# NETWORK PROGRAMMING

CSE 5462 Project Report

Ganga Reddy Tankasala (tankasala.1@osu.edu)
Jagannath Narasimhan (narasimhan.23@osu.edu)

**04/23/2015**

## Table of Contents

## 1. Overview

The goal of this project was to implement a TCP-like reliable transport layer protocol using the unreliable service provided by UDP, and then write a simple file transfer application to demonstrate its operation.

## 2. Details of Each Process Used

### 2.1. FTPC (Client Process)

Here we use the client application which splits the files that is to be transferred into equal parts (except for the last packet) and sends to the TCPD Client Process. This communication with the TCPD Client process is using UDP Sockets.

The general operations of the Client Process are as follows:

a. Splits the file into packets of n-bytes (1000bytes in our case) each and send them to the FTPD-M2 process. The last packet be of size = size of file to be transferred in bytes % 1000, where % is the modulo operation which calculates the remainder.
b. Function calls (SOCKET(), BIND() and CONNECT()) are made in this process.
c. Read the information from the file to be transferred and call function SEND() to transfer the data to FTPS. SEND() is a blocking call and does not return a successful acknowledgement until all the are written to the TCPD Buffer.
d. FTPC Client process is invoked using the following command:

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

### 2.2. Client Daemon (TCPD – Client )

The Client Daemon is a background process running on the client side. This process implements the circular buffer and is the process that receives the packets from the client process and forwards in the required fashion to Troll process.

Some of the processes of the TCP Daemon process that is being run on the client side is as mentioned below:

a. Receive packets from the client process and store it in the wrap around buffer.
b. Calculate and update RTT and RTO.
c. Calculate CRC Checksum for the packets to be sent.
d. Communicate with Timer process and trigger resending of packets to the Troll process if the ACK for a particular packet is not received within time out.
e. Close all connections with Timer and FTPC processes.

To keep track of the packets that are being acknowledged, we use a sliding window concept in this process. This daemon process does the following in brief: Create packets by adding the header and payload as needed. Sliding Window is initialized to keep track of the packets that are being Acknowledged or not. Connection to the Timer process is done. Calculate CRC of packets to be transmitted. Check for ACK by SEQ number. If ACK received, remove the packet from the buffer. If ACK is not received for a particular SEQ number on time out of timer process, retransmit that particular packet to Troll process and begin timer process for that packet afresh. Update Sliding Window. Keep updating calculation of RTT and RTO from the ACKs that are received. Connect to TCPD Server process and LISTEN for incoming packets (ACKs).

### 2.3. Server Daemon (TCPD – Server)

The Server Daemon is a background process running on the server side. Some of the processes of the TCP Daemon process that is being run on the client side is as mentioned below:

a. Receive messages from the TROLL process arriving from the client side and store the packets in a buffer.
b. Calculate the CRC Checksum for all received packets.
c. SEND ACKS for all packets that are received.
d. SEND received packets to the FTPS process once all packets are received.
e. CLOSE connections with TROLL and FTPS.

### 2.4. FTPS (Server Process)

Here we use the server process which writes the packets that are received from the client process into a file. The packets are received from the client process via TCPD Client, TROLL and TCPD Server processes. Some of the operations of the FTPS process is as mentioned below:

a. Receive the packets from the TCPD Server and create file and store in a directory.
b. Function calls such as SOCKET(), BIND() and ACCEPT() are made.
c. Blocks and waits until a connection is made from Client.

### 2.5. TROLL Process

In the Standard Linux CSE Network, dropping packets may not be very probable. Also, detecting which packets are dropped is not very easy. Hence, to simulate a lossy network, and to increase our control on rate of dropping, garbling and delaying packets in the CSE Network, we use the TROLL Process. This process allows us to vary the rates and percentages of Dropped, garbled and delayed packets. When the Trace option is turned ON, we get detailed information on which packets are dropped, garbled and delayed.

## 3. Buffer and Wraparound Buffer

### 3.1. Buffer

Buffer (Data Buffer) is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another. Buffer is the fundamental unit through which data transfer takes place in the following ways:

a. Within a system ( data moving between processes)
b. Between I/O systems ( data movement from/ to I/O devices)

Buffers can be implemented in a fixed memory location in hardware or using virtual data buffer in software. Generally the data buffers are implemented in software which use the Random Access memory (due to faster access time compared to hard disk drives.)

Network communication involves transfer of data bytes from one system to other connected system in the network. This process involves utilization of data buffers at the sending system, transporting system and receiving system.

## 3.2. Wrap around buffer (Cyclic buffer)

Buffers are highly useful when the input and output rate of data transfer are different at any intermediate node. A circular buffer is a memory allocation scheme where memory is reused (reclaimed) when an index, incremented modulo the buffer size, writes over a previously used location. A circular buffer makes a bounded queue when separate indices/pointers are used for inserting and removing data. A circular buffer with *n* elements is used to implement a queue with n-1 elements--there is always one empty element in the buffer used to distinguish between a full and empty buffer.

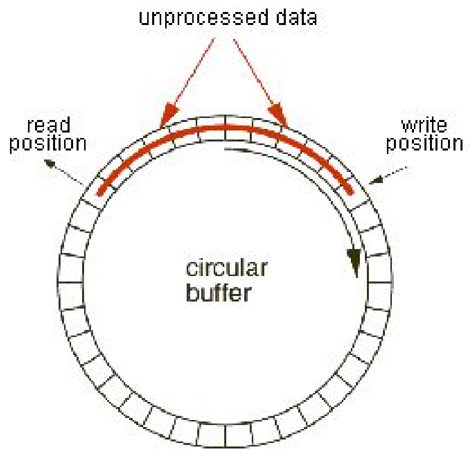

Figure 1: Circular Buffer

## 3.3. Buffer Management and Sliding Window Protocol

Sliding window protocols are used where reliable in-order delivery of packets is required.
Fixed window size of 20.
The send and receive buffers will be wrap-around (or circular) buffers of 64 KB for sending and 64 KB for receiving and MSS (max number of bytes sent as payload) of 1000 Bytes.

At any time the send buffer maintains a list of sequence numbers it is permitted to send - these fall within the **sending window**. These are frames sent-but-no-ack and frames not-yet-sent.

The receiving window maintains a list of sequence numbers it is expected to receive. Once it receives them, it acknowledges and moves forward.

## 4. Timer Process

Delta-list is one of many ways to maintain a list of timers. A doubly linked list will be implemented for delta_list. Most of the details mentioned in the project page will be maintained such as

- Delta-time: This "time" field contains the time beyond the previous event (previous node). This is maintained in the form of difference from the previous node.
- Port number: This is the port number where a notification needs to be sent when the timer expires.
- Information: Information of byte sequence number of the first byte of the packet for which the timer was started that needs to be sent back to the process.

The following are function calls to add and delete entries from the delta_list.

a. void Insert ( Node )
b. void Delete ()
c. void Start_timer()
d. Void Cancel_timer()

Structures needs to be define for the following

a. Node ( of linked list)
b. Timer_recv_packet
c. Timer_send_packet

## 5. Checksumming Algorithm (CRC)

The CRC (Cyclic Redundancy Code) checksumming technique should be used for computing the checksum. The binary representation of payload whose checksum has to be calculated is padded with n bits (set to zeros). A general n-bit binary CRC, includes a ($n$ + 1)-bit pattern representing the CRC's divisor (polynomial) which does a XOR operation on the binary data iteratively from MSB to LSB of data. The last n bits which are modified due to consecutive XOR operations is considered as checksum and added to the payload.

The validity of a received message can easily be verified by performing the above calculation again, this time with the n bit checksum added instead of zeroes (as in constructing the checksum). The remainder should equal zero if there are no detectable errors.

## 6. RTT and RTO

Jacobson's algorithm is implemented for computing RTT and RTO where RTT = Round Trip Time, RTO = Retransmit Timeout, SRTT = Smoothed RTT.

The lower the timeout is, the less time we have the transmission stopped. However if the timeout is too low, then we risk of retransmitting the packets assuming the packets are dropped during load congestions adding more congestion.The original RFC793 TCP sets the timeout to be twice the estimated RTT, which is estimated with the following average (a low-pass filter which cuts fast variations):

$SRTT[i] = (1-a) * SRTT[i-1] + a * RTT$
$0 < a < 1$; usually $a = 1/8$

$RTO = 2 * SRTT$

However the above does not address the RTO effectively as when the load increases RTT increases and variance of RTT increases even more. By setting the RTO to two times RTT may result in retransmission of packets due to shorter RTO. To avoid this

Jacobson proposed an alternative approach based on the variance of RTT.

$RTO = SRTT + u * Dev(RTT)$

Where Dev (RTT) is the mean deviation of the RTT samples and u uses to be u=4.

## 7. Packet Formats

Formats for all packets exchanged between TCPD, FTPC, FTPS, TROLL,and TIMER processes. Information passed between different ends in a network are passed in discrete steps known as packet. Packet can be termed as a fundamental unit of data transfer in a network. In the process of communication from client/server to server/client, the packet reaches various stages such as

a. Server (FTPS)
b. TCP Daemon at server end(TCPD)
c. TROLL (server end)
d. TROLL ( Client end)
e. TCP Daemon at Client end(TCPD)
f. Timer Process
g. Client ( FTPC)

| | |
|---|---|
| Server | The system which is actively listening and catering requests from clients |
| Client | The system which is requesting information from Server |
| Daemon | Daemons are processes that run continuously in the background and perform functions required by other processes |
| Troll | All communication will go through Troll Process, which is a utility that allows to introduce network losses and delay in order to simulate real network behaviour within a system. |

The following packet structures will be defined for data exchange between TCPD, FTPC, FTPS, TROLL, and TIMER.

struct ftpc_to_daemon
struct daemon_to_ftpc
struct ftps_to_daemon
struct daemon_to_ftps
struct daemon_to_troll

struct troll_to_daemon
struct daemon_to_timer
struct timer_to_daemon

Client reads the file and SENDS the buffer to client daemon (M2)

struct ftps_to_daemon
struct ftpc_to_daemon

Daemon writes the data to a circular buffer (64k) and the buffer management functions will then take bytes from the buffer and create packets.

```
struct daemon_to_troll
{
Struct sockaddr_in dest;
char body [MSS_SIZE + TCP_HEADER_SIZE];
}

struct troll_to_daemon
{
        struct sockaddr_in dest;
        char body[ TCP_HEADER_SIZE];
}

struct packet
{
        struct tcphdr hdr;
        char payload[MSS_SIZE];
}
```

## 8. Description of Implementation of all Capital Functions

The following functions are implemented for the project

a. BIND -- Binding addresses TCPD server , client , Troll
b. SEND -- Send message from Client/Server to TCP Daemon
c. RECV -- Recv message from TCP Daemon by Server/ Client
d. CONNECT -- To Connect Client/Server to TCP Daemons
e. ACCEPT -- First Connection from an address
f. CLOSE -- End message to mark the end of file transfer

## 9. Connection Shutdown

Unlike TCP, UDP is a connectionless protocol hence shutdown of the connection is not possible. However for acknowledging the complete transfer of data /files, application of artificially created predetermined chat scripts for sending shutdown/finish (and their acknowledgements) can be used to mark the end of connection between the server and client.

## 10. Description of how to Compile, Run and Test the Program

## 11. Possible Future Extensions to make this Program more efficient and to add more features

### Time Line and Distribution

Taking into consideration our emphasis on learning, we agreed to work independently on the project at each stage but before we move on the next stage, we will merge two of our independent work. Having mentioned that we also took sole responsibility for the work in the following fashion. The code sharing will be done using private repositories in Git. The code will remain private until completion of the project.

a. Connection setup -- Ganga Reddy -- Feb 15
b. RTT Computation -- Jagannath Narasimhan -- Feb 22
c. Checksum Computation -- Ganga Reddy -- Mar 01
d. Packet Formation -- Jagannath Narasimhan -- Mar 08
e. Timer Implementation -- Ganga Reddy -- Mar 09
f. Buffer management and sliding window protocol -- Both -- Mar 15
g. Connection shutdown -- Both -- Mar 21

A time buffer of one week is given at each stage so as not cross any of the required deadlines. Project documentation will be done for each stage of project and final report will be verified before Mar 28.