Project 1 Final Report
Name: Zicong Zheng, Shuang Li

1. Overview

In our project, we implement 5 process and adopt troll to simulate the various quirks and vagaries of a network environment.

a. Ftpc

Ftpc is the client end of this ftp. In ftpc, we can assign the IP, port and the filename which we want to send. It will send packet to tcpd, in which we implement the basic function of TCP protocol.

b. Ftps

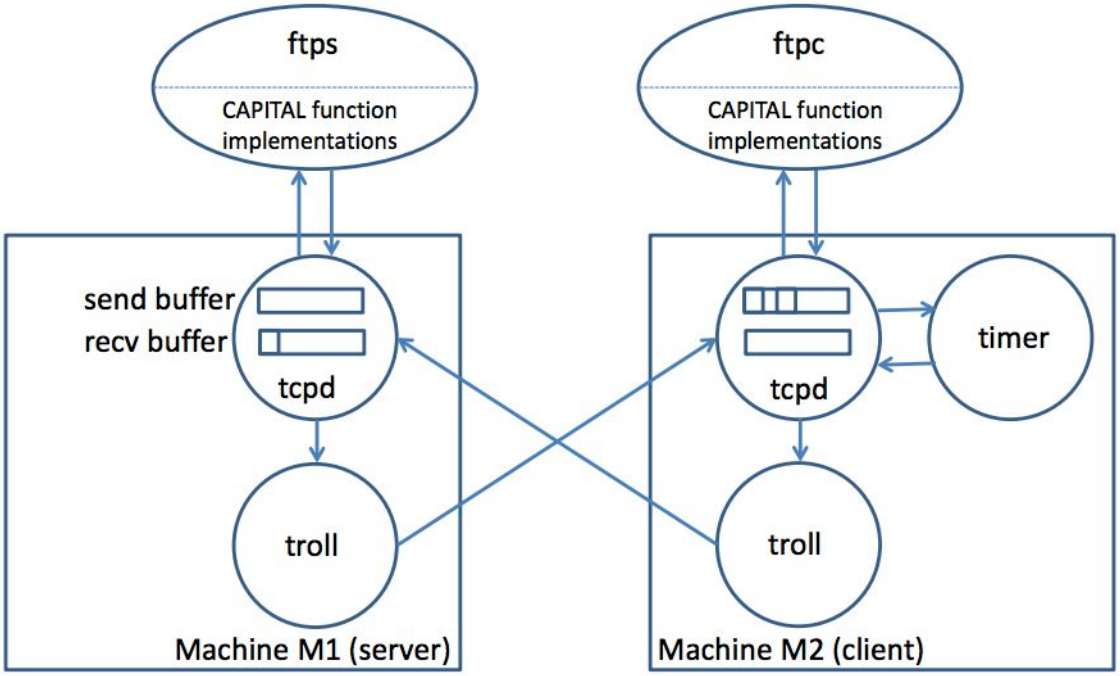
Ftps in the server side which used to receive file sent from ftpc.

c. Tcpd

Tcpd has two type. One is used for client end, and the other is for server end. Most of the core functions like check sum, package retransmission are in this process. It use UTP to implement a reliable transmission.

d. Timer

Timer maintains a delta time list to keep monitor any timerout transmission.



(Figure 1: The overview of our project. From the handout of project1 in CSE5462: http://web.cse.ohio-state.edu/~prasun/cse5462/proj/proj.pdf)

2. Packet Format

Each packet sent out from tcpd client contains four components: troll header for troll transmission, tcp header for tcp protocol, time_stamp for storing the time when send out the packet, and a buffer with size of 1000 bytes for storing actual data.

The ack packet also contains the first three parts, the same with regular packet, but it does not contains a buffer, since there is no need to store data in ack packets.

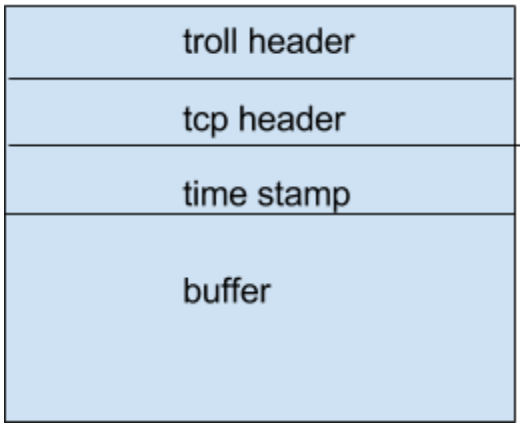


Figure 2: The packet format detail

3. Checksum Computation

Use CRC (Cyclic Redundancy Code) checksumming technique to computing the checksum for each packet (includes ack packet), and then store it in the checksum field in the tcp header.

Here we write a function crcFast(uint8_t const message[], int nBytes) to compute the checksum byte by byte. Note that for a given input remainder and generator polynomial, the output remainder will always be the same. So it's possible to precompute the output remainder for each of the possible byte-wide input remainders and store the results in a lookup table. That lookup table can then be used to speed up the CRC calculations for a given message. We create an array crcTable[256] and use the function crcInit() to initialize it.

The tcpd client side program will check the checksum field whenever receive an ack packet, only accept ungarbled packets.

The same as tcpd server side, it will check the checksum field whenever receive a packet from troll running on the client side, only accept ungarbled packets.

Reference website:

http://www.barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code

4. Buffer Management

On the tcpd client side, there is a buffer called sendbuf, in which the size is 64KB. When tcpd receive a packet from ftpc, and if the packet is outside of the sliding window, the packet will be stored in the sendbuf. Use a variable called bufpos to indicate where to write for the next packet coming from ftpc.

On the tcpd server side, there is a buffer called recvbuf, in which the size if also 64KB. When tcpd receive a packet from client side, store it in the correct position in recvbuf according to its sequence number. It will skip the duplicate packets and garble packets.

5. Sliding Window Protocol

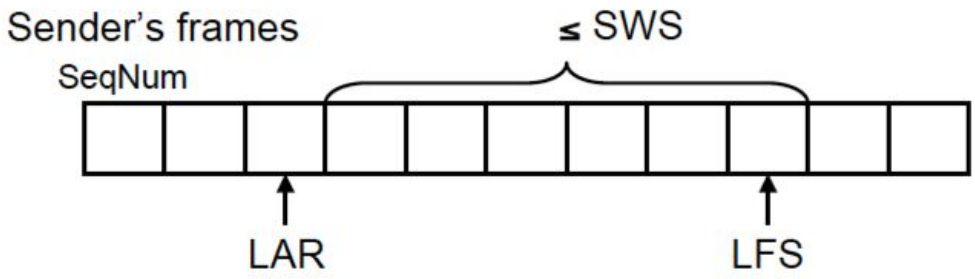


Figure 3: Sender Side Sliding Window

On the sender side, sender assigns a sequence number to each packet, in our project, the sequence number equals the buffer position in which the data stored.

- SWS: Send Window Size
- LAR: Largest ACK Received
- LFS: Last Frame Sent

Sender Algorithm:

(1) LFS - LAR <= SWS

(2) Associates timeout with each frame sent. Retransmits corresponding packets if no ACK received before timeout.

(3) When ACK arrives, increase LAR, which means another frame can be sent

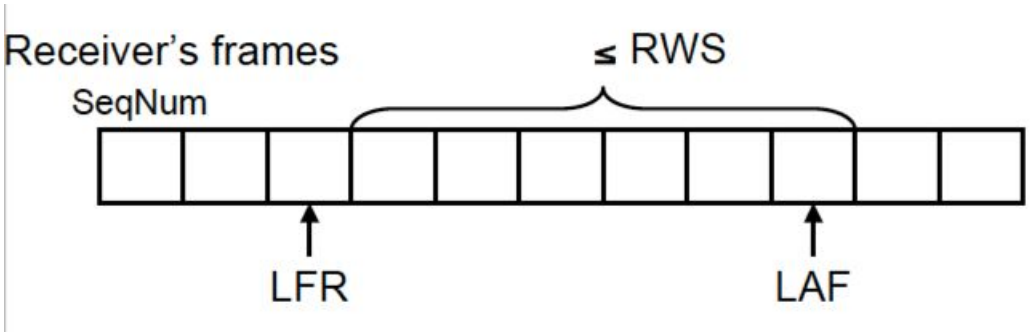


Figure 4: Receiver's Side Sliding Window

On the receiver side,

- RWS: Receive Window Size
- LAF: Largest Acceptable Frame
- LFR: Largest Frame Received

Receiver Algorithm:

(1) When packet numbered SeqNum arrives, if (SeqNum ≤ LFR) or (SeqNum > LAF) discard(still send the ack back to client side), else accept the packet

(2) Update LFR when receive ack = LFR + 1, since in our program, when we receive an ack, we will empty corresponding position of the buffer, so we find the first non-empty position to update LFR.
(3) LAF = LFR + RWS

6. Capital Function

(a) SOCKET

Return socket() function.

(b) BIND

Return bind() function.

(c) ACCEPT

Null function.

(d) CONNECT

Null function.

(e) SEND

ftpc will read bytes from the file and use the function SEND() to send data to ftps. The SEND() function call will need to send these bytes to the local tcpd process.

(f) RECV

ftps executes the RECV() function and waits for data to arrive. The RECV function will send a packet to the local tcpd process to inform that it is waiting for data and the maximum amount of data that is requesting to receive. When data comes in to the tcpd process at the server machine, it sends it to the right port on which ftps is waiting for data.

(g) CLOSE

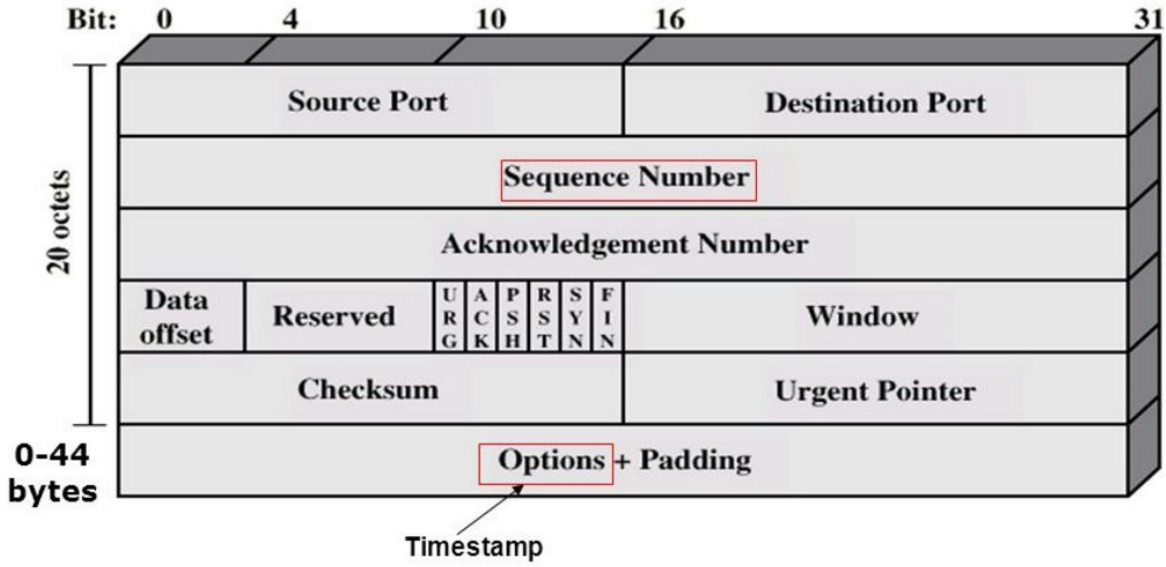
Return close function.

7. RTT Computation

Implement the Jacobson's algorithm for computing RTT and RTO

Calculating RTT:

We use TCP timestamps, defined in RFC 1323, to store the packet sending time. As shown below, TCP timestamp use the options part of the TCP header. It has the size of 16 byte.



(Figure 5:Timestamp in TCP header. Image from
http://images.slideplayer.com/20/5954997/slides/slide_12.jpg)

Before we send a packet, we get the present time in a timeval structure, and copy it into Timestamp domain in TCP header. Then we send this packet. When the tcpd server side receives this packet, it copy the Timestamp domain out from the packet and put it into ack packet, then send it back to tcpd client end. When the tcpd client-end receives the ack packet, it copy out the Timestamp, get present time and calculate rtt use:

$$rtt = time_{send} - time_{recv}$$

Everytime we get a new rtt, we calculate RTO by Jacobson's algorithm, the details are as follows:

$$srtt_{n+1} = \alpha \ RTT + (1- \alpha)\ srtt_n$$
$$rttvar_{n+1} = \beta\ (|\ RTT - srtt_{n+1}\ |) + (1- \beta)\ rttvar_n$$
$$RTO_{n+1} = srtt_{n+1} + 4\ rttvar_{n+1}$$

(http://images.slideplayer.com/28/9350474/slides/slide_46.jpg)

8. Timer Implementation

As the figure showing below, we use two processes to implement timer. At first, we use fork to create a child process as timer, and the parent process is driver as you may think. We open two pipelines for inter communication. When we send a packet at tcpd, the driver will send a packet to timer. This packet contains the sequence number and RTO. When timer receives this packet, it will create a node in delta list. In timer process, it maintains a timeval structure called lasttime, which stores last time when messages was sent into timer. Every time a new message was sent in, timer will get present time and calculate the elapsed time using

$$elapsed_{time} = present_{time} - last_{time}.$$

Then it goes through the delta list, find the timeout nodes and send a packet to driver to indicate the timeout nodes.

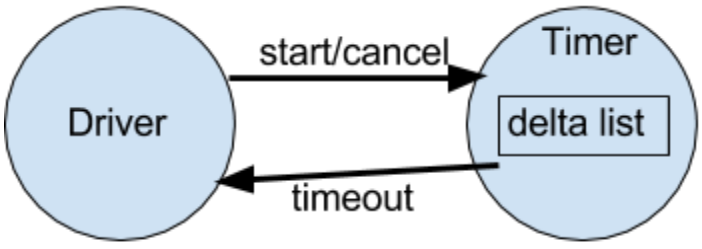


Figure 6: Implementation of Timer

9. Potential Enhancements for the Future

a. Fix bugs in retransmission.

b. Instead of send ack packet for every packet received, we can implement TCP's cumulative ACK.
c. Implement slow-start, congestion control and flow control algorithms.
d. Change the hard-coding code, so that we can send packets according to the parameters typed in the command line.

10. Division of Responsibilities

Zicong Zheng	Transfer Framework, Checksum Computation, Packet Format Design, Buffer Management, Sliding Window Protocol, Implementation of Capital Function
Shuang Li	RTT, RTO Computation, Timer Implementation, Packet Retransmission