CSE 5462

REPORT

Jiabei Xu, Guanzhu Li

Introduction

TCP is a reliable stream delivery service which could guarantee all bytes received in the correct order as the bytes sent. Not like TCP, UDP uses a simple connectionless transmission model with a minimum of protocol mechanism, which has no handshaking dialogues, and thus there is no guarantee of the information delivery, ordering, or duplicate. In this project, we would like to build a TCP-like reliable transport layer protocol using the unreliable service provided by UDP, and this application could transfer file. We use checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram to ensure the transfer.

Process

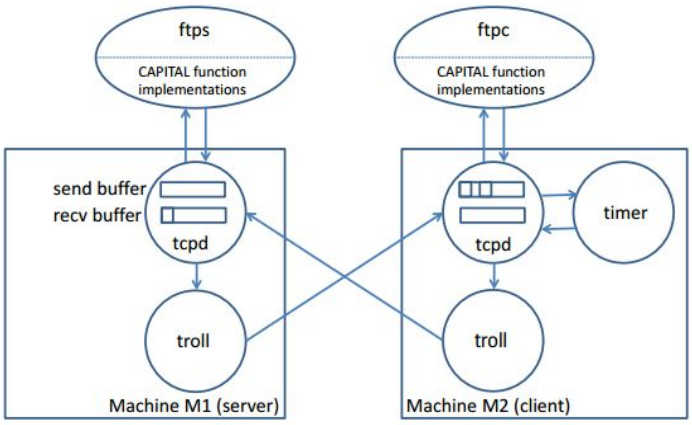

*Image from Project description, writer Prasun Sinha, OSU

The focus of the project is on data transfer. We use the TCP daemon process to achieve TCP functionality. UDP sockets is implemented to process the application communication. Local troll processes ensures the communication between the two machines.

(a) Start the troll process and the TCPD process on machines SERVER and CLIENT.
(b) On machine SERVER, function `SOCKET()`, `BIND()` and `ACCEPT()` are called to start the file-transfer server ftps, and it will block in the first call to `RECV()`.
(c) On machine CLIENT, function `SOCKET()`, `BIND()` are used to start the file-transfer client ftpc.

(d) Function `CONNECT()` initiate TCP handshaking between the two TCPD processes, we don't need to use this in this project because we are not using a authentic tcp connection.

(e) In TCPD(CLIENT), the buffer is composed for the connection.

(f) In `ftpc`, it will read bytes from the file, and function `SEND()` will be used to send data to `ftps`.

(g) A new packet will be created, which contains the bytes from buffer, server message, TCPD(SERVER) message, and prepare to send it to TCPD(SERVER).

(h) The ftps application will call function `RECV()` to receive data .

(i) During upon process, in the machine SERVER, function checksum will examine the file content from received data whether it is same or not, to ensure the correctness. Acknowledge number and sequence number will also compared to check whether there is missing or duplicate problem.
If the message exists ordering, missing or duplicate problem, a new packet(acknowledge message) on TCPD(SERVER) will be composed, and it should contain the acknowledge number(sequence) which unit need to be re-sent, and through troll, the message will send to TCPD(CLIENT).
Based on the message from TCPD(SERVER), TCPD(CLIENT) will construct a new packet and resend the data.

(j) In the machine CLIENT, RTO(Retransmission TimeOut)/RTT(Round Trip Time) retransmission mechanism is to measure whether the data need to retransmit or not.Timer is utilized to calculate the transmission time interval.
Once timeout, TCPD(CLIENT) will, through troll, resend the previous packet to TCPD(SERVER).

(k) After sending all the bytes of the file, ftpc closes the connection. The `CLOSE()` function call will initiate to close the connection.

(l) When receiving all the bytes of the file, ftps will also use `CLOSE()` function to close the connection.

Component

(a) Wraparound buffer
Selective Repeat algorithm is implemented to solve sequence number dilemma in

communications. The buffers are designed as wrap-around buffers that 64 KB for sending, 64KB for receiving, and 1000 Bytes MSS. Use Sliding Window, a fixed window size of 20, in wraparound buffer to avoid reordering problem. In the receiver, LastByteRead point to the node is reading in TCP buffer, and NextByteExpected is the last node receiver expected to get in window. LastByteRcved is the last position of packet.
In the sender, LastByteAcked is pointed to the position acked, and LastByteSent is the node sent but not acked. LastByteWritten is the node written. Also, the sender will adjust the size of sending data according to the window situation to make sure the data received.

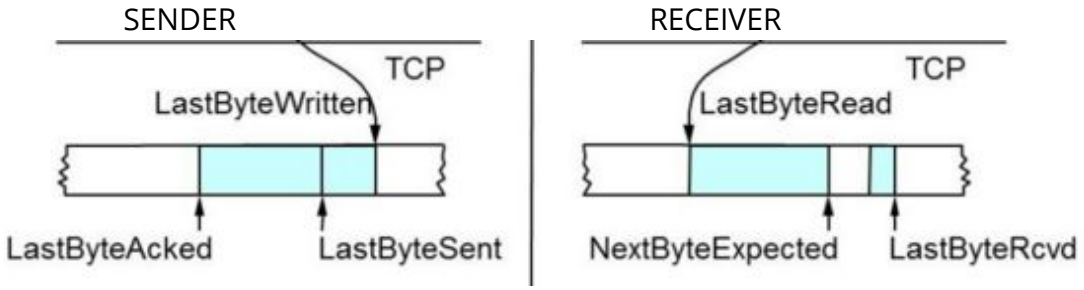

*Image from http://coolshell.cn/articles/11609.html

(b) Timer

The timer is a controller to decide whether the packet needs to resend, and the time interval(delta-time) is calculated according RTP/RTT algorithm. During the transmission, a delta-list timer will start when each packet begin to send. When the timer expires, the packet will need to be re-transmitted.

In the delta-list, Delta-time contains the time beyond the previous node. Port number is the port needs to be sent when the timer expires. The information needs to be sent back to the process that started this timer.

When new port added, the local timer need to initiate a new timer, where the delta-list will be updated, and TCPD(CLIENT) will send a message to the it that indicates the delta-time timer needs to run. When it times out, the timer process could send notification that this port expires that the corresponding node will be deleted, and the byte sequence number of the packet for which this timer will start.

(c) Checksum

A checksum is a small-size datum from a block of digital data for the purpose of detecting errors, which could ensure the data integrity during the transmission. In this project, CRC (Cyclic Redundancy Code) checksumming technique is implemented to calculate the checksum.

In packet receive process, TCPD(CLIENT) packet contains the checksum which is calculated by content and its length before sending to TCPD(SERVER). When TCPD(SERVER) receive the packet, it will recalculate the checksum, and compare it with the checksum in TCPD(CLIENT) packet to check the content integrity.

In retransmission process, CRC is implemented to check the integrity of acknowledge message.

(d) RTT/RTO

RTO(Retransmission TimeOut)/RTT(Round Trip Time) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received. This retransmission mechanism is to reduce long time waiting, and more efficient. Once times out, TCPD(CLIENT) will, through troll, resend the previous packet to TCPD(SERVER).

(e) Capital Function

(1) `SOCKET()`

Implemented `socket()` to open socket.

(2) `BIND()`

The function `BIND()` is used to listen from certain port. In machine SERVER, ftps listen the TCPD(SERVER) port, and in the machine CLIENT, ftpc will listen to the ctrl port.

(3) `ACCEPT()`

In this project, `ACCEPT()` is a null function.

(4) `CONNECT()`

In this project, we would not implement TCP handshaking. So `CONNECT()` is also a null function.

(5) `SEND()`

The `SEND()` function call will need to send these bytes to the local TCPD process, and these bytes will be temporarily saved in a wrap-around buffer.

And `SEND()` is implemented as a blocking function call that It would not return until all bytes in the buffer passed in the argument is written in the TCPD buffer.

(6) `RECV()`

`RECV()` will read the bytes from TCPD buffer, and it could fill up the maximum size of buffer for receiving. Also, it will return the number of bytes it received.

(7) `RECV_CTRL()`

Function `RECV_CTRL()` is to control the sending status, if the window is full, the ftpc will stop sending.

(8) `CLOSE()`

Implement `close()` to close the connection. In this project, we use `close()` directly.

(f) Packet Formats

We have defined `TcpdMessage` as the packet for transmitting in this project. It consists of three parts: `tcp_header`, `tcpd_header`, and `contents`. It will be stored in `troll_message.content` when sending to troll. Detailed data structure of each component is shown below.

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

*Image from Wikipedia

In TCP Header, `seq`, `ack_seq`, `FIN`, `check` will be occupied to ensure the transmit.

TCPD Header

Sin_family: AF_INET	Sin_port: IP port.	Sin_addr: IP address.	sin_zero

TCPD Message

Struct TCPD Header	Strunc TCP Header	Char content(1000)

(Troll) Net Message

Struct socketaddr_in	Char content

Execution

1. build

   Just run make.

2. run

   server(beta):

   (1) start `troll` on server(note: troll port is fixed)

   ```
   ./troll 7010 -x25 -g25 -m25 -t
   ```

   (1) open a new terminal, run `tcpd` as a daemon.

   ```
   ./tcpd -s &
   ```

   (2) run `ftps`.

   ```
   ./ftps <valid-port>
   ```

   client:

   (1) ssh to new host, and run `troll`(note: troll port is fixed)

   ```
   ./troll 7000 -x25 -g25 -m25 -t
   ```

   (2) open a new terminal. run `tcpd` as daemon

   ```
   ./tcpd -c &
   ```

   (3) start `timer`

   ```
   ./timer &
   ```

   (3) run `ftpc`

   ```
   ./ftpc beta <valid-port> <local-file-to-transfer>
   ```

   file will be transferred to recv/ directory.

Future

If we send package too fast, tcpd is prone to have error. We are not sure whether it is troll's problem or not. We will look into it in the future.