CSE 5462: Project Report

Adithya Bhat and Rakesh Mohan

Section 1: Project Overview and details of each process

Problem Statement

In this work we propose to implement a TCP-like reliable transport layer protocol, using the unreliable services provided by User Datagram Protocol (UDP). The operations and functionality of the proposed protocol will be tested by a simple file transfer application running on top of the protocol.

Introduction

In this work we implement a TCP like reliable transport layer protocol using the unreliable services provided by UDP. The operation of the new implemented protocol is then validated by a simple file transfer application. Specifically, we implement the functionalities of TCP socket functions like Send and Recv using the UDP socket functions.
The focus of the project is on data transfer. Most of the TCP functionality is implemented in the TCPD (TCP daemon) process which is equivalent to the TCP in the OS that runs in the background. The functions such as Send and Recv are required to communicate with the local TCPD process. The communication between the application process and the local TCPD process is implemented with UDP sockets. UDP communication within a machine is assumed to be reliable. We then write a simple file-transfer application that uses our TCP implementation. The file-transfer protocol includes a server called ftps and a client called ftpc.

The ftpc client sends all the bytes of that local file using our implementation of TCP. The ftps server receives the file and then stores it. The file-transfer application uses a simple format. The first 4 bytes (in network byte order) contain the number of bytes in the file to follow. The next 20 bytes contains the name of the file. The rest of the bytes to follow contain the data in the file. To simulate real network behavior, all communication between the two machines goes through troll processes. Troll is a utility that allows us to introduce network losses and delay.

Client Process

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().

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>

Client Daemon

The operations of the TCP Daemon process in the client side are as given below:
o Receive message from FTPC and store in wrap-around (Circular) buffer.
o Calculate and keep updating the RTT and RTO values.
o Calculate the CRC for a packet.
o Manage the Wrap-around buffer.
o Keep track of the ACKs for the packet and resend is required by communicating with the Timer.
o Close the connections with the Timer and the FTPC process.

1) Store packets (1000 byte) from ftpc in a buffer. 2) Establish connection with the Timer. 3) Calculate the initial RTT/RTO 4) Create a sliding window to keep track of packets that were ACKed. 1) Construct the packet(add header and data) 2) Calculate the CRC for the packet. Check for ACK and get the sequence by SEQ=ACK 1) Receive seq # on timer expiry 2) Retransmit the packet to TROLL 1) Store the packets along with header in a circular buffer. Timer & Troll Interaction 1) From the previous RTO of the ACK, calculate the Absolute Expected Return Time (AERT) of the packet. 2) Send the sequence number and the AERT to the timer process to add to the delta list 3) Send the packet to the Troll process Clear the packet from the buffer and advance the window Send the sequence number to the Timer process to clear from the delta list 1. Update RTT with value received from this packet. 2. Recalculate the RTO from the new RTT value FTPC 1)Connect to TCPD_M2 on port 1051 2) Send all bytes of the file in 1000-byte packets. TIMER TROLLStep 2 -- Packet from FTPC Step 1 LISTEN LISTEN Recv from TROLL Recv from TIMER B B A A

Server Daemon

The operations of the TCP Daemon process in the server side are as given below:
o Receive message from the TROLL and store packets in a buffer.
o Calculate the CRC for the received packet.
o Send the ACKs for the received packets.
o After receiving all packets, send them to the FTPS process.
o Close the connections with the TROLL and the FTPS process.

Server Process

Operations:

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

2. Makes the function calls SOCKET(), BIND() and ACCEPT().
3. Blocks until a connection is made to it from the client.

Troll

In the CSE network it is hard to artificially create real network scenarios (lossy links, packet garbling etc.) We use the troll utility to control the rate of garbling, discarding, delaying or duplication of packets. All packets will first go through a local troll process running on the same machine, where they will be subject to delay, garbling and/or drops. The packets will then be forwarded to the intended destination.

Section 2: Detailed description of each project component

Circular Buffer

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:
a. Pointer to the actual buffer location in memory that gives the start of the buffer.

Seq #, time
Seq #, time
b. Pointer to the end of the buffer location in memory that gives the end of the buffer.
c. Pointer to insert data into the buffer, which moves along with the data inserted.

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.

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.

Checksum:

We implemented the CRC check summing algorithm. Specifically, we have used the standard CRC-16 implementation with following parameters:

Width = 16 bits; **Polynomial =** 0x8005; Initial **Remainder =** 0x0000;
Final **XOR Value =** 0x0000
Reflect Data? = Yes; **Reflect Remainder? =** Yes; **Check Value =** 0xBB3D

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 and h is set to .25)
RTO = A + 4D

The steps followed before sending a packet will be as follows:

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

Packet formats:

Packet format for all communications (FTPC to TCPD_M2,TCPD_M2 to TCPD_M1 , TCPD_M1 to FTPS)

```
data_packet {
char payload[1000];
int sequence_number;
char FYN;
crc checksum;
}
```

Packet format between TCPD_M2 and Timer

```
timer_packet {
float time ; //Associated Time
int sequence_number ; //Sequence Number of Packet
int type ; // 0 - Insert; 1 - Delete
}
```

Implementation of TCP socket functions

SOCKET()
Creates a socket connection. Internally uses the standard sock() call but creates a UDP socket.

BIND()
Binds the created socket to the address specified. Uses standard bind().

ACCEPT()
Implements a select() function call which is a blocking call. FTPS makes use of this function while waiting to receive data from TCPD_M1 so that it unblocks on receipt of data.

CONNECT()

Implements a sendto() function call. Library makes use of CONNECT() to send the destination address details of FTPS (where TCPD_M1 is also located) to TCPD_M2.

SEND()
Implements a sendto() function call. FTPC makes use of SEND() to send data to TCPD_M2.

RECV()
FTPS makes use of this function to receive the file payload from TCPD_M1. Uses the address that was

CLOSE()
This function closes the socket connections. If it is FTPC, it also sends the "end-of-file" message to TCPD_M2 which then forwards it to TCPD_M1 (More details in connection shutdown) signaling the end of transmission.

Connection Shutdown

The 4-way TCP connection teardown has been implemented in this project the library libtcpd_socket, TCPD_M2 and TCPD_M1 have their roles in this. On CLOSE() the library sends "end-of-file" message to TCPD_M2. On its receipt TCPD_M2 sends the FIN to TCPD_M1 and enters into the FIN_WAIT1 state.

TCPD_M1 sends the ACK for the FIN received from TCPD_M2 and enters CLOSE_WAIT, but does not send a FIN back until its buffer is empty (meaning, it has received all data packets). After the buffer becomes empty, M1 sends a FIN and enters into the LAST_ACK state. It also starts a timer so that it can retransmit the FIN if the ACK from TCPD_M2 does not return in time. After the ACK comes back, TCPD_M1 closes its connection and enters CLOSED state.

When a FIN from TCPD_M1 is received TCPD_M2 enters the FIN_WAIT2, sends an ACK when its buffer is empty and enters the TIME_WAIT state where it waits for a period of 2MSL. This is to make sure that last ACK sent is not lost. Post 2MSL time it closes

Section 3: Detailed description of how to compile and run

Included as a Readme file with the code; not repeated in the report due to space restrictions.

Section 4: Project Management:

TASK	RESPONSIBILITY	
Project Proposal	Rakesh	Adithya
Function Calls, Algorithms	Rakesh	
Coding - ftpc,ftps,ftpd-M1		Adithya
Coding - Timer, ftpd-M2	Rakesh	
Testing	Rakesh	Adithya
Final Report	Rakesh	Adithya

References:

- TCP/IP Illustrated, Volume 1 – The Protocols by W. Richard Stevens.
- https://en.wikipedia.org/wiki/Circular_buffer
- https://en.wikipedia.org/wiki/Sliding_window_protocol
- https://en.wikipedia.org/wiki/Round-trip_delay_time
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm
- http://beej.us/guide/bgnet/output/html/multipage/index.html