post this assignment on forums where you can pay programmers to write code for you. We will be monitoring such forums. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or

a URL when the code appears (as comments). Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

9 Marking Policy

You should test your program rigorously before submitting your code. For the standard version, your code will be marked using the following criteria:

1. Successful compilation of all code source files: 1 mark
2. We will first run MTP 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 the ones on the assignment webpage) using MTP. 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 MTP. (14 marks)
3. Successful operation of the MTP protocol. This will involve a number of tests as indicated below: (20 marks)
   (a) Initially we will set the drop probability for the PLD to zero, and transfer a file using MTP.
   (b) We will then test how reliable your protocol is by gradually increasing the drop probability. 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 MTP_Sender, MTP_Receiver to verify the functioning of your programs.
   (d) We will thoroughly test the operation for a wide range of parameters: window size, drop probability, `timeout`, etc.
4. Your report, which includes a description of how you implemented the programs and the answers to the prescribed questions (as described in the submission section): 10 marks. Note that, we will verify that the description in your report confirms 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.

We would of course like you to complete the entire assignment. You should implement as much as you possibly can.

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 6 marks.