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

### Abhishek Dharmapurikar Ananth Mahadevan

##### 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 six main portions in this implementation. They are given below:

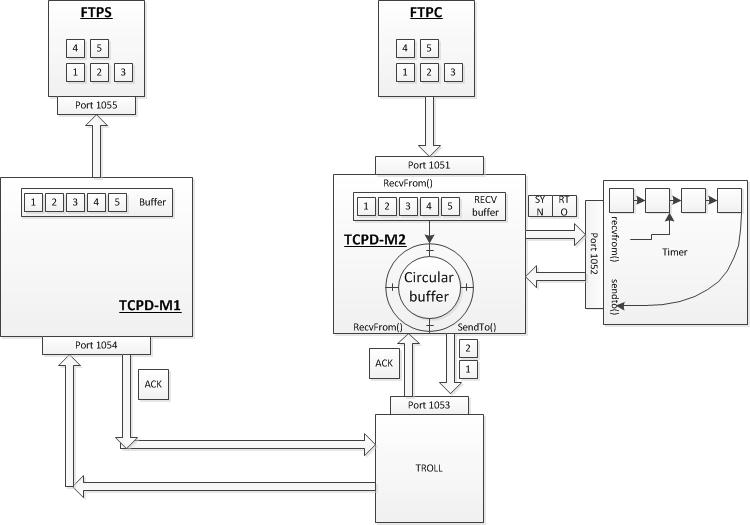
1. FTPC – the client application that sends the file to the destination.
2. FTPS – the server application that receives the file sent by the source.
3. TCPD (M1/M2) – The TCP Daemon process is equivalent to the TCP stack in the OS and is responsible for the actual transfer of data. This involves storing 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.

##### Important Terms and definitions:

|  |  |
| --- | --- |
| TCP | Transport Control Protocol. Provides a reliable way of transferring packets from source to destination. |
| UDP | User Datagram Protocol. Provides an unreliable way of transferring packets from source to destination. |
| ACK | Acknowledgement. This is a confirmation message that is sent back to the source for a packet received. |
| RTT | Round Trip Time (Round Trip Delay) is the time taken for a packet to travel from a source to a destination and back to the source again. |
| RTO | Recovery Time Objective. The maximum possible tolerable length of time for the ACK of a packet to come back to the source. |
| CRC | Cyclic Redundancy Code. |
|  |  |
|  |  |
|  |  |
|  |  |

##### Architecture:

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



##### 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.

###### FTPC:

**Wrapper functions calls for TCP using UDP:**

1. **Initialize the Socket connection:**

SOCKET (int family, int type, int protocol)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Family | Integer | AF\_INET |
| Type | Integer | SOCK\_DGRAM |
| Protocol | Integer | 0 |

1. **Bind Socket name to Socket:**

BIND (int client\_socket, struct sockaddr \*myaddr, int addrlen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Myaddr | Pointer | - |
| Addrlen | Integer | 0 |

1. **Accept a connection from the Socket:**

ACCEPT (int client\_socket, (struct server\_addr \*) &client\_addr, &sin\_size)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Client\_addr | Pointer | - |
| Sin\_size | Integer | 0 |

1. **Establish connection with the server:**

//CONNECT (int client\_socket, (struct server\_addr \*) &client\_addr, sizeof(struct sockaddr))

CONNECT (NULL)

1. **Send message from buffer via the Socket:**

SEND (int client\_socket, const void \*msg, int len, int flags, const struct sockaddr \*to, int tolen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Msg | Pointer to buffer | - |
| Len | Integer | 0 |
| Flags | Integer | 0 |
| To | Pointer | - |
| Tolen | Integer | 0 |

1. **Read message from the Socket and place in buffer:**

RECV(int client\_socket, void \*buf, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Buf | Pointer to buffer | - |
| Len | Integer | - |
| Flags | Unsigned Integer | 0 |
| From | Pointer | - |
| Fromlen | Integer | 0 |

1. **Close the Socket connection:**

CLOSE()

**UDP wrapper functions: (Is this needed?)**

1. **Initialize the Socket connection:**

Socket(int family, int type, int protocol)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Family | Integer | AF\_INET |
| Type | Integer | SOCK\_DGRAM |
| Protocol | Integer | 0 |

1. **Bind Socket name to Socket:**

Bind(int sockfd, struct sockaddr \*myaddr, int addrlen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Myaddr | Pointer | - |
| Addrlen | Integer | 0 |

1. **Read from the Socket and place in buffer:**

Recvfrom(int sockfd,void \*buf, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Sockfd | Integer | 0 |
| Buf | Pointer to buffer | - |
| Len | Integer | - |
| Flags | Unsigned Integer | 0 |
| From | Pointer | - |
| Fromlen | Integer | 0 |

1. **Send message from the buffer via the Socket:**

Sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Sockfd | Integer | - |
| Msg | Pointer to buffer | - |
| Len | Integer | 0 |
| Flags | Integer | 0 |
| To | Pointer | - |
| Tolen | Integer | 0 |

1. **Close the Socket connection:**

Close(client\_socket)

**Miscellaneous function calls:**

1. **To wait for events on multiple Sockets:**

Select(int maxfd, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Maxfd | Integer | - |
| Readfds | Pointer | - |
| Writefds | Pointer | - |
| Exceptfds | Pointer | - |
| Timeout | Pointer | - |

###### TCPD-M2:

**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.

**Function calls:**

* + sendTimerInfo (int sequenceNumber, int time, int flag)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| sequenceNumber | Integer | 0 | Contains the sequence number of a particular packet. |
| time | Integer | 0 | Contains the RTO value of the packet. |
| flag | Integer | 0 | Indicates one out of 3 operations: Insert, Update, Delete. |

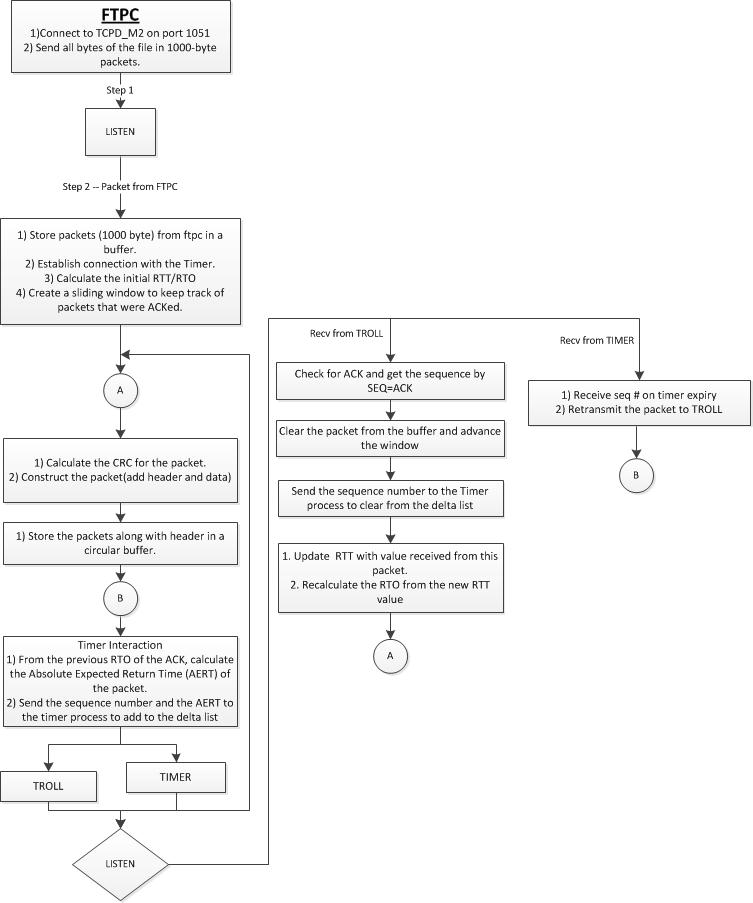
* + sendDataToTroll()
  + recvDataFromTroll()
  + calculateCRC()
  + calculateRTT()

**Circular-buffer implementation:**

The circular buffer helps in 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.

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| startPointer | Pointer | - |
| endPointer | Pointer | - |
| dataPointer | Pointer | - |
| Count | Integer | 0 |



###### FTPS:

**Wrapper functions calls for TCP using UDP:**

1. **Initialize the Socket connection:**

SOCKET (int family, int type, int protocol)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Family | Integer | AF\_INET |
| Type | Integer | SOCK\_DGRAM |
| Protocol | Integer | 0 |

1. **Bind Socket name to Socket:**

BIND (int client\_socket, struct sockaddr \*myaddr, int addrlen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Myaddr | Pointer | - |
| Addrlen | Integer | 0 |

1. **Accept a connection from the Socket:**

ACCEPT (int client\_socket, (struct server\_addr \*) &client\_addr, &sin\_size)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Client\_addr | Pointer | - |
| Sin\_size | Integer | 0 |

1. **Establish connection with the server:**

//CONNECT (int client\_socket, (struct server\_addr \*) &client\_addr, sizeof(struct sockaddr))

CONNECT (NULL)

1. **Read message from the Socket and place in buffer:**

RECV(int client\_socket, void \*buf, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen)

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Initial value** |
| Client\_socket | Integer | - |
| Buf | Pointer to buffer | - |
| Len | Integer | - |
| Flags | Unsigned Integer | 0 |
| From | Pointer | - |
| Fromlen | Integer | 0 |

1. **Close the Socket connection:**

CLOSE()

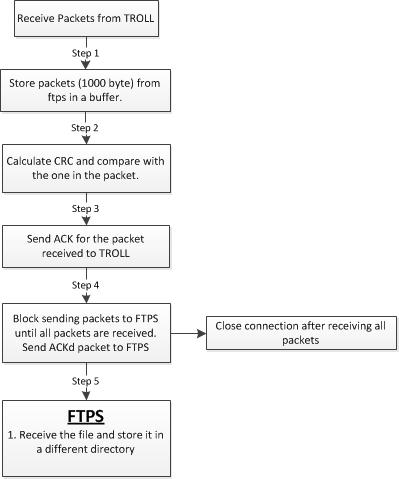
Operations:

* + Receive the file from TCPD-M1 and store it in a directory different from that of FTPC.

###### TCPD-M1:

Operations:

* + Receiver message from Troll and store in buffer.
  + CRC
  + Send ACK to Troll.
  + Send the Acknowledge packet to FTPS
  + Close the connection.



##### RTT and RTO calculation:

##### Checksum Computation:

##### Packet format:

**TCP Header:**

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

TCP Segment

IP datagram

16-bit source port number 16-bit destination port number 32 – bit sequence number

32 – bit acknowledgement number

4-bit header and the six flags 16-bit window size

16-bit TCP checksum 16-bit urgent pointer Data

20 bytes

###### **Timer**:

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

Seq #, time

Seq #, time

Typedef struct timerNode {

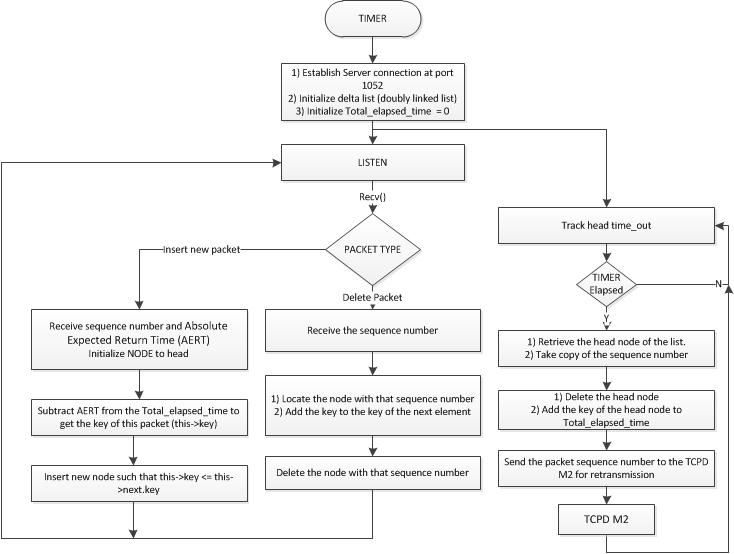
timerNode \*previous;

unsigned int data;

timerNode \*next;

}

* + **Function calls:**
    - sendSignalToTCPD(int sequenceNumber)
      * This function sends an interrupt to the TCPD-M2 process when the ACK for a packet does not come back on time.



##### Connection shutdown:

***Tools:***

***Project Management:***

**Task split:**

|  |  |  |
| --- | --- | --- |
| **Task** | **Responsibility** | |
| Project proposal | Ananth | Abhishek |
| Architecture design |  | Abhishek |
| Function calls, Algorithms | Ananth |  |
| Coding – ftpc, ftps, ftpd-M1 |  | Abhishek |
| Coding – Timer, ftpd-M2 | Ananth |  |
| Testing | Ananth | Abhishek |
| Final report | Ananth | Abhishek |

***Time-line:***

|  |  |
| --- | --- |
| **Task** | **Due date** |
| Project proposal | 4/12/2012 |
| Coding – Timer | 4/21/2012 |
| Ftpd-M1 and Ftpd-M2 | 4/28/2012 |
| ftpc, ftps | 5/5/2012 |
| Testing | 5/17/2012 |
| Final report | 5/26/2012 |

##### Conclusion:

##### 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>