Homework #4

✓ **Please upload your answer sheet in LMS. The uploading file must be PDF.**

✓ **Also upload your source codes in the Linux servers as shown below.**

✓ **Due date: 11pm, 12/06 (Mon)**

1. **Implement the stop-and-wait protocol-based file transfer program over UDP as shown below. You should upload the source codes in Linux servers and explain about your source codes in the answer file with screenshot. (60points)**

- Server)

structure of Packet: 먼저 packet 의 구성은 struct 을 통해서 만들었습니다. 따라서 rdt3.0 에서 사용되는 pkt#와 ack#의 값을 담은 packet 을 client 에게 전송할 수 있습니다. 또한 15 번째 줄의 pkt_kind 는 packet 이 어떤 종류의 packet 인지를 표시하는 것이고, 2(FIN)이 들어오면 client 가 파일 전송을 끝냈다는 뜻입니다. 또한 data_type 은 payload 에 어떤 데이터가 저장되어 있는지 알 수 있는 tag 입니다. payload_len 은 data 의 길이라고 생각하시면 됩니다. 마지막으로 payload 에는 전송 해야할 data 가 저장되어 있습니다.

```
14  typedef struct packet {
15      int pkt_kind ; // 0: ack#, 1: pkt#, 2: FIN
16      int pkt_num ; // 0 or 1, -1: not used
17      int ack_num ; // 0 or 1, -1: not used
18      int data_type ; // 0: file_name, 1: file_data, -1: some message
19      int payload_len ;
20      Data payload ; // data
21  } Packet ;
```

Figure 1. structure of Packet

Flow of code: Server 가 CLI 를 통해서 입력 받은 <port>에 socket 을 열고, Figure1 의 packet 을 통해서 client 로부터 data 를 받습니다. 또한 udp_rdt_ft_server.c 은 rdt 3.0 을 기반으로 구현된 코드이기 때문에 client 로부터 받은 pkt#을 받고, 그에 해당하는 ack#를 다시 client 에 전송해야 합니다. 따라서 Figure 2 의 코드로 client 로부터 packet 을 받은 후, 구현된 동작을 하게 됩니다.

```
81          // receive packet
82          if ( recvfrom( serv_sd, pkt_recv, sizeof( Packet ), 0, (struct sockaddr*)&clnt_addr, &clnt_addr_size ) == -1 ) {
83              printf( "Fail to receive packet.\n" ) ;
84              fclose( fp ) ;
85              free( pkt_send ) ;
86              free( pkt_recv ) ;
87              exit(1) ;
88          }
```

Figure 2. receive packet from client

client 로부터 받은 packet 의 data_type 이 0 이면 payload 에 filename 이 저장되어 있어서, 파일을 열고 다시 client 에 ack#를 보내게 됩니다. data_type 이 1 이면 payload 에 file 의 data 가 저장되어 있어, 열려있는 file 에 data 를 저장하고 다시 client 에 ack#을 전송합니다. 만약 client 가 data 전송이 끝났다면 pkt_kind 에 2(FIN) 저장되어 있기 때문에 client 에 FIN 과 ack#이 저장되어 있는 packet 을 전송합니다.

- **Client)**

client 도 server 와 똑같은 형태의 packet structure 을 사용합니다. 먼저 프로그램을 돌리면, CLI 를 통해서 입력 받은 file 이 안 열려있기 때문에, fp 가 NULL 일 것입니다. 따라서 fopen()을 통해서 파일을 열어주고 filename 을 server 로 전송하게 됩니다. 그리고 그 다음 동작부턴 fp 에 값이 할당돼 있기 때문에 정상적으로 data 를 server 에 전송하게 됩니다.

```
86          // when file is not opend
87          if ( fp == NULL ) {
88              // open file
89              if ( ( fp = fopen( argv[3], "rb" ) ) == NULL ) {
90                  printf( "Fail to open %s.\n", argv[3] ) ;
91                  exit( 1 ) ;
92              }
```

Figure 3. when file is not opened

하지만 server 와 다르게 client 는 packet loss 가 일어나는 상황을 고려해야하기 때문에 server 로부터 ack# packet 을 받을 때 timeout 을 정해야 합니다. 따라서 loss 가 없는 상황이라면 다음 pkt#와 data 를 server 로 전송하고, loss 가 일어났다면 timeout 이 일어나기 때문에 packet 을 retransmission 합니다. 또한 premature timeout 이 일어날 수 있기 때문에, duplicated 된 ack#는 Figure 4 처럼 무시하게 됩니다.

```
165             // case: ignore packet
166             else if ( pkt_recv->ack_num != pkt_num && pkt_recv->pkt_kind == 0 ){
167                 // printf( "------receive from server ack#: ack%d\n", pkt_recv->ack_num ) ;
168                 printf( "Ignore packet.\n" ) ;
169                 continue ;
170             }
```

Figure 4. ignore ack#

만약 모든 data 를 다 전송했다면 feof()함수를 통해 감지하고, Figure 5 처럼 FIN 을 server 로 전송하게 됩니다.

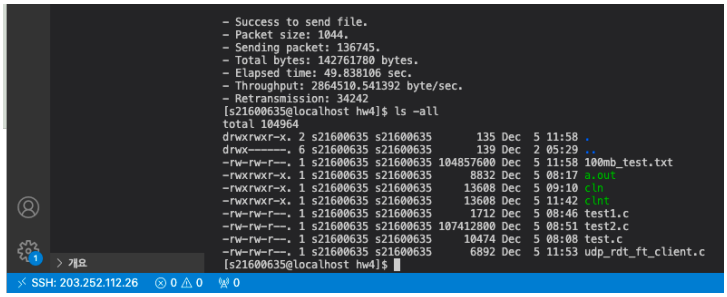
```
108             // end of file
109             if ( feof( fp ) ) {
110                 // send pkt option to FIN
111                 set_pkt( &pkt_send, 2, pkt_num, -1, -1, 0,"nothing in payload" ) ;
112
113                 // send pkt
```

Figure 5. when ended transmission

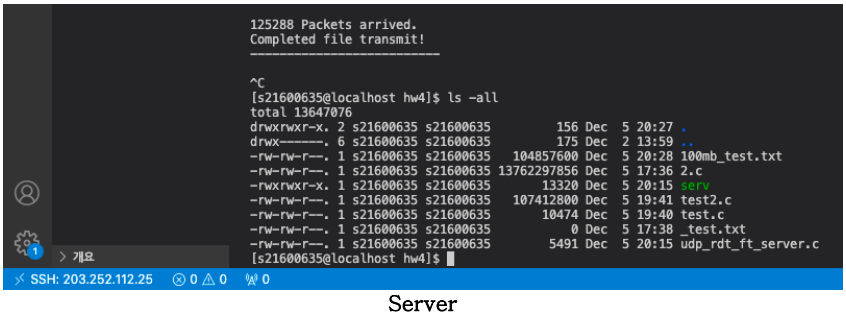
그리고 server 로부터 FIN 과 ack#을 담은 packet 을 받게 되고 파일 전송이 끝나게 됩니다.

- 결과)

client 가 100mb_test.txt 를 server 전송했을 때, retransmission 은 총 34242 번 일어났다는 것을 볼 수 있다. 하지만 전송된 파일의 크기를 비교해 봤을 때, 크기가 같다는 것을 보면 전송이 제대로 됐다는 것을 볼 수 있다.



Client



Server

Figure 6. Result

2. What is the NAT traversal problem? What are the solutions? You need to give two or more solutions and explain its mechanisms. (Within one page) (20points)

NAT 은 Network Address Translation 의 약자이며, <private IP address, port#>를 <public IP address, new port#>로 바꿔주는 것이다. 따라서 local network 에서 public network 로 packet 을 전송할 때 NAT translation table 에 바뀐 정보를 저장한다. 따라서 public network 에서 local network 로 packet 이 들어올 때도 제대로 작동할 수 있게 된다. 하지만 반대로 NAT translation table 에 mapping 정보가 없다면, public network 에 있는 host A 가 private network 에 있는 host B 와 통신을 하고 싶을 때, host A 가 보낸 packet 은 NAT 에 의해서 drop 되게 된다. 또한 서로 다른 private network 에 존재하는 두명의 host 간에도 packet 을 제대로 보낼 수가 없다. 이러한 문제를 NAT traversal 이라고 한다. NAT traversal 문제를 해결하기 위해선 Relaying, Connection Reversal, UDP Hole Punching 와 같이 3 가지 방법이 존재한다.

Relaying 이란 말 그대로 packet data 를 중계하는 것이다. 이 방법은 private IP 를 갖고있는 두 host 간에 통신을 가능하게 만들고 public IP 를 가지는 server 을 통해서 동작된다. 먼저 host A 와 B 가 있다고 가정했을 때, A 가 B 로 packet 을 보내고 싶다면 외부 server 로 packet 을 보내고, server 가 그 packet 을 다시 NAT 을 통해 바뀐 B 의 public IP 에 전송해준다. 따라서 Relaying 은 NAT traversal 을 해결할 수 있지만, 외부 server 을 통해 packet 을 주고받기 때문에 overhead 가 크다는 단점이 있다.

Connection Reversal 은 두 host 중 한명이 private IP 이고, 다른 한명은 public IP 를 갖고 있는 경우에 통신이 가능하게 하는 방법이다. 먼저 host A 가 private IP 를 갖고있고 B 가 public IP 를 갖고 있고 외부 server 가 존재한다고 가정했을 때, A 가 server 로 register 정보를 보낸다. 그리고 B 가 server 로 A 와 통신하고 싶다는 정보를 넘기면 server 를 통해 A 가 B 로 연결 요청을 하도록 만들어 통신이 가능하게 한다. Connection Reversal 은 server 을 통해 패킷을 전송하지 않아 큰 overhead 가 없지만, 한 명의 host 는 public IP 를 갖고 있어야 한다는 단점이 있다.

UDP Hole punching 은 가장 효율적인 NAT traversal 방법이다. 먼저 이전과 같이 public IP 를 갖는 server 가 있고, 두 host A 와 B 가 private IP 를 갖고 있다고 가정했을 때, A 와 B 둘다 server 로 register 정보를 보낸다. 그 뒤로 A 가 B 와 통신하고 싶을 때, server 을 통해 B 의 public IP 를 알게 되고, B 도 server 을 통해 A 의 public IP 를 알게 된다. 따라서 두 host 는 routing 을 통해 서로 직접 통신할 수 있게 된다.

3. Explain the IPv6 addressing in terms of address space, representation, and difference from IPv4. (Within one page) (20points)

IPv6 는 IP 를 표현하기 위해서 128-bit 를 사용한다. 따라서 현재 가장 많이 사용되고 있는 IPv4 이 표현할 수 있는 IP 종류보다 훨씬 많은 IP 를 표현할 수 있다. 또한 IPv6dml datagram format 을 보면 크기가 40bytes 로 고정된 header 를 갖고 있으며, IPv4 보다 적은 종류의 정보로 구성돼 있다. 따라서 header 처리에 대한 overhead 가 줄어들기 때문에 빠른 processing 과 forwarding 이 가능하다. 또한 header 에 flow label 이 있어, network layer 에서 packet 의 flow 마다 처리할 수 있게 된다. 하지만 아직 모든 network 가 IPv6 를 사용하는 것은 아니므로, tunneling 을 통해서 IPv4 의 payload 에 IPv6 packet 을 넣어 IPv4 기반인 network 에서도 통신이 가능하게 만들 수 있다. tunneling 을 할 땐, IPv4 로 바뀌는 router 의 IP 를 src 로 지정하고, 끝나는 router 의 IP 를 dst 로 지정하여 IPv4 형태의 packet 을 만들어 전송한다.

※ For Q2 and Q3, you can search in the Internet. However, you should provide the answer in your language.

※ You can write the answer in Korean.