CprE 489, Section 4
Lab Experiment #3: TCP Sockets Programming

Sean Gordon & Noah Thompson

Experience:

This lab was an introduction to UDP socket programming. We used UDP sockets to listen to video data streamed from VLC to a server port. The server port then sent the video as a datagram to a client port. This was done using the c functions sendto() and recvfrom().

Exercises:

The two approaches for compiling server information to send to the client were:

1. Increasing the loss rate caused more data to be lost. Since UDP does not account for data loss, the received transmission was missing more frames as the loss rate was increased.
2. TCP would ensure that every single frame would be sent. While this may be ideal for critical data that needs to be complete, a streamed video does not require that every frame is transmitted, as the time taken to retransmit dropped packets would cause long buffering pauses when viewing the video. Some data loss may be necessary to ensure a smooth user experience in this particular case.

Effort:

Both group members provided even levels of effort towards the completion of the lab. Both members worked on the same program in tandem.

Sean: 50% || Noah: 50%

Code:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <netinet/in.h>
#include <linux/unistd.h>
#include <time.h>
      //1370 bytes packet size
int main(int argc, char** argv)
{

      if (argc == 6){ //Makes sure there are 6 arguments
            char * listener_IP = argv[1];
            char* listener_port = argv[2];
            char* dest_IP = argv[3];
            char* dest_port = argv[4];
            char* loss_rate = argv[5];

            char buffer[4000];
            char cliBuffer[4000];

            struct sockaddr_in serveraddr, clientaddr, dummy;
            u_short listener_port_int = atoi(listener_port);
            u_short dest_port_int = atoi(dest_port);
            int sersock, clisock;

            sersock = socket(PF_INET, SOCK_DGRAM, 0);
            if(sersock < 0){perror("socket: ");}
            serveraddr.sin_addr.s_addr = inet_addr(listener_IP);
            serveraddr.sin_port = htons(listener_port_int);
            serveraddr.sin_family = PF_INET;



            int loss_rate_int = atoi(loss_rate);

            clientaddr.sin_port = htons(dest_port_int);
            clientaddr.sin_addr.s_addr = inet_addr(dest_IP);
            clientaddr.sin_family = PF_INET;
```

```
            printf("Client port: %d\nServer Port: %d\n",
clientaddr.sin_port, serveraddr.sin_port);


            //while(1){
            int cliLen = sizeof(clientaddr);
            int serLen = sizeof(serveraddr);

            clisock = socket(PF_INET, SOCK_DGRAM, 0);
            if(clisock < 0){perror("Clisock: ");}

            int bindTest = bind(sersock, (struct sockaddr *) &serveraddr,
sizeof(serveraddr));
            if (bindTest != 0){ perror("Bind: ");}



            while(1){

                  int receive = recvfrom(sersock, buffer, 4000, 0, NULL,
NULL);
                  if (receive < 0){perror("Rec: ");}
                  else if(receive == 1){printf("Received\n");}

                  int randomnum = (rand() % 100) + 1;
                  printf("Rand: %d\nLoss: %d\n ", randomnum,
loss_rate_int);
                  if (randomnum > loss_rate_int)
                  {
                        int serLen = sizeof(serveraddr);

                        int sendNum = sendto(clisock, buffer,
sizeof(buffer), 0, (struct sockaddr *) &clientaddr, serLen);
                        if (sendNum < 0){perror("send: ");}

                  }
            }
      }
}
```