### Implementation of TCP-like reliable transport layer protocol using UDP

### Abhishek Dharmapurikar Ananth Mahadevan

Contents

[Problem Statement: 2](#_Toc326236648)

[Abstract: 2](#_Toc326236649)

[1. Architecture: 3](#_Toc326236650)

[2. Design (function calls and Data structures): 3](#_Toc326236651)

[2.1 Packet format: 3](#_Toc326236652)

[2.2 Library function calls: 4](#_Toc326236653)

[2.3 FTPC: 4](#_Toc326236654)

[2.4 TCPD-M2: 4](#_Toc326236655)

[2.5 FTPS: 4](#_Toc326236656)

[2.6 TCPD-M1: 4](#_Toc326236657)

[2.7 RTT and RTO calculation: 4](#_Toc326236658)

[2.8 Checksum Computation: 4](#_Toc326236659)

[2.9 Timer: 4](#_Toc326236660)

[2.10 Connection shutdown: 4](#_Toc326236661)

[3. Results: 4](#_Toc326236662)

[4. Project Management: 4](#_Toc326236663)

[5. Miscellaneous: 4](#_Toc326236664)

[6. Acknowledgements: 4](#_Toc326236665)

[7. References: 4](#_Toc326236666)

### Problem Statement:

The goal of the project is to implement a TCP-like reliable transport layer protocol using the unreliable service provided by UDP and transfer a file from one application to another. This involves using TCP’s function calls such as - ACCEPT, CONNECT, SEND, RECV – but implemented using UDP’s function calls.

### Abstract:

UDP is a simple, datagram-oriented, transport layer protocol and provides no reliability. It sends the datagrams to the IP layer, but there is no guarantee that they ever reach their destination. TCP on the other hand provides a connection-oriented, reliable, byte stream service. This means that two applications exchanging data have to first establish a TCP connection with each other before they can start exchanging data. The aim of this project is to transfer data from one machine to another reliably using TCP’s features but implemented using UDP’s system calls.

There are five main portions in this implementation. They are as given below:

1. FTPC – the client application that sends the file to the destination.
2. FTPS – the server application that receives the file sent by the source.
3. TCPD (M1/M2) – The TCP Daemon process is equivalent to the TCP stack in the OS and is responsible for the actual transfer of data. This involves storing the packets in a buffer, sending them via the Troll to the destination TCPD process, making sure that the packets sent are acknowledged before sending the next batch of packets (sliding-window) and communicating with the Timer.
4. TROLL - Any normal network is characterized by lot of cross-traffic and other network noise. This is simulated in the project by making use of the TROLL application.
5. Timer - The “Timer” process helps in making sure that the packets sent did reach their destination. This is done by means of measuring the round trip time (RTT) taken for the packet and storing that value along with the packet number in the Timer. The TCPD removes the value from the timer if the ACK for the packet comes back, else an interrupt is sent back to the TCPD which resends the packet.

### Architecture:

The pictorial representation of the overall architecture described in the previous section is shown below.

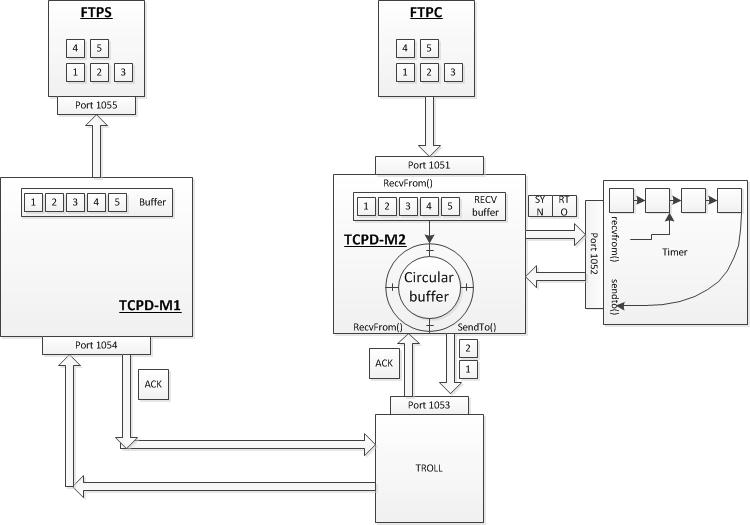


Figure : High level architecture

### Design (function calls and Data structures):

The design of this process involves making use of data structures such as Linked Lists and Arrays, which is described below.

### Packet format:

The format of the data packet sent is as follows:

1. The first 4 bytes contain the number of bytes in the file being transferred.
2. The next 20 bytes will contain the name of the file.
3. The rest of the bytes being transferred will contain the data in the file.
4. The end of transmission is indicated using a special set of characters.
   1. **Packet format between FTPC and TCPD\_M2:**

FTPC initially sends the name of the file and its size to TCPD\_M2. It then starts sending the file in 1000 bytes chunks.

* 1. **Packet format between TCPD\_M2 and Timer:**

TCPD\_M2 sends the Sequence number of the packet, the RTO value of the packet and a mode (Insert, Delete and End) to the timer.

* 1. **Packet format between TCPD\_M2 and TCPD\_M1:**

TCPD\_M2 sends the address of the destination (TCPD\_M1) and the data payload to the Troll.

* 1. **Packet format between TCPD\_M1 and FTPS:**

The packet format between TCPD\_M1 and FTPS is the same as that between FTPC and TCPD\_M2.

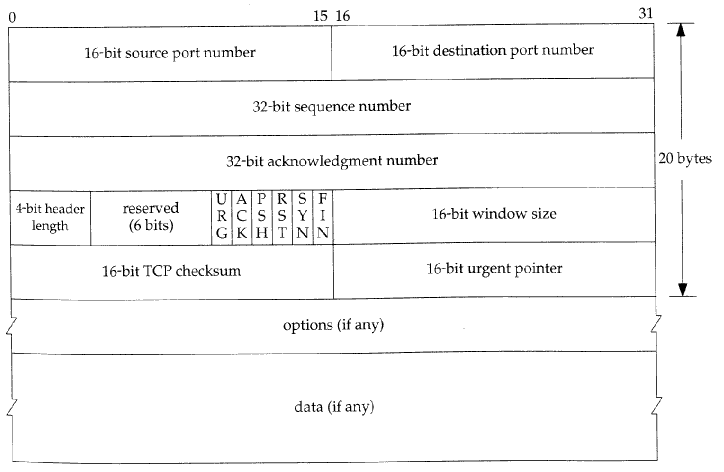
* 1. **TCP Header:**

The TCP packet format is as shown below. It consists of a header portion and the data portion. When this packet is sent to the IP layer, it is further packetized with IP’s header information. When this packet is received at the receiver side, the reverse process happens where the IP layer removes its header and sends its payload to TCPD-M1. TCPD-M1 sends the payload to the FTPS process.

|  |  |  |
| --- | --- | --- |
| IP Header | TCP Header | TCP Data |

TCP Segment

**IP datagram**



### Library function calls:

* 1. SOCKET()

Creates a socket connection.

* 1. BIND()

Binds the created socket to the address specified.

* 1. ACCEPT()

Implements a select() function call which is a blocking call. FTPS makes use of this function while waiting to reeive data from TCPD\_M1.

* 1. CONNECT()

Implements a sendto() function call. FTPC makes use of CONNECT() to send the IP and port number of TCPD\_M1 to TCPD\_M2.

* 1. SEND()

Implements a sendto() function call. FTPC makes use of SEND() to send data to TCPD\_M2.

* 1. RECV()

FTPS makes use of this function to receive the file payload from TCPD\_M1.

* 1. CLOSE()

This function closes the socket connections. It also sends the “end-of-file” message to TCPD\_M2 which then forwards it to TCPD\_M1 signaling the end of transmission.

### 2.3 FTPC:

This is the client application that splits the file to be transferred into equal parts and sends it to the TCPD-M2 process. Communication with the local TCPD process is via UDP sockets.

**Operations**:

1. Splits the file into packets of 1000-bytes each and send them to the FTPD-M2 process.
2. Makes function calls such as SOCKET(), BIND() and CONNECT(). Here CONNECT() is a null function, since there is no TCP handshaking.
3. Read bytes from the file and uses SEND() to send data to FTPS. SEND() is a blocking call and does not return until all bytes are written to the TCPD buffer.
4. The FTPC is started using the below command:

Ftpc <remote-IP> <remote-port> <local-file-to-transfer>

### 2.4 TCPD-M2:

This process has the implementation of the TCP stack and is roughly equivalent to the TCP in the OS.

**Operations:**

The operations of the TCP Daemon process in the client side are as given below:

* + Receive message from FTPC and store in wrap-around (Circular) buffer.
  + Calculate and keep updating the RTT and RTO values.
  + Calculate the CRC for a packet.
  + Manage the Wrap-around buffer.
  + Keep track of the ACKs for the packet and resend is required by communicating with the Timer.
  + Close the connections with the Timer and the FTPC process.



Figure : Flow of TCPD\_M2

1. **Initialize Circular-buffer implementation with sliding window**:

The circular buffer helps to keep track of the packets that were sent to the server. It can be implemented by making use of three pointers and a variable to keep track of the count of the values inserted into the buffer. The three pointers are:

1. Pointer to the actual buffer location in memory that gives the start of the buffer.
2. Pointer to the end of the buffer location in memory that gives the end of the buffer.
3. Pointer to insert data into the buffer, which moves along with the data inserted.

### 2.5 FTPS:

**Operations:**

* + Receive the file from TCPD-M1 and store it in a directory different from that of FTPC.
  + Makes the function calls SOCKET(), BIND() and ACCEPT().
  + Blocks until a connection is made to it from the client.

### 2.6 TCPD-M1:

**Operations:**

The operations of the TCP Daemon process in the server side are as given below:

* + Receive message from the TROLL and store packets in a buffer.
  + Calculate the CRC for the received packet.
  + Send the ACKs for the received packets.
  + After receiving all packets, send them to the FTPS process.
  + Close the connections with the TROLL and the FTPS process.

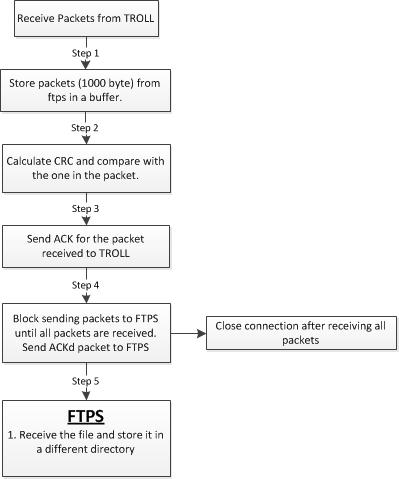


Figure : TCPD-M1

### 2.7 RTT and RTO calculation:

RTT is the time taken between the transmission of a packet and the reception of its ACK. It can be determined programmatically by sending a packet to a receiver, receiving the ACK and finding out the time it took for the round trip. The RTT and RTO can be calculated using the formulae given below (Jacobson’s implementation):

Err = M – A (M is the current RTT measurement; A is the smoothed RTT (average);

A = A +g Err (g is the gain for the average and is set to 0.125)

D = D + h (|Err| - D) (D is the smoothed deviation of the RTT values)

RTO = A + 4D

The steps followed before sending a packet is as follows:

1. Send a packet and receive the ACK to find the RTT.
2. Find the average of the RTT.
3. Compute the above values and the RTO.
4. Use these values as a method of congestion avoidance.

### 2.8 Checksum Computation:

In order to calculate the Checksum, a pseudo header was created which contains information taken from fields in both the TCP header and the IP datagram in which the TCP segment will be encapsulated. The checksum is calculated over all the octets of the pseudo header, TCP header and data. If the data contains an odd number of octets, a zero octet (pad) is added to the end of the data. The pseudo header and the pad are not transmitted with the packet. The pseudo header is as shown below:

|  |  |  |
| --- | --- | --- |
| **Field** | **Size** | **Description** |
| Source address | 4 | 32-bit IP address of the originator of the datagram (taken from IP header) |
| Destination address | 4 | 32-bit IP address of the recipient of the datagram (taken from IP header) |
| Reserved | 1 | 8 bits of zeros |
| Protocol | 1 | The protocol is TCP and this field will have the value 6 |
| TCP length | 2 | The length of the TCP segment including both header and data, which is computed. |

### **2.9 Timer**:

The Timer was implemented using a doubly-linked list which stores the sequence number of the packet and its corresponding timer value.

Seq #, time

Seq #, time

The Timer interacts with the TCPD\_M2 process using the following packet format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| sequenceNumber | Integer | 0 | Contains the sequence number of a particular packet. |
| time | Integer | 0 | Contains the RTT value of the packet. |
| flag | Integer | 0 | Indicates one out of 2 operations: Insert, Delete. |

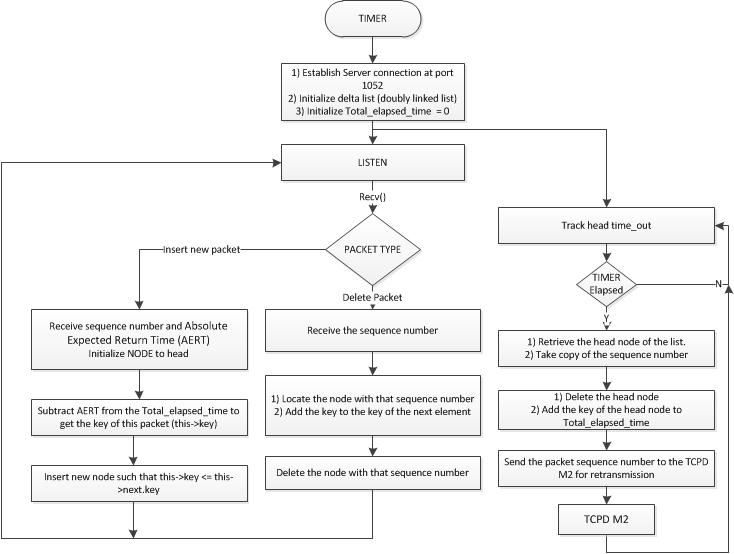


Figure : Timer

### 2.10 Connection shutdown:

The 4-way TCP connection teardown has been implemented in this project. When the “end-of-message” (##$$CONNECTEND$$##) is received from FTPC, TCPD\_M2 sends the FIN to TCPD\_M1 and enters into the FIN\_WAIT\_1 state. TCPD\_M1 sends the ACK for the FIN, but does not send a FIN back until its buffer is empty (meaning, it has received all data packets). After the buffer becomes empty, it sends a FIN and enters into the FIN\_WAIT\_1 state. It also starts a timer so that it can retransmit the FIN if the ACK from TCPD\_M2 does not come back. After the ACK comes back, TCPD\_M1 closes its connection while TCPD\_M2 enters into the TIME\_WAIT state where it waits for a period of 2MSL.

### Results:

We tested our implementation with different file types, different sizes and with different troll parameters, as shown below:

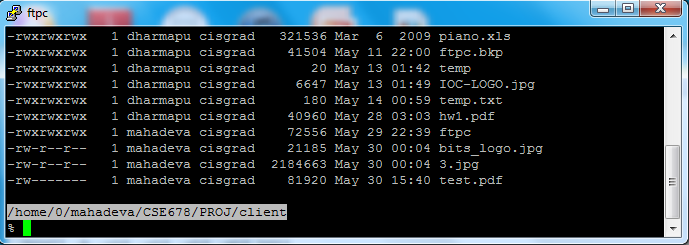
|  |  |  |  |
| --- | --- | --- | --- |
| File name | File size (kb) | Troll parameters | Result |
| piano.xls | 321.5 | Drop – 5, 10 and 25%  Delay – 5, 10 and 25%  Garble – 5, 10 and 25%  Duplicate – 5, 10 and 25% | PASS |
| IOC-LOGO.jpg | 6.6 | Drop – 5, 10 and 25%  Delay – 5, 10 and 25%  Garble – 5, 10 and 25%  Duplicate – 5, 10 and 25% | PASS |
| test.pdf | 82 | Drop – 5, 10 and 25%  Delay – 5, 10 and 25%  Garble – 5, 10 and 25%  Duplicate – 5, 10 and 25% | PASS |

**Application details:**

|  |  |  |
| --- | --- | --- |
| Application | Server | Port |
| Ftpc | 164.107.112.41 (mu) | - |
| TCPD-M2 (client) | 164.107.112.41 (mu) | 1051 |
| Timer | 164.107.112.41 (mu) | 1052 |
| Troll | 164.107.112.41 (mu) | 1053 |
| TCPD-M1 | 164.107.112.23 (Kappa) | 1054 |
| Ftps | 164.107.112.23 (Kappa) | - |

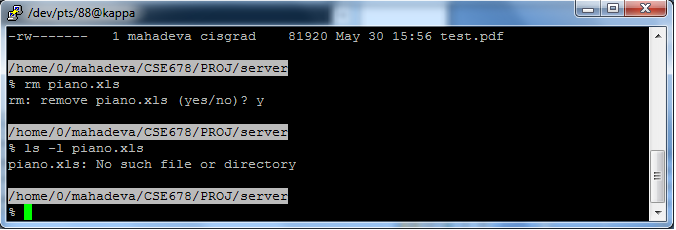
The screen-grabs of the applications are shown below:

1. List of test files.



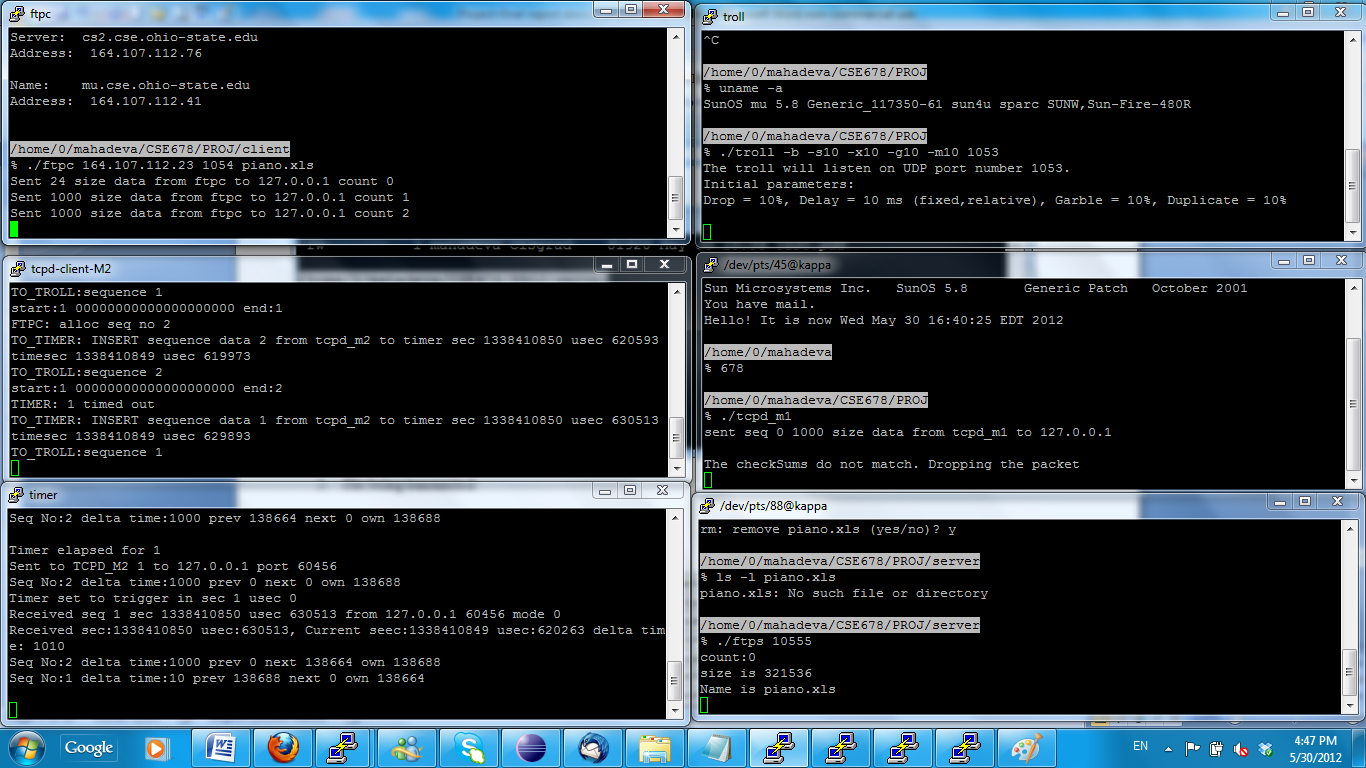
Figure

1. Absence of the file at the server.



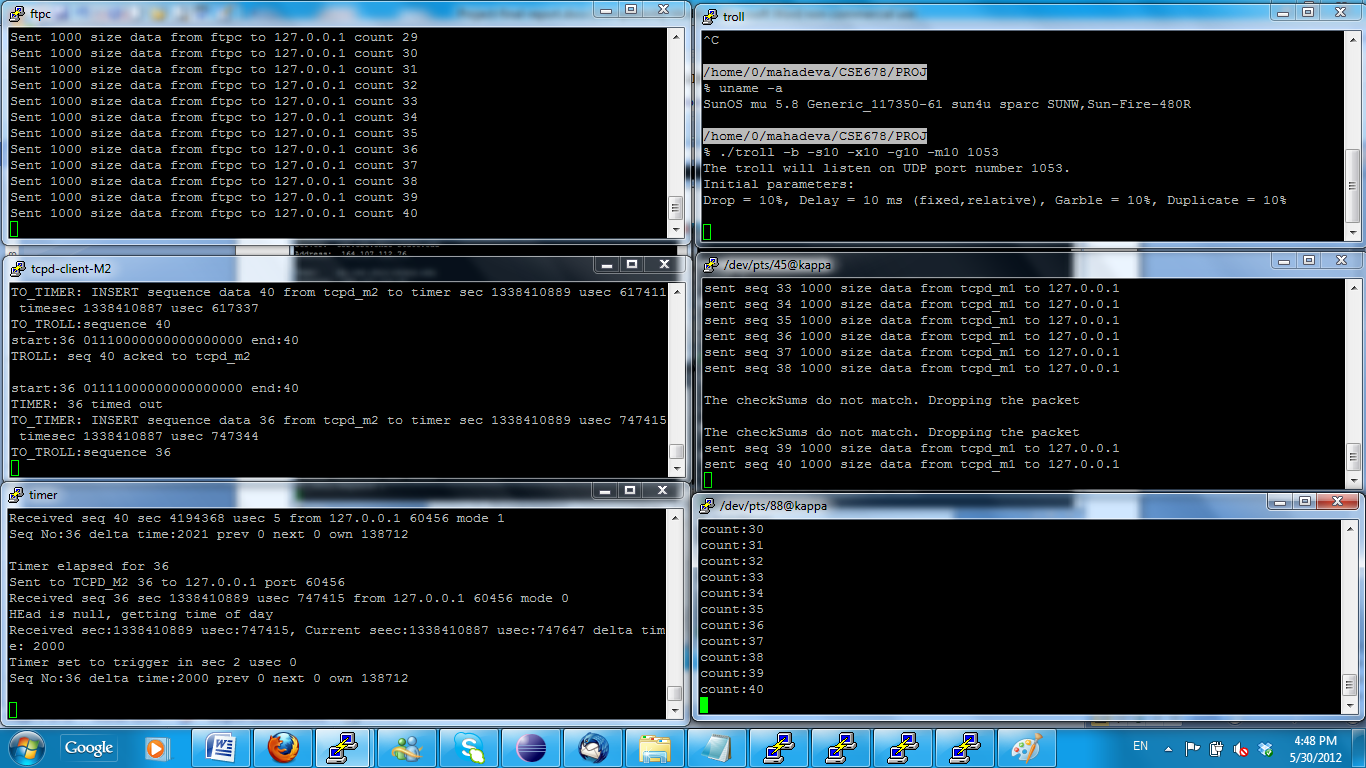
Figure

1. File being transmitted – Initial stage:



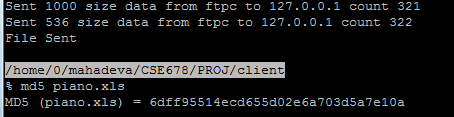
Figure

Intermediate stage:

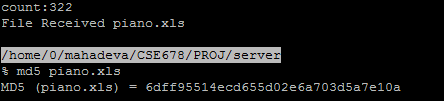


Figure

1. MD5 checksum of the file:



Figure



Figure

### Project Management:

**Task split:**

|  |  |  |
| --- | --- | --- |
| **Task** | **Responsibility** | |
| Project proposal | Ananth | Abhishek |
| Architecture design | Ananth | Abhishek |
| Coding – Timer, Sliding window |  | Abhishek |
| Coding – Circular Buffer, RTO, Checksum | Ananth |  |
| Testing | Ananth | Abhishek |
| Final report | Ananth | Abhishek |

### Miscellaneous:

##### 5.1 Unique features of our implementation:

##### 5.2 Plan vs Implementation:

There were quite a few changes between our plan and the actual implementation. Abhishek implemented the timer and the sliding window, while Ananth implemented the Circular buffer, RTT/RTO computation and the checksum computation. This is how we worked:

1. Created a common shared directory in the stdsun machine so that both of us could compile our code with the main code if required.
2. After implementing our individual components, we merged the code.
3. Both of us tested the entire project together so that when there were issues, we were able to exchange ideas while troubleshooting. This helped us clear quite a lot of confusion and enabled us to understand the concepts much better.

##### 5.3 Yet to implement:

We don’t have anything else to be implemented and have completed what was required of the project.

##### 5.4 Most complicated part:

The following were according us the most complicated part:

1. Timer:
   1. Implementation of the delta list took a long time as we had difficulty understanding how it actually works. We were initially not sure how situations when 2 timers expire at the same time would work. It was unclear as to what would happen when a timer expires and precisely at that moment the ACK for a packet arrives. Which would be picked up by select() – the timer call or the troll call.
2. Sliding window:
3. Maintaining the sequence numbers for the packets:
   1. We had issues where the packets that were not sent to TCPD-M1 were being acknowledged and the sliding window at TCPD-M2 started moving towards the right. It took us lot of hours to figure out the cause of the issue.

##### 5.5 Future plans and extensions:

1. We plan to implement a JPEG/MPEG encoder.
2. Steganography - To use this project in order to transfer and receive encrypted messages.
3. Implement the bit torrent protocol.

### Acknowledgements:

1. We wish to acknowledge our instructor Mr. Peter Dohm for his guidance during the project.
2. We also wish to acknowledge the known and unknown individuals whose code or postings in bulletin boards guided us when we were stuck while troubleshooting.

### References:

1. TCP/IP Illustrated, Volume 1 – The Protocols by W. Richard Stevens.
2. <https://en.wikipedia.org/wiki/Circular_buffer>
3. <https://en.wikipedia.org/wiki/Sliding_window_protocol>
4. <https://en.wikipedia.org/wiki/Round-trip_delay_time>
5. <https://en.wikipedia.org/wiki/Transmission_Control_Protocol>
6. <http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm>
7. <http://www.netfor2.com/tcpsum.htm>
8. <http://ssfnet.org/Exchange/tcp/tcpTutorialNotes.html#SW>
9. <http://akomaenablog.blogspot.com/2008/06/simple-senderreceiver-program-to.html>