COMP3331/9331 Computer Networks and Applications

Assignment for Term 2, 2021

Version 1.0

Due: 11:59am (noon) Friday, 6 August 2021 (Week 10)

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. Change Log

Version 1.0 released on 21st June 2021.

2. 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 Padawan Transport Protocol (PTP). PTP 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 numbers, 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 on the Internet, where they have implemented their own proprietary transport protocol over UDP.

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

2.1 Learning Objectives

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

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

Non-CSE Student: The rationale for this option is that students enrolled in a program that does not include a computer science component have had very limited exposure to programming and in particular working on complex programming assignments. A Non-CSE student is a student who is not enrolled in a CSE program (single or double degree). Examples would include students enrolled **exclusively** in a single degree program such as Mechatronics or Aerospace or Actuarial Studies or Law. Students enrolled in dual degree programs that include a CSE program as one of the degrees do not qualify. Any student who meets this criterion and wishes to avail of this option MUST email cs3331@cse.unsw.edu.au to seek approval before **5pm**, **2nd July (Friday, Week 5)**. We will assume by default that all students are attempting the CSE version of the assignment unless they have sought explicit permission. **No exceptions**.

3. Assignment Specification

As part of this assignment, you will have to implement Padawan Transport Protocol (PTP), a piece of software that consists of a sender and receiver component that allows reliable **unidirectional** data transfer. PTP includes some of the features of the TCP protocol that are described in sections 3.5.4 and 3.5.6 of the textbook (7th edition). You will use your PTP protocol to transfer simple text (ASCII) files (examples provided on the assignment webpage) from the sender to the receiver. You should implement PTP 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, **PTP 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.

NOTE: Section 3.5 of the textbook which covers the bulk of the discussion on TCP is available to download on the assignment page.

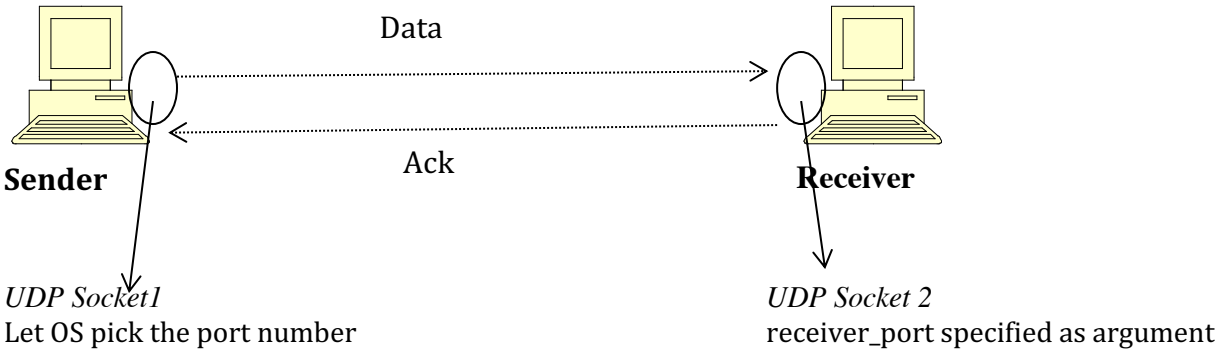

Figure 1: The basic setup of your assignment. A file is to be transferred from the Sender to the Receiver. Sender will run on the sender side while Receiver will run on the receiver side. Note that data segments will flow from the sender to receiver, while ACK segments will flow from the receiver to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

3.1 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 of the text 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. It is possible for the Receiver to combine the ACK and FIN in one message. See Section 3.5.6 of the text for further details. The Sender should terminate after connection closure.

3. Sender must maintain a single timer for timeout operation (Section 3.5.4 of the text). You must use a constant timeout in your program. The value of the timeout will be supplied to Sender as an input argument.

4. Sender should implement all the features mentioned in Section 3.5.4 of the text, with the exception of doubling the timeout. The PTP 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 (of text) for ACK generation. All segments should be immediately acknowledged, i.e., you do not have to implement delayed ACKs.

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

7. One of the command line arguments, MSS (Maximum segment size) is the maximum number of bytes of data that your PTP 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. You may safely assume that the MSS will be smaller than the maximum possible size of a UDP segment (64Kbytes).

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:

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

10. Even though you will use UDP since the sender and receiver will mostly be running on the same physical machine, there will be no real possibility of PTP segments being dropped. In order to test the reliability of your protocol, it is imperative to introduce artificially induced packet loss. For this purpose, you must also implement a Packet Loss (PL) Module as part of the Sender program. The details for this module are explained later in the specification.

4.3 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 use a random initial sequence number.
2. You do not need to implement timeout estimation. The timer value is provided as a command line argument.
3. You do not need to double timeout interval.
4. You do not need to implement any flow nor congestion control.
5. PTP does not have to deal with corrupted packets. Packets will rarely be corrupted when the sender and receiver are executing on the same machine. In short, it is safe for you to assume

that packets are only lost.

6. You do not need to handle abnormal behaviour, i.e., Sender or Receiver program crashing. In other words, you do not need to implement functionality like RST in TCP.

4.4 Packet header and MSS

In designing the segment header, you only need to include the fields that you think are necessary for PTP. You can draw inspiration from TCP but the exact format of the PTP 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 PTP 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 PTP headers. You may use the port number and IP address included within the UDP datagram that encapsulates the PTP segments.

4.5 Sender

This section provides details on the Sender.

The Sender should accept the following eight (8) arguments (note that the last two arguments are used exclusively by the PL 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. This should match the command line argument of the same name for the Receiver.
3. `FileToSend.txt`: the name of the text file that has to be transferred from sender to receiver using your reliable transport protocol. You may assume that the file included in the argument will be available in the current working directory of the Sender with the correct access permissions set (read).
4. `MWS`: the maximum window size used by your PTP protocol in bytes.
5. `MSS`: Maximum Segment Size which is the maximum amount of data (in bytes) carried in each PTP segment. NOTE: In our tests we will ensure that MWS is exactly divisible by MSS.
6. `timeout`: the value of timeout in milliseconds.

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

7. `pdrop`: the probability that a PTP 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 PL.
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 FileToSend.txt MWS MSS timeout pdrop seed
```

If you use C:

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

If you use Python:

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

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

It is very likely that you will be executing the Sender and Receiver on the same machine. In this case use 127.0.0.1 (localhost) for the `receiver_host_ip`.

4.5.1 The PL Module

The PL module should be implemented as part of your Sender program. The function of the PL is to emulate packet loss on the Internet. Even though theoretically UDP datagrams will get lost, in our test environment these events will occur very rarely. Further to test the reliability of your PTP protocol we would like to be able to control the percentage of packets being lost. You can assume that packets will not be delayed or corrupted in the network.

The following describes the sequence of steps that the PL should perform on receiving a PTP segment:

1. If the PTP 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 PL module and will not be dropped.

2. If the PTP segment is not for connection establishment or teardown, the PL module 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 PL is ready to transmit a PTP segment, the Sender should encapsulate the PTP segment in a UDP datagram (i.e., create a UDP datagram with the PTP 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.

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

4.5.2 Seed for random number generators

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 of the spec 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 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.

4.5.3 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 PTP acknowledgement from the UDP datagram that it receives and then process it as per the operation of your PTP 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 PL 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. You may assume that the sender program will have permission to create files in its current working directory. Information about dropped segments 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) and the fields should be tab separated.

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   34.40      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 PL module)
- Number of Retransmitted Segments
- Number of Duplicate Acknowledgements received

NOTE: Generation of this log file is very important. It will help your tutors in understanding the flow of your implementation and marking. So, if your code does not generate any log files, you will only be graded out of 25% of the marks.

The Sender should not print any output to the terminal. If you are printing output to the terminal for debugging purposes, make sure you disable it prior to submission.

4.6 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. `FileReceived.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 FileReceived.txt
```

If you use C:

```
./receiver receiver_port FileReceived.txt
```

If you use Python:

```
python receiver.py receiver_port FileReceived.txt
```

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

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 must not implement delayed ACKs. The format of the acknowledgement segment must be exactly similar to the PTP data segment. It should however not contain any payload.

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

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 `FileReceived.txt`. You may assume that the receiver program will have permission to create

files in its current working directory. All incoming data should be stored in this file. The Receiver should first extract the PTP segment from the arriving UDP datagrams and then extract the data (i.e., payload) from the PTP segment. Note that, the Receiver should examine the header of the UDP datagram that encapsulates the PTP segment to determine the UDP port and IP address that the Sender is using. This information is needed to send the ACK segment to the Sender. The ACK should be encapsulated in an UDP datagram and sent to the Sender.

The data should be written into `FileReceived.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 FileReceived.txt FileToSend.txt`). When testing your program, if you have the Sender and Receiver executing in the same working directory then make sure that the file name provided as the argument to the Receiver is different from the file name used by the sender.

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 – tab separated fields.

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)

NOTE: Generation of this log file is very important. It will help your tutors in understanding the flow of your implementation and marking. So, if your code does not generate any log files, you will only be graded out of 25% of the marks.

The Receiver should not print any output to the terminal. If you are printing output to the terminal for debugging purposes, make sure you disable it prior to submission.

4.7 Overall structure

The overall structure of your protocol is depicted in Figure 2. The PL module is only at the Sender.

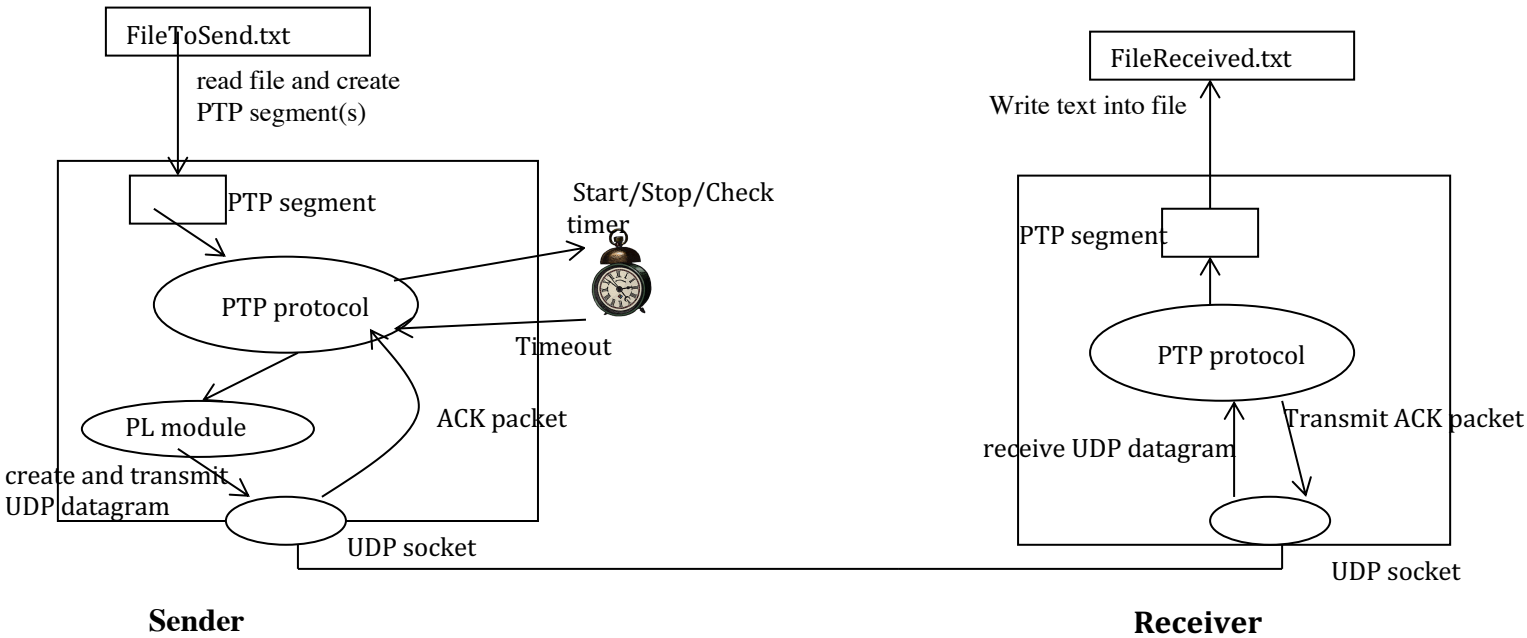

Figure 2: The overall structure of your assignment

~~Receiver Design~~

The Receiver program logic is straightforward. Upon receive of a UDP segment through the socket, the Receiver should extract the PTP segment which is encapsulated within the UDP segment. It should then execute the PTP protocol logic (outlined in Section 4.6 of the spec). This includes the connection setup, data transmission and finally connection teardown. During the data transmission phase an ACK segment should be generated for each received PTP segment. Each PTP segment sent by the Receiver must be encapsulated in a UDP datagram and sent to the Sender. The Receiver may have to buffer the PTP segment (if out of order) or else write the data contained in the PTP segment to the file.

To summarise, the key steps are:

1. Connection setup
2. Data Transmission (repeat until end of file)
    a. Receive PTP segment
    b. Send ACK segment
    c. Buffer data or write data into file
3. Connection teardown

Sender Design

The Sender program logic is a little more complicated. The Sender must first execute connection setup, followed by data transmission and finally correction teardown. During data transmission, the Sender should transmit a number of PTP segments (based on the MWS and MSS), all of which need to be buffered (in case of retransmissions) and wait for the corresponding ACKs. A timer should be started for the oldest unacknowledged segment. Each data segment should also be passed through the PL module which determines if the segment should be dropped or forwarded. Each PTP segment to be transmitted must be encapsulated in a UDP datagram and sent to the Receiver. The Sender should also process incoming ACK segments from the Receiver. In the case of a timeout, the Sender should transmit the oldest unacknowledged segment. Given the complexity and the need to deal with multiple events, there are two options you may consider for the design of the Sender: (i) using multiple threads to manage the various events (ii) non-blocking or asynchronous I/O by using polling, i.e., select().

To summarise, the key steps are:

1. Connection setup
2. Data Transmission (repeat until end of file)
    a. Read file
    b. Create PTP segment
    c. Start Timer if required (retransmit oldest unacknowledged segment on expiry)
    d. Send PTP segment to PL module
    e. If PTP segment is not dropped, transmit to Receiver
    f. Process ACK if received
3. Connection teardown

6. 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 stop-and-wait protocol (version rdt3.0 from the textbook and lectures). First, make sure that your program works without implementing the PL module. Next, implement the packet drop functionality of the PL 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) – i.e. the sliding window protocol as used by

TCP. As before, test the no loss case first. Then, extend your program to handle packet losses. Once you have the complete PTP protocol implemented run comprehensive tests to ensure that your program works correctly. It is imperative that you rigorously test your code to ensure that all possible (and logical) interactions can be correctly executed. **Test, test and test**.

- You are free to design your own format and data structure for the messages. Just make sure your program handles these messages appropriately.
- **Backup and Versioning:** We strongly recommend you to back-up your programs frequently. CSE backups all user accounts nightly. If you are developing code on your personal machine, it is strongly recommended that you undertake daily backups. We also recommend using a good versioning system so that you can roll back and recover from any inadvertent changes. There are many services available for this, which are easy to use. We will NOT entertain any requests for special consideration due to issues related to computer failure, lost files, etc.
- **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) or using VLAB. 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 8.2, Java 11, Python 2.7 and 3.7. 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 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 or any code fragments on the forum.
- We will arrange for additional consultations in Weeks 7-10 to assist you with assignment related questions. Information about the consults will be announced via the website.

7. 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 or Python). This is because we need to know how to resolve the dependencies among all the files that you have provided. After running your makefile we should have the following executable files: `sender` and `receiver`. In addition, you should submit a small report, `report.pdf` (no more than 5 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. Further details about what should be included in the report are provided in Section 8.

You are required to submit your source code and report.pdf. You can submit your assignment using the give command through VLAB. Make sure you are in the same directory as your code and report, and then do the following:

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

2. When you are ready to submit, at the bash prompt type `3331`

3. Next, type: `give cs3331 assign assign.tar` (You should receive a message stating the result of your submission). The same command should be used for 3331 and 9331.

Alternately, you can also submit the tar file via the WebCMS3 interface on the assignment page.

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 the VLAB environment before submission. In the past, there were cases where tutors were unable to compile and run students' programs while marking. To avoid any disruption, please ensure that you test your program in the VLAB environment before submitting the assignment. Note that, we will be unable to award any significant marks if the submitted code does not run during marking.**

- You may submit as many times as you wish 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 network errors 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 10, then your final mark will be 10 – 1 (10% penalty) = 9.

8. Report

In addition, 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 PTP protocol. Provide a list of features that you have successfully implemented. In case you have not been able to get certain features of PTP working, you should also mention that in your report.
2. A detailed diagram of your PTP 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. 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 PTP programs

transferring the file test1.txt (available on the assignment webpage). Show the sequence of PTP packets that are observed at the receiver. It is sufficient to just indicate the sequence numbers of the PTP packets that have arrived. You do not have to indicate the payload contained in the PTP 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 PTP. Show a table that indicates how many PTP packets were transmitted (this should include retransmissions) in total and how long the overall transfer took. Discuss the results.

9. 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.

10. 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 PTP 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 PTP. 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 PL module and the reliability of PTP. (**5 marks**)
2. Successful operation of the PTP protocol. This will involve a number of tests as indicated below: (**12 marks**)

(a) Initially we will set the drop probability (`pdrop`) for the PL to zero, and transfer a file using PTP.

(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): **3 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.

NOTE: If your Sender and Receiver do not generate the log files as indicated earlier in the specification, you will only be graded out of 25% of the total marks (i.e., a 75% penalty will be assessed).