Initial Design Plan (UDP to TCP project)

**Objective**

The objective of this project is to build a reliable transfer protocol over the UDP transportation protocol. UDP is a transport protocol which does not provide reliable packet delivery. Unlike TCP it does not make use of functionalities such as acknowledgements, timeouts, delayed acks, sequence numbers etc. to ensure reliable packet delivery.

Hence the objective of this project is to implement various concepts of reliable packet delivery over UDP to understand in depth how TCP provides reliable packet delivery. It is also an exercise at socket programming, protocol and message design in java.

**Design**

Basic Functioning of protocol –

1. The protocol will be used to transfer simple text files.
2. It will be implemented as two separate programs – a sender and a receiver.
3. Only unidirectional flow of data will be implemented. Data segments will flow from Sender to Receiver while ACK segments will flow from Receiver to Sender.

Detailed design features -

1. A three-way handshake (SYN, SYN+ACK, ACK) for the connection establishment shall be implemented. The ACK sent by the sender to conclude the three-way handshake will notcontain any payload.

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. The Sender should terminate after connection closure.

3. Sender must maintain a single timer for timeout operation. The value of the timeout will be supplied to Sender as an input argument.

4. The PTP protocol must include the simplified TCP sender and implement fast retransmit.

5. All segments will be immediately acknowledged. The Receiver will not implement delayed acks.

6. There will be sequence numbers and acknowledgement numbers in the header of each segment.

7. One of the command line arguments, MSS (Maximum segment size) is the maximum number of bytes of data that the 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.

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.

Packet header and MSS

The packet header will have flags for SYN and FIN segments. Since, ACK flags are obvious in most cases they will not be implemented.

Sender

This section provides details on the Sender.

The Sender should accept the following eight (8) arguments (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
3. FileToSend.txt: the name of the text file that has to be transferred from sender to receiver using your reliable transport protocol. File is assumed to be in the working directory.
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 the segments. (MSS will be directly divisible by MWS).
6. timeout: the value of timeout in milliseconds.

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

1. 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.
2. 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

4

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.*

1. 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.

1. **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

7

page8image3904544

FileReceived.txt

page8image31427200

Write text into file

FileToSend.txt

read file and create PTP segment(s)

page8image31428544

PTP segment PTP protocol

PL module

Start/Stop/Check timer

PTP segment

page8image3899504page8image3899616page8image31430848

Timeout

PTP protocol  
Transmit ACK

page8image31431040

ACK packet

receive UDP datagram

packet

create and transmit UDP datagram

UDP socket

UDP socket

**Sender**

**Receiver**

**Figure 2:** The overall structure of your assignment

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.

**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)
   1. Receive PTP segment
   2. Send ACK segment
   3. 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)
   1. Read file
   2. Create PTP segment
   3. Start Timer if required (retransmit oldest unacknowledged segment on expiry)
   4. Send PTP segment to PL module
   5. If PTP segment is not dropped, transmit to Receiver
   6. Process ACK if received
3. Connection teardown