REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 : 컴퓨터 네트웍 개론
과 제 명 : HW 2
학 과 : 컴퓨터공학과
학 번 : 2013314967
이 름 : 정기빈

1. Development Environment

OS : Windows 7 64 bits

PL : C Language

Compiler Version : 14.0 (Visual 2015)

2. How to run

2.1 Sender

1) Sender를 먼저 접속 시킨다. Sender가 접속하지 않은 상황에서 Receiver 접속 시 Receriver가 종료되도록 프로그래밍하였다.

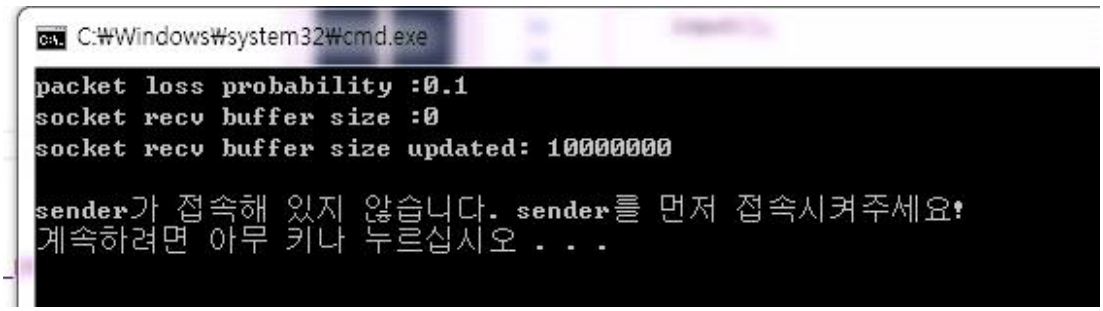

그림 1 Receiver 종료 화면

2) Sender 접속 시 window size, timeout, 전송할 패킷의 수를 설정해야 한다.

그림 전송할 패킷의 수

3) 그 후 패킷 전송을 시작할지 묻는 메시지가 나오는데, y를 누르면 패킷을 전송하기 시작한다. 만약 이때 receiver가 접속해있지 않다면 receiver가 접속할 때까지 기다린다.

그림 전송 시작 메시지

2.1 Receiver

1) Receiver를 접속하기 전에 Sender가 먼저 접속해있어야 한다. Sender가 접속하지 않은 상황에서 Receiver 접속 시 Receriver가 종료되도록 프로그래밍하였다.

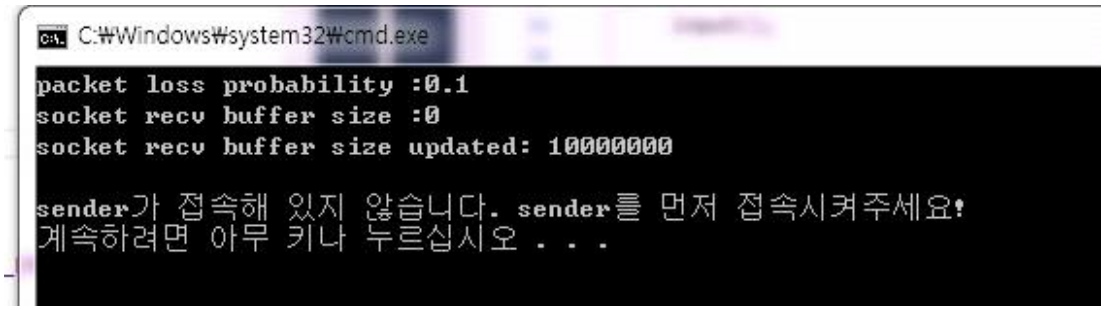

그림 4 Receiver 종료 화면

2) Receiver의 패킷 드랍율을 0.0~1.0의 수로 설정해야 한다.

그림 패킷 드랍율 설정

3) 모든 설정을 완료할 경우 소켓의 receive socket buffer의 사이즈를 출력한다. 만약 10MB 이하일 경우 10MB로 설정한 후 이를 사용자에게 알린다

그림 사이즈

4) 모든 과정이 끝나면 sender를 패킷을 전송하기 시작한다.

그림 패킷전송 시작

3. How to Design

3.1 Sender

1) 먼저, Sender는 소켓을 생성 후 bind한다

```
// 소켓 생성
send_soc = socket(PF_INET, SOCK_DGRAM, 0);
if (send_soc == INVALID_SOCKET)
    error_handle("socket() error!");

// 어드레스 설정
memset(&send_addr, 0, sizeof(send_addr));
send_addr.sin_family = AF_INET;
send_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
send_addr.sin_port = htons(12000);

if (bind(send_soc, (SOCKADDR*)&send_addr, sizeof(send_addr)) == SOCKET_ERROR)
    error_handle("bind() error!");
```

그림 소켓 생성과 bind

2) bind 후 Sender는 receive 용 쓰레드를 생성한다.

```
hThread = (HANDLE)_beginthreadex(NULL, 0, thread_func, 0, 0, &threadID);
```

그림 쓰레드 생성

3) Sender는 사용자로부터 windows size, timeout 등 설정값을 입력받은 후 receiver가 접속할 때까지 무한루프에서 대기한다. Is_connected 변수는 0으로 초기화 되어있는데, receiver로부터 접속 메시지를 받으면 이 필드를 1로 설정하여 무한루프에서 탈출한다.

int is_connected = 0;

```
if (input())
    return 0;

printf("waiting for connection.. \n");
while (!is_connected);
printf("connected!\n\n");
```

그림 접속 대기

4) 다음은 receive용 쓰레드 함수내 코드로 receiver 접속 메시지를 처리하는 부분이다. "connect"라는 내용의 메시지를 받으면 "accept"라는 메시지를 receiver측으로 회신 후 is_connected를 1로 설정한다.

```
nbyte = recvfrom(send_soc, rcv_buf, BUF_SIZE, 0, (struct sockaddr *)&recv_addr, &addrlen);
if (!strcmp(rcv_buf, "connect"))
{
    memcpy(send_buf, "accept", 7);
    if (sendto(send_soc, send_buf, 7, 0, (struct sockaddr*)&recv_addr, sizeof(recv_addr)) < 0)
        error_handle("sendto() error!");
    memcpy(rcv_buf, ".", 2);
    is_connected = 1;
    continue;
}
```

그림 접속 메시지 처리

5) Sender 설계의 가장 핵심은 다음 두 배열이다. acks 배열은 해당 seq_number의 ack메시지를 받았는지 확인하기 위한 배열이고, packets은 해당 seq_number의 패킷을 이미 전송했는지 알기 위한 배열이다.

```
int acks[10000000], packets[10000000];
```

그림 12 acks, packets 배열

6) pkt, tpkt 또한 Sender 설계를 위해 중요하다. pkt은 현재 Sender가 보내고 있는 패킷의 번호를 가리키는 변수이고, tpkt은 현재 window의 가장 작은 패킷의 번호를 가리킨다. tpkt에 해당하는 ack 번호를 받을 때 tpkt을 증가시킴으로써 window를 이동시킨다.

```
int pkt = 0, tpkt = 0;
```

그림 변수

7) 패킷 전송을 시작하면서 두 개의 타이머를 작동시킨다. start는 경과시간을

알기 위한 타이머이고, timer는 time out을 체크하기 위한 타이머이다.

```
start = clock();
timer = clock();
```

그림 14 두개의 타이머

8) 패킷전송을 하기 위해 반복문을 동작한다. 반복문의 종결 조건은 *(전송할 패킷의 수 - 1)*의 ack를 수신하는 순간이다.

```
while (!acks[pkt_num - 1])
{
```

그림 15 반복문 종료 조건

9) 반복문마다 타이머 경과 시간을 체크한다. 타이머가 제한 시간을 초과할 경우 이에 대해 출력한 후 pkt 을 tpkt으로 초기화 시킨다. 즉 window의 가장 작은 번호의 패킷부터 재전송하기 시작한다.

```
t = (clock() - timer) / (float)1000;
if (t > t_out)
{
    sprintf(temp, "\n%.3f pkt: %d | Timeout since %.3f\n\n", (clock() - start) / (float)1000, tpkt, ((clock() - start) / (float)1000) - t);
    printf(temp);

    pkt = tpkt;
    timer = clock();
}
```

그림 17 타이머 시간 초과

10) window size의 패킷들을 전송한다. 패킷전송은 그저 패킷의 번호를 전송하는 방식으로 구현하였다. 패킷 전송 시 packets 배열을 체크하여 재전송인지 확인하여 이를 출력한다. 패킷을 전송할 때 packets[i]를 1로 설정한다.

```
if (pkt < tpkt + w_size && pkt < pkt_num)
{
    t = (clock() - start) / (float)1000;
    sprintf(temp, "%.3f pkt: %d Sender > Receiver", t, pkt);
    if (packets[pkt]) strcat(temp, "<retransmitted>");

    printf("%s\n", temp);

    sprintf(send_buf, "%d\n", pkt);
    sendto(send_soc, send_buf, strlen(send_buf) + 1, 0, (struct sockaddr*)&recv_addr, sizeof(recv_addr));
    packets[pkt] = 1;
    pkt++;
}
```

그림 18 패킷 전송

11) receive용 쓰레드 함수이다.

- ack number -1을 수신하거나 acks[i]가 1일 경우(이미 수신한 경우), ack 수신 메시지만 출력하고 무시한다.
- acks[i]가 0일 경우(수신하지 않은 경우), tpkt을 증가시킴으로써 window를 이동시킨다.

```
else
{
    int rcv_ack = atoi(rcv_buf);
    if (rcv_ack == -1 || acks[rcv_ack])
    {
        t = (clock() - start) / (float)1000;
        sprintf(temp, "%.3f ACK: %d Sender < Receiver\n", t, rcv_ack);
        printf(temp);
    }

    else
    {
        acks[rcv_ack] = 1;

        t = (clock() - start) / (float)1000;
        sprintf(temp, "%.3f ACK: %d Sender < Receiver\n", t, rcv_ack);
        printf(temp);
        timer = clock();
        tpkt++;
    }
}
```

그림 패킷 전송

12) 마지막으로 *수신한 패킷 수/ 경과시간* 으로 throughput을 출력한다

```
t = (clock() - start) / (float)1000;
sprintf(temp, "%.3f | %d pkt trasmission completed. Throughput: %f pkts/sec\n", t, pkt_num, pkt_num / t);
printf(temp);
```

그림 20 througput 출력

3.2 Receiver

1) 소켓을 생성한 후 sender가 접속해 있는지 확인한다. sender가 접속해 있지 않을 경우 종료시킨다.

```
// 소켓 생성
sock = socket(PF_INET, SOCK_DGRAM, 0);
if (sock == INVALID_SOCKET)
    error_handle("socket() error!");
memset(&send_addr, 0, sizeof(send_addr));
send_addr.sin_family = AF_INET;
//send_addr.sin_addr.s_addr = htonl(INADDR_ANY);
send_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
send_addr.sin_port = htons(12000);

if (!check_sender())
{
    printf("sender가 접속해 있지 않습니다. sender를 먼저 접속시켜주세요!\n");
    return 0;
}
```

그림 21 소켓 생성과 sender 체크

2) 그 후 subf 함수를 호출하여 패킷 수신을 시작한다. 첫 패킷 수신 시 타이머를 작동시킨 후 경과시간을 출력한다. 수신한 패킷 번호를 출력한다.

```
int rcv_pkt = atoi(rcv_buf);
if (is_first)
{
    //assert(rcv_pkt == 0);
    start = clock();
    is_first = 0;
}

t = (clock() - start) / (float)1000;
sprintf(temp, "%.3f pkt: %d Receiver < Sender\n", t, rcv_pkt);
printf(temp);
```

그림 22 패킷 수신

3) 다음은 패킷 드랍율을 계산하는 식이다. 계산한 패킷 드랍율이 초기 입력값보다 낮을 경우 패킷을 드랍시킨다.

percent = rand() % 10000 / 100.f;

```
percent = rand() % 10000 / 100.f;
if (percent < (loss_prob * 100) )
{
    sprintf(temp, "%.3f pkt: %d | Dropped\n", t, rcv_pkt);
    printf(temp);
}
```

그림 22 패킷 드랍

4) receiver 설계에 가장 중요한 변수는 pkt 변수이다. 이 변수는 현재까지 수신한 패킷 번호 중 가장 큰 번호를 가리킨다. pkt은 처음에 수신한 패킷이

하나도 없으므로 -1로 초기화 된다.

```
int pkt = -1;
```

그림 23 pkt 변수

5) 정상적으로 pkt이 도착했으면 수신한 패킷 번호는 *pkt + 1* 이어야 한다. 그 경우에는 수신한 패킷 번호를 출력한 후 pkt을 증가시킨다.
아닐 경우에는 그냥 패킷 수신만을 출력한다.

```
if (pkt + 1 == rcv_pkt)
{
    sprintf(send_buf, "%d", rcv_pkt);
    sendto(sock, send_buf, strlen(send_buf) + 1, 0, (struct sockaddr*)&send_addr, sizeof(send_addr));

    t = (clock() - start) / (float)1000;
    sprintf(temp, "%.3f ACK: %d Receiver > Sender\n", t, rcv_pkt);
    printf(temp);
    pkt++;
}

else if (pkt + 1 < rcv_pkt)
{
    sprintf(send_buf, "%d", pkt);
    sendto(sock, send_buf, strlen(send_buf)+1, 0, (struct sockaddr*)&send_addr, sizeof(send_addr));

    t = (clock() - start) / (float)1000;
    sprintf(temp, "%.3f ACK: %d Receiver > Sender\n", t, pkt);
    printf(temp);
}
```

그림 24 패킷 수신

4. Experiment Results

4.1 loss probability

Window size 8, 시간 제한 1초에서 loss probability 변화에 따른 good put은 다음과 같다.

Loss probabilty가 낮을수록 goodput의 결과가 좋지 못했다.

그림 25 loss probability

4.2 window size

Loss probability 0.08, 시간 제한 1초에서 window size 변화에 따른 goodput의 결과는 다음과 같다.
Window size가 높을수록 오히려 goodput이 상대적으로 더 낮았다. 하지만 그렇게 loss probability의 경우처럼 큰 차이는 아니었다.

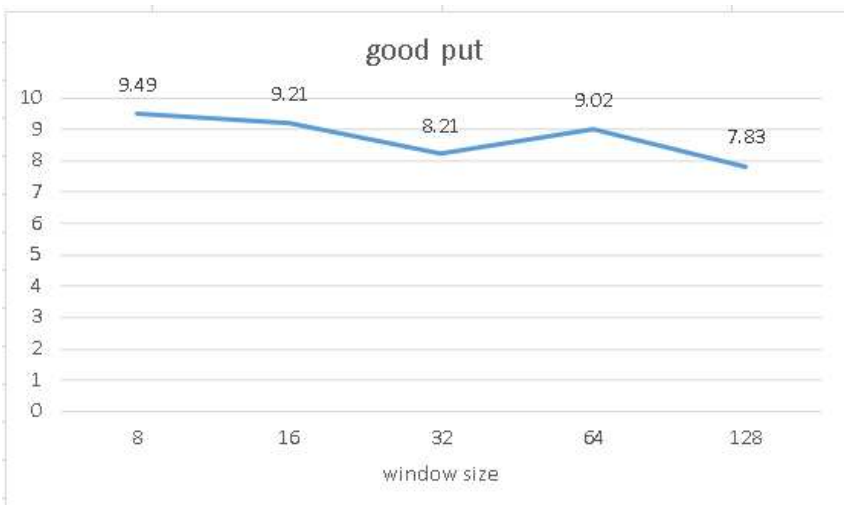

그림 26 window size