The University of New South Wales

COMP3331/9331 Computer Networks and Applications

Assignment for Session 2, 2018

Version 1.0

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

1. Change Log

Version 1.0 released on 17th August 2018.

See the changes marked in Red color.

2. Due date:

Due: 11:59pm Friday, 19th October 2018 (Week 12).

Early bird incentive: 10% bonus marks if the assignment is submitted before 11:59 pm Friday, 12th October 2018 (Week 11). Note that the final marks of assignment component are capped at maximum 20. E.g., if you get 19 marks and have submitted as an early bird, your marks are increased (and capped to) 20 marks.

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 Simple Transport Protocol (STP). STP will include most (but not all) of the features that are described in Sections 3.5.3, 3.5.4 and 3.5.6 of the text Computer Networking (7th edition). 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.

Note that it is mandatory that you implement STP over UDP. Do not use TCP sockets. You will not

receive any mark for this assignment if you use TCP socket.

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.

3. 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 3.5.3, 3.5.4 and 3.5.6 of the textbook (7th edition). You will use your STP protocol to transfer pdf files 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 3.5.3, 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.

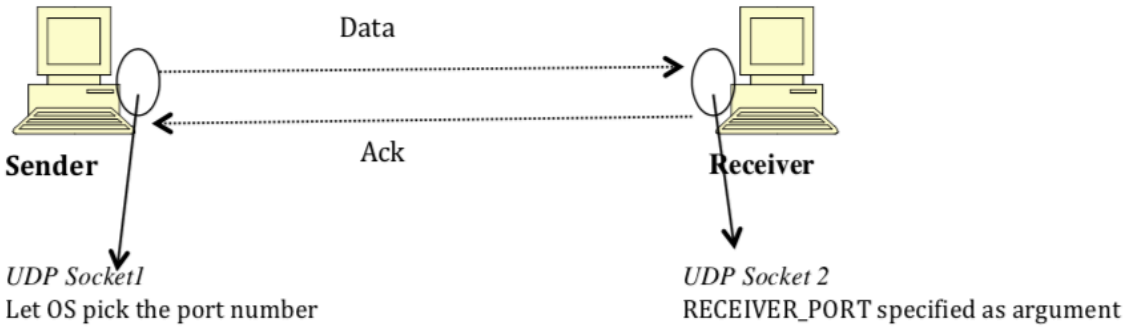

Figure 1: The basic setup of your assignment. A file is to be transferred from the Sender to the Receiver. Sender will run on the sender side while Receiver will run on the receiver side. Note that data segments will flow from the sender to receiver, while ACK segments will flow from the receiver to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

4. Assignment Specifications

This section gives detailed specifications of the assignment. Total marks for this assignment are 20. You

are free to choose C, JAVA or Python as the programming language (please see Section 5). **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). We are unable to mark your assignment if it does not compile or run correctly on CSE lab computers, resulting in loss of significant marks**.

4.1 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.

4.2 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 text for further details.

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

3. Sender must maintain a single-timer for timeout operation. You are required to implement round-trip-time estimation and RTO estimation discussed in Section 3.5.3 of the text. The timeout is not a constant value but is given by the formula on page 243 of the text (TimeoutInterval = EstimatedRTT + 4 * DevRTT). **Use the initial value of EstimatedRTT = 500 milliseconds and DevRTT = 250 milliseconds**.

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 249-251). 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 in 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/delayed/corrupted. In order to test the reliability of your protocol, it is imperative to introduce artificially induced packet loss, delays and corruption etc. 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.

4.3 Features excluded

You do not need to implement any flow nor congestion control features for this assignment.

4.4 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 segments.

4.5 Sender

This section provides details on the Sender.

The Sender should accept the following fourteen (14) arguments (note that the last eight 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.pdf`: The name of the pdf file that has to be transferred from sender to receiver using your STP.

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`: This value is used for calculation of timeout value. See Section 7 of the specification for details.

The following 8 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 segments are dropped by the PLD.

8. `pDuplicate`: The probability that a data segment which is not dropped will be duplicated. This value must also be between 0 and 1.

9. `pCorrupt`: The probability that a data segment which is not dropped/duplicated will be corrupted. This value must also be between 0 and 1.

10. `pOrder:` The probability that a data segment which is not dropped, duplicated and corrupted will be re-ordered. This value must also be between 0 and 1.

11. `maxOrder:` The maximum number of packets a particular packet is held back for re-ordering purpose. This value must be between 1 and 6.

12. `pDelay`: The probability that a data segment which is not dropped, duplicated, corrupted or re-ordered will be delayed. This value must also be between 0 and 1.

13. `maxDelay`: The maximum delay (in milliseconds) experienced by those data segments that are delayed.

14. `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.pdf MWS MSS gamma pDrop pDuplicate pCorrupt pOrder maxOrder pDelay maxDelay seed
```

If you use C:

```
sender receiver_host_ip receiver_port file.pdf MWS MSS gamma pDrop pDuplicate pCorrupt pOrder maxOrder pDelay maxDelay seed
```

If you use Python:

```
python sender.py receiver_host_ip receiver_port file.pdf MWS MSS gamma pDrop pDuplicate pCorrupt pOrder maxOrder pDelay maxDelay seed
```

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

4.5.1 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, packet corruption, packet re-ordering 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, corrupted, re-ordered and delayed. Your PLD module should take care of the following events;

- Drop packets
- Duplicate packets
- Create bit errors within packets (a single bit error)
- Transmits out of order packets
- Delays packets

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 without going through PLD.

Remark: In order to reduce the complexity of connection setup, the connection establishment and teardown segments from the Sender can bypass the PLD module.

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 segment. To implement this simply generate a random number between 0 and 1. If the chosen number is less than `pDrop`, drop the STP segment.

   (b) If the segment is not dropped, with probability `pDuplicate`, forward the STP segment twice back-to-back to UDP.

   (c) If the packet is not dropped or duplicated, with probability `pCorrupt`, introduce one bit error (you can simply flip any one bit of data) and forward the STP segment to UDP.

   (d) If the packet is not dropped, duplicated or corrupted, with probability `pOrder` save the current STP segment and wait for forwarding of `maxOrder` segments to UDP before forwarding the saved STP segment to UDP. If there is a segment already waiting for re-ordering, forward the new STP segment without any delay i.e., there should be only one segment waiting for re-ordering at a time.

   (e) If the STP segment is not dropped, duplicated, corrupted or re-ordered, with probability `pDelay` the segment is to be delayed by anywhere between 0 to `MaxDelay` milliseconds before forwarding to UDP. In other words, the amount of the delay that is experienced by the segment is in the interval `[0, MaxDelay]` with a uniform distribution.

   (f) If the STP segment is not dropped, duplicated, corrupted, re-ordered or delayed, forward the STP segment to UDP.

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 file has been transmitted reliably (i.e. the sender window is empty and the final ACK of STP closure 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, there is no PLD module on the receiver. ACK segments are thus never dropped, duplicated, corrupted, re-ordered or delayed.

4.5.2 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.

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.

- 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();`
- 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();`
- 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 (it is using a UDP socket). 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 ACKs should bypass the PLD module.

Recall that the sender has a single-timer for timeout operation which measures timeout for the oldest segment in the window. The sender is allowed to have other timers in PLD e.g, to cater for the pDelay parameter. Additionally, a segment being delayed due to pDelay would not have any effect on the following packets. For example, if segment 10 is to be delayed by 100 msec, segments following it (11 onwards) continues getting processed, segment 10 is processed again by PLD (transmitted to UDP without getting any further error) when the 100 msec has passed. In other words, pDelay can also cause re-ordering.

Sender will not measure the sampleRTT (for maintaining its timer for timeout) for any segment that it re-transmits.

If a segment is marked for reordering due to `pOrder`, the PLD checks if there is already a segment waiting in queue to be re-ordered, if so, it sends the new segment immediately to UDP.

If a segment is dropped by PLD, the sender consider that it has transmitted that segment (and loss has occurred out in the network) and update the total bytes sent and total segments dropped.

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, delayed, corrupted segments should also be included. Start each entry on a new line. The format should be as follows:

`<event> <time> <type-of-packet> <seq-number> <number-of-bytes-data> <ack-number>`

where <event> = `snd/rcv/drop/corr/dup/rord/dely/DA/RXT or a combination.`

`corr` = corrupted, `dup` = duplicated, `rord`=re-ordered, `dely`= delayed, `DA`=duplicate Acks received and `RXT`= retransmission

`<type-of-packet>` could be S (SYN), A (ACK), F (FIN) and D (Data) or a combination of these. DA represent duplicate Acks received and RXT is for retransmissions.

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`):

- Size of the file (in Bytes)
- Segments transmitted (including drop & RXT)
- Number of Segments handled by PLD
- Number of Segments Dropped
- Number of Segments Corrupted
- Number of Segments Re-ordered
- Number of Segments Duplicated
- Number of Segments Delayed
- Number of Retransmissions due to timeout
- Number of Fast Retransmissions
- Number of Duplicate Acknowledgements received

A sample Sender_log.txt file is available under the Assignment link that shows the log file for a simple case where the Sender transmits a pdf file (test0.pdf available under Assignment link) of length 3028 bytes of data. The values of various parameters used are MSS=150 bytes, MWS=600, gamma=4, pDrop=0.1 and seed =100.

Pl note that generation of these log files 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 max 5 marks.

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. `file_r.pdf`: the name of the pdf file into which the data sent by the sender should be stored (this is a copy of 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_r.pdf
```

If you use C:

```
receiver receiver_port file_r.pdf
```

If you use Python:

```
python receiver.py receiver_port file_r.pdf
```

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 do not 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 packets that arrive out-of-order.

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 with a SYNACK segment.

After the completion of the three-way handshake, the receiver should create a new file called `file_r.pdf`. All incoming correct data (in order) should be stored in this file. At the end of the transfer,

the Receiver should have a duplicate of the pdf file sent by the Sender.

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 that the Sender is using.

Any segment that is found corrupted is discarded at the receiver without generating a duplicate Ack. Reason being the destination IP/destination port/source IP/source port may have been corrupted resulting in data/Ack being delivered to the wrong host or process.

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 Data Received (bytes)
- Total segments received
- Data segments received
- Data Segments with bit errors
- Duplicate data segments received
- Duplicate Acks sent

Pl note that generation of these log files are 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 max 5 marks.

4.7 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.

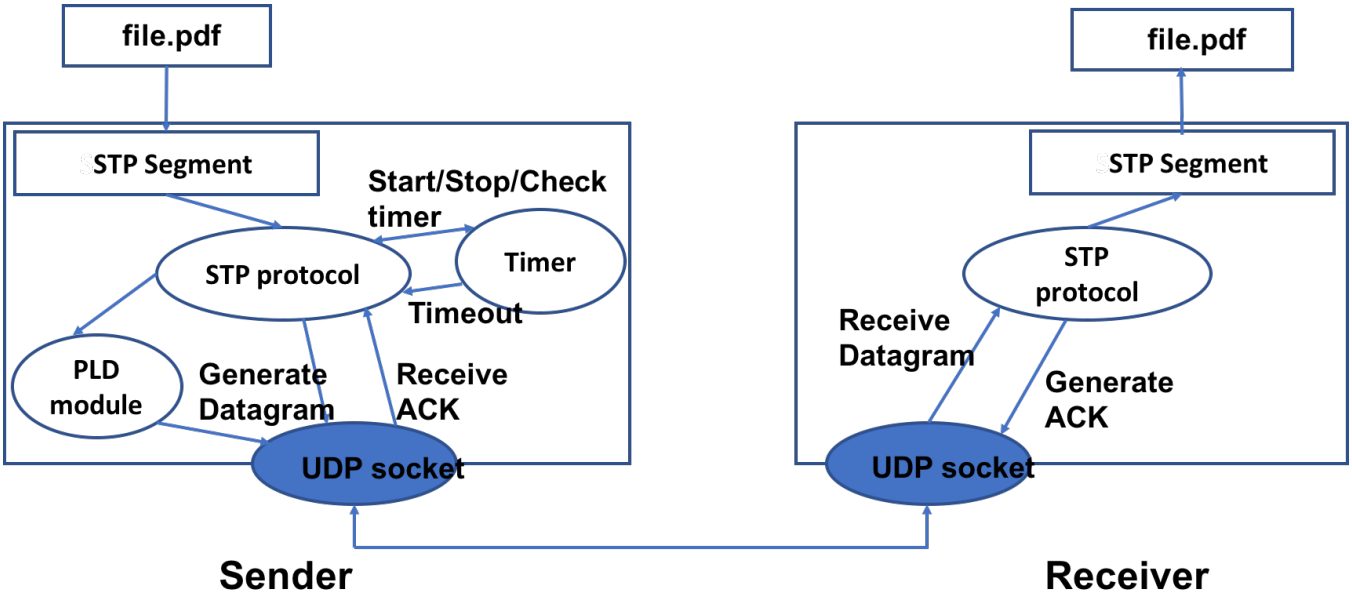

Figure 2: The overall structure of your assignment

5. Additional Notes

- This is NOT a 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. You can now incrementally introduce other type of errors. 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 other errors. 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). We are unable to mark your assignment if it does not compile or run correctly on CSE lab computers, resulting in loss of significant marks**. 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 specifications for you.

- We will arrange special tutorial sessions for this assignment to assist you with understanding of these specifications and get you started with socket programming in the language of your choice (C/Java/Python). More details and the schedule will be provided on the course website.
- 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 or any code fragments 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 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. Beside these source code files, you should submit a small report, report.pdf (no more than 5 pages) see details in Section 7.

You can submit your assignment using the give command in an xterm from any CSE machine. 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). Use cs3331 even if you are enrolled in 9331.

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 Linux machine 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 CSE Linux-based machine 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 can submit as many times 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: 15% reduction
- 2 days after deadline: 30% reduction
- 3 days after deadline: 45% reduction
- 4 days after deadline: 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 – 1.5 (15% penalty) = 8.5.

7. Report

You have to submit a small report, report.pdf (no more than 5 pages) that 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. Discuss any design trade-offs considered and made. Describe possible improvements and extensions to your program and indicate how you could realise them.
4. Indicate any segments of code that you have borrowed from the Web or other books.
5. Answer the following questions: (include any output as an appendix to the main report.pdf, appendix is not included in the 5-page limit)

(a) Run your protocol using pDrop = 0.1, MWS = 500 bytes, MSS = 100 bytes, seed = 100, gamma = 4, and pDuplicate, pCorrupt, pOrder, MaxOrder, pDelay, MaxDelay all set to 0. Transfer the file **test0.pdf** (available on the assignment webpage). The file should be received correctly at the Receiver. 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. Run an additional experiment with pdrop = 0.3, transferring the same file (**test0.pdf**). In your report, discuss the resulting packet sequences of both experiments indicating where dropping occurred. Also, in the appendix section show the packet sequences for both the experiments.

(b) The timeout for STP is given by:

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

where `gamma` will be supplied to the program as an input argument, see Section 4.5.

Set pdrop = 0.5, MWS = 500 bytes, MSS = 50 bytes, seed = 300, pdelay = 0.2, MaxDelay = 1000 and pDuplicate, pCorrupt, pOrder, MaxOrder all set to 0. Run three experiments with the following different gamma values:

i. gamma = 2
ii. gamma = 4
iii. gamma = ~~9~~ 6

and transfer the file **test1.pdf** using STP. Show a table that indicates how many STP packets were transmitted in total and how long the overall transfer took. Discuss the results.

(c) Use the following values and run STP to transfer **test2.pdf**.

```
MWS=500bytes MSS=50 gamma=4 pDrop=0.1 pDuplicate=0.1 pCorrupt=0.1
pOrder=0.1 maxOrder=4 pDelay=0 maxDelay=0 seed=300
```

Has the file been successfully transferred? How long the overall transfer took? For this experiment, which of the factor (out of `pDrop`, `pDuplicate`, `pCorrupt` and `pOrder`) is the most critical contributing most in the overall transfer time? How have you determined this? Provide the screen shot for the initial transfer (connection establishment + first 20 entries) and the last 20 entries plus the summary statistics table for the sender_log.txt and receiver_log.txt files in appendix. Do not attach the complete log files due to their sizes.

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 (not complete modules/functions) 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 where 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. Your code will be marked using the following criteria:

1. We will first run STP with all error probabilities 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 pdf file using STP. We will then test if the stop-and-wait version can tolerate packet loss, by varying the drop probability `pDrop`. In all tests, we will also check your log files to verify the functioning of the PLD module and the reliability of STP. (**5 marks**)
2. Successful operation of the *pipelined* STP protocol. This will involve a number of tests as indicated below: (**10 marks**)

(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.

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

(c) We will thoroughly test the operation for a wide range of parameters: `MWS, pDrop, pCorrupt, pDuplicate` etc.

(d) We will conduct additional tests to see how your program can cope with different `pDelay` (and `MaxDelay`) and `pOrder` (and `maxOrder`). 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 appropriate parameters.

3. Your report, which includes a description of how you implemented the programs and the answers to the prescribed questions (as described in Section 7 of the specification): (**5 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.