

Introduction to Computer Networks - Lab 2

Name: Pei Chi, Huang

Student ID: 108020017

Department: Interdisciplinary Program of Science

I. Basic Implementation

■ Server side

The server side connects to an UDP socket of port 9999 and wait for a request from a client. It will respond to the client when requested “download fileName”. If the file exists, the server sends packets to the client using stop-and-wait mechanism, and receives ACK from the client. The sender will also handle timeout events by resending the packets if not receives the ACK in 100 milliseconds.

sendFile()	
<pre>int filesize = ftell(fd); int file_cnt = filesize / MAXLINE; if((filesize % MAXLINE) != 0) file_cnt++; snd_pkt.header.no_of_byte = filesize % MAXLINE;</pre>	Calculate the total number of packets needed, and set the remaining number of bytes at the last packet.
<pre>while(fread(buffer, 1, sizeof(buffer), fd) > 0) { // update send packet header memcpy(snd_pkt.data, buffer, sizeof(buffer)); snd_pkt.header.seq_num++; // printf("packet %d:\n", snd_pkt.header.seq_num); // ===== // Set is_last flag for the last part of packet // ===== if(file_cnt == snd_pkt.header.seq_num + 1) { snd_pkt.header.islast = 1; } // ===== // Send video data to client // ===== numbyte = sendto(sockfd, &snd_pkt, sizeof(snd_pkt), 0, (struct sockaddr *) &client_info, len); printf("\tSend %d bytes\n", numbyte);</pre>	Read 1024 bytes from the given file name and send to the socket. The sequence number is updated if a new packet is created. Also, set the flags of the last packet to be 1.
<pre>int ret = recv(sockfd, &rcv_pkt, sizeof(rcv_pkt), 0); while (ret < 0) { sendto(sockfd, &snd_pkt, sizeof(snd_pkt), 0, (struct sockaddr *) &client_info, len); printf("\tTimeout!! Resend packet!\n"); ret = recv(sockfd, &rcv_pkt, sizeof(rcv_pkt), 0); } printf("\tReceive a packet ack_num = %d\n", rcv_pkt.header.ack_num);</pre>	Using a nonblocking recv() with timeout interval. If there is ACK received within the time interval, the packet is successfully transmitted and the server keep sending the next packet. If not, resend the packet.
<pre>printf("Send file successfully\n"); fclose(fd); return 0;</pre>	If all the packets are sent, close the file and return. Otherwise, repeat the above steps.

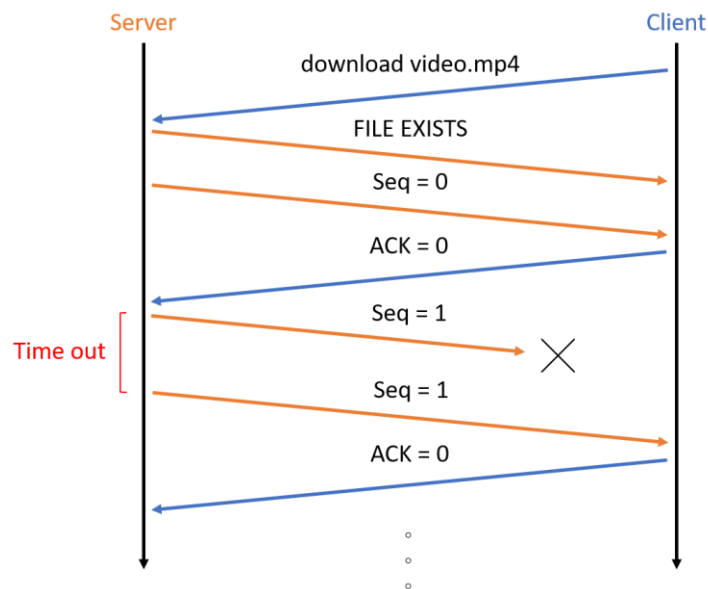
* The recv() only returns after a large amount of time (longer than timeout) here.

■ Client Side

The client side connects to the UDP socket of the server and make commands “download fileName” to the server. If the file exists, the client starts to receive the packets and replies an ACK back to the server. During the process, the client may loss packets at random, and will ignore the packet received.

recvFile()	
<pre>sprintf(fileName, "download_"); strcat(fileName, str); // Open file fd = fopen(fileName, "wb");</pre>	Open the file requested.
<pre>recvfrom(sockfd, &rcv_pkt, sizeof(rcv_pkt), 0, (struct sockaddr *) &info, (socklen_t *) &len);</pre>	Receiving the packet sent from the sender.
<pre>if(isloss(0.5)) { printf("\tOops! Packet loss!\n"); }</pre>	The packet losses with the possibility of 0.5. If a loss event occurs, ignore the packet.
<pre>printf("\tReceive a packet seq_num = %d\n", rcv_pkt.header.seq_num); // ===== // Reply ack to server // ===== snd_pkt.header.ack_num = rcv_pkt.header.seq_num; snd_pkt.header.seq_num = 0; snd_pkt.header.is_last = 0; sendto(sockfd, &snd_pkt, sizeof(snd_pkt), 0, (struct sockaddr *) &info, len); printf("\tSend ACK of packet %d\n", rcv_pkt.header.seq_num);</pre>	If the packet is successfully transmitted, the client replies to the server with an ACK of the sequence number of the received packet.
<pre>if(rcv_pkt.header.is_last == 1) { fwrite(rcv_pkt.data, 1, rcv_pkt.header.last_no_byte, fd); break; } else { fwrite(rcv_pkt.data, 1, sizeof(rcv_pkt.data), fd); }</pre>	Write the data to the download file. If the packet is denoted as the last one, the whole file is transmitted, then return.

■ Diagram



II. Result

Both server.c and client.c are compiled by gcc. The result was shown below:

Server side	Client side
<pre> peggy@DESKTOP-HEPFL9:/mnt/c/Users/peggy/CanLab/CAN-Lab2\$ make server gcc server.c -lpthread -o server -lm peggy@DESKTOP-HEPFL9:/mnt/c/Users/peggy/CanLab/CAN-Lab2\$./server 9999 Parameter: Server's IP is 127.0.0.1 Server is listening on port 9999 server waiting.... process command.... filename is video.mp4 FILE EXISTS server: sent 1040 bytes to 127.0.0.1 transmitting... Send 1040 bytes Receive a packet ack_num = 0) Send 1040 bytes Timeout!! Resend packet! Receive a packet ack_num = 1) Send 1040 bytes Receive a packet ack_num = 2) Send 1040 bytes Receive a packet ack_num = 3) Send 1040 bytes Timeout!! Resend packet! Timeout!! Resend packet! Receive a packet ack_num = 4) Send 1040 bytes Timeout!! Resend packet! Receive a packet ack_num = 5) Send 1040 bytes Receive a packet ack_num = 110) Send 1040 bytes Receive a packet ack_num = 111) Send 1040 bytes Receive a packet ack_num = 112) Send 1040 bytes Receive a packet ack_num = 113) Send 1040 bytes Timeout!! Resend packet! Timeout!! Resend packet! Receive a packet ack_num = 114) Send 1040 bytes Receive a packet ack_num = 115) Send 1040 bytes Timeout!! Resend packet! Receive a packet ack_num = 116) Send 1040 bytes Timeout!! Resend packet! Receive a packet ack_num = 117) Send 1040 bytes Timeout!! Resend packet! Receive a packet ack_num = 118) Send 1040 bytes Timeout!! Resend packet! Timeout!! Resend packet! Timeout!! Resend packet! Receive a packet ack_num = 119) Send 1040 bytes Timeout!! Resend packet! Receive a packet ack_num = 120) Send file successfully server waiting.... </pre>	<pre> peggy@DESKTOP-HEPFL9:/mnt/c/Users/peggy/CanLab/CAN-Lab2\$ make client gcc -o client client.c -lm peggy@DESKTOP-HEPFL9:/mnt/c/Users/peggy/CanLab/CAN-Lab2\$./client give me an IP to send: 127.0.0.1 server's port? 9999 Waiting for a commands... download video.mp4 client: sent 1040 bytes to 127.0.0.1 client: receive 1040 bytes from 127.0.0.1 FILE EXISTS Receiving... Receive a packet seq num = 0 Send ACK of packet 0 Oops! Packet loss! Receive a packet seq num = 1 Send ACK of packet 1 Receive a packet seq num = 2 Send ACK of packet 2 Receive a packet seq num = 3 Send ACK of packet 3 Oops! Packet loss! Oops! Packet loss! Receive a packet seq num = 4 Send ACK of packet 4 Oops! Packet loss! Receive a packet seq num = 5 Send ACK of packet 5 Oops! Packet loss! Receive a packet seq num = 6 Send ACK of packet 6 Receive a packet seq num = 110 Send ACK of packet 110 Receive a packet seq num = 111 Send ACK of packet 111 Receive a packet seq num = 112 Send ACK of packet 112 Receive a packet seq num = 113 Send ACK of packet 113 Oops! Packet loss! Oops! Packet loss! Receive a packet seq num = 114 Send ACK of packet 114 Receive a packet seq num = 115 Send ACK of packet 115 Oops! Packet loss! Receive a packet seq num = 116 Send ACK of packet 116 Oops! Packet loss! Receive a packet seq num = 117 Send ACK of packet 117 Oops! Packet loss! Receive a packet seq num = 118 Send ACK of packet 118 Oops! Packet loss! Oops! Packet loss! Receive a packet seq num = 119 Send ACK of packet 119 Oops! Packet loss! Receive a packet seq num = 120 Send ACK of packet 120 End of receiving Total byte received: 123431 Total cost 122 secs Waiting for a commands... </pre>