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

### Abhishek Dharmapurikar Ananth Mahadevan

Contents

[Problem Statement: 2](#_Toc321951929)

[Abstract: 2](#_Toc321951930)

[Important Terms and definitions: 2](#_Toc321951931)

[Architecture: 3](#_Toc321951932)

[Design (function calls and Data structures): 3](#_Toc321951933)

[FTPC: 3](#_Toc321951934)

[TCPD-M2: 4](#_Toc321951935)

[FTPS: 6](#_Toc321951936)

[TCPD-M1: 7](#_Toc321951937)

[RTT and RTO calculation: 8](#_Toc321951938)

[Checksum Computation: 8](#_Toc321951939)

[Packet format: 8](#_Toc321951940)

[*Timer*: 9](#_Toc321951941)

[Connection shutdown: 9](#_Toc321951942)

[Tools: 10](#_Toc321951943)

[Test-cases: 10](#_Toc321951944)

[Project Management: 10](#_Toc321951945)

[References: 10](#_Toc321951946)

### 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.

### Architecture:

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

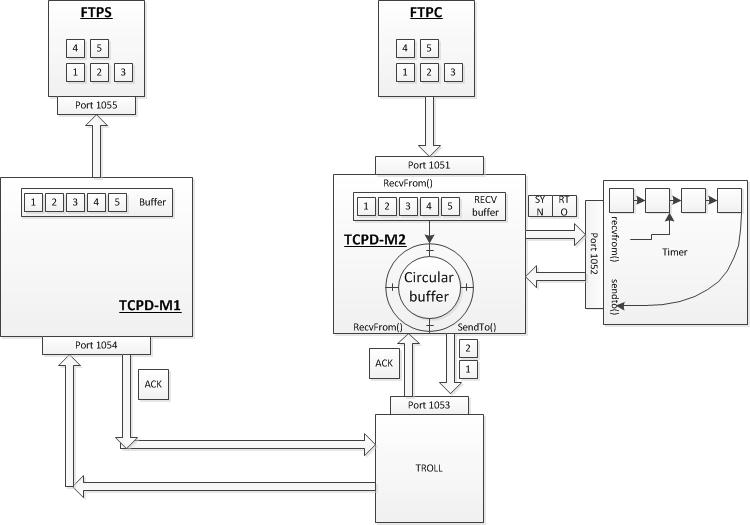


Figure : High level architecture

### 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.

### FTPC:

**Operations**:

1. To split the file into packets of 1000-bytes each and send to the FTPD-M2 process.

**Functions calls (wrapper) 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 (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()

**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:



Figure 2 Flow of 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.

**Important Function calls:**

1. **Setup UDP connection as a server with the FPTC as client on port 1051 and setup Client connection with troll.**
   * socket(int socket\_family, int socket\_type, int protocol);

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| socket\_family | Integer | AF\_INET | To indicate the family of the protocol. |
| socket\_type | Integer | SOCK\_DGRAM | To specify the type of protocol TCP/UDP |
| protocol | Integer | 0 | Indication to choose protocol on type |

* + bind(server\_s, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| socket\_family | Integer | AF\_INET | To indicate the family of the protocol. |
| socket\_type | Integer | SOCK\_DGRAM | To specify the type of protocol TCP/UDP |
| protocol | Integer | 0 | Indication to choose protocol on type |

1. **Recv Data from from FTP\_C and store it in the receive buffer**

recvfrom(int sockfd, void \*recv\_buffer, int len, unsigned int flags, struct sockaddr \*from, int \*fromlen);

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| sockfd | Integer | returned by socket | socket file descriptor |
| recv\_buffer | void \* | 0 | Points to the receive buffer |
| len | Integer | 1000 | length of the bytes to be received from FTPC |
| flag | unsigned int | 0 | usually set to 0 |
| from | struct sockaddr | - | returnes the details of the client the data is received from |
| fromlen | int \* | sizeof(struct sockaddr) | Initialized to the size of the buffer associated with address |

1. **Setup connection with the timer as a client**.

socket(int socket\_family, int socket\_type, int protocol);

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| socket\_family | Integer | AF\_INET | To indicate the family of the protocol. |
| socket\_type | Integer | SOCK\_DGRAM | To specify the type of protocol TCP/UDP |
| protocol | Integer | 0 | Indication to choose protocol on type |

1. **Estimate the initial RTT and calculate RTO from the value.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| RTT | Integer | 5 | RTT estimator |
| RTO | Integer | calculated from RTT | RTO estimator |

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

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 |

The size of the buffer would be (1000(payload) + 20(header))\*64 bytes.

To implement the sliding window following variables would be used

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comment** |
| LastPacketAcked | Pointer | Start of buffer | Indicating the last acked packet |
| LastPacketSent | Pointer | Start of buffer | Indicating the last packet that was sent. |
| LastPacketWritten | Pointer | Start of buffer | To point to the last packet that was written to the buffer. |

1. **Packetization: Header construction.**

The destination address and port number known, a sequence number is calculated and the header is calculated for the data in the receive buffer in chunks of 1000 bytes.

1. **CRC calculation.**

With the payload and the header decided upon the CRC calculation is done on the packet. For that a custom function would be used.

int calculateCRC(int sequenceNumber)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comments** |
| sequenceNumber | Integer | - | The sequence number of the packet on which CRC is to be calculated is sent |

The calculation of CRC is explained in later sections.

1. **Add the packet to the buffer.**

Add the checksum calculate from the above buffer to the packet header and store the packet in the buffer pointed by LastPacketWritten and then increment the value by one.

1. **Calculate AERT for the packet.**

To the current timestamp value add the RTO value to get the Absolute Expected Return Time for the packet. This value would be sent to the timer.

1. **Send the packet AERT to the Timer**

With the AERT for the packet calculated it is sent to the timer for managing the retransmissions signals.

The packet to the timer would consist of.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comments** |
| sequenceNumber | Integer | - | The sequence number of the packet |
| AERT | Integer | - | Absolute Expected Return timer |
| flag | Integer | 0 | Indicates that the packet is to be inserted into the delta list. |

1. **Recvfrom TROLL.**

The acknowledgement of the packets that are sent would be received from TROLL. Once such packet arrives the packet sequence in the field pointed by LastPacketAcked +1 is checked for sequence number. If the sequence number matches the same value is incremented. The current timestamp is also noted.

1. **Delete the packet from the delta list.**

A packet with the following information is sent to the Timer process to delete the entry of the packet from the delta list.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comments** |
| sequenceNumber | Integer | - | The sequence number of the packet |
| flag | Integer | 1 | Indicates that the packet is to be deleted from the delta list. |

1. **Calculate the RTO from the acknowledged packet.**

The function would be used to calculate the RTO passing the current RTT of the acked packet.

calculateRTO(int M);

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comments** |
| M | Integer | - | The RTT of the packet recently acknowledged |

1. **Recv Packet from Timer**

If a packet is received from the Timer it would be to indicate that the packet with the sequence number sent has to be retransmitted. The details received in the packet would be.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comments** |
| sequenceNumber | Integer | - | The sequence number of the packet to be retransmitted |

Then the steps for transmission of the packet would be repeated.

### FTPS:

**Operations:**

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

**Functions calls (wrapper) 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(int sockfd)

### 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.

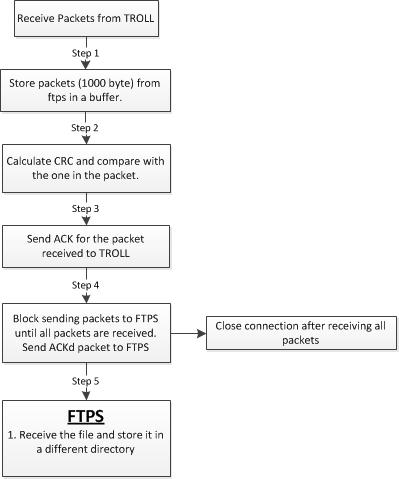
**Function calls:**

* + recvfrom() To receive the data from TROLL. Implementation same as explained in TCPD\_M2
  + int calculateCRC(int sequenceNumber)

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Data type** | **Initial value** | **Comments** |
| sequenceNumber | Integer | - | The sequence number of the packet on which CRC is to be calculated is sent |

The calculation of CRC is explained in later sections.

* + sendto() To send the acknowledge for the packet received to TROLL. Implementation same as explained in TCPD\_M2.
  + sendDataToFTPS()
  + close()



### 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:

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 will be 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.

### Checksum Computation:

In order to calculate the Checksum, a pseudo header is 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. |

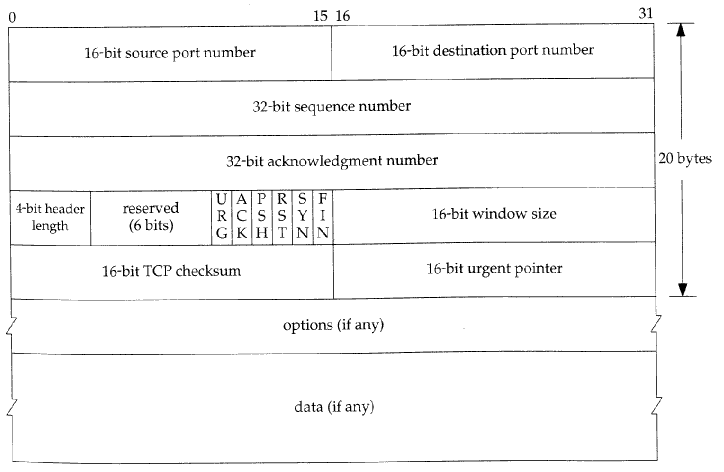
### Packet format:

**TCP Header:**

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

TCP Segment

**IP datagram**



### **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

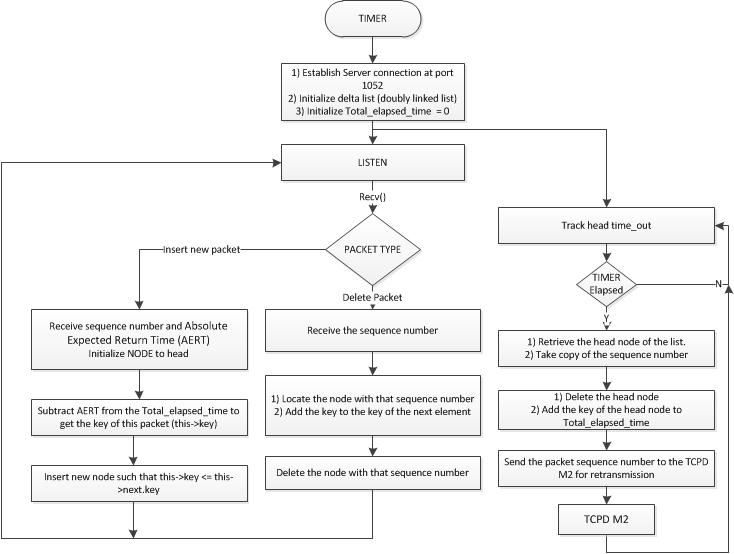
Typedef struct timerNode {

timerNode \*previous;

unsigned int data;

timerNode \*next;

}



The Timer would interact with the TCPD\_M2 process with 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. |

### Connection shutdown:

Mentioned as part of the individual process’ function calls.

### Test-cases:

|  |  |  |
| --- | --- | --- |
| **Process** | **Test** | **Expected result** |
| FTPC | Send packets to FTPD-M2 | Data to be stored in the receive buffer of FTPC |
| FTPD-M2 | Calculate the CRC | Proper value of CRC to be calculated |
| FTPD-M2 | Create header for packet | Header information to be created for the packets. |
| FTPD-M2 | Calculate RTO and RTT | Proper value of RTT and RTO to be calculated. |
| FTPD-M2 | Send data to Troll | Packet information to be sent to the Troll. |
| FTPD-M2 | Signal from Timer | To resend packet for the corresponding seq # to the Troll. |
| Timer | Send signal to FTPD-M2 on expiry of time | FTPD-M2 to resend packet for the corresponding seq # to the Troll. |
| FTPD-M1 | Calculate the CRC | Proper value of CRC to be calculated |
| FTPD-M1 | Send ACK to Troll | FTPD-M2 to receive the ACK. |
| FTPD-M1 | Send packets to FTPS | Packets to be sent only after all of them are received. |
| FTPS | Recreate the file | The file has to be recreated and stored in a new location. |

### 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 |

### 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>