Bruce Bitwayiki
Thomas Ruminski

CprE489 Lab #3 Report (Section 3)
UDP Socket Programming

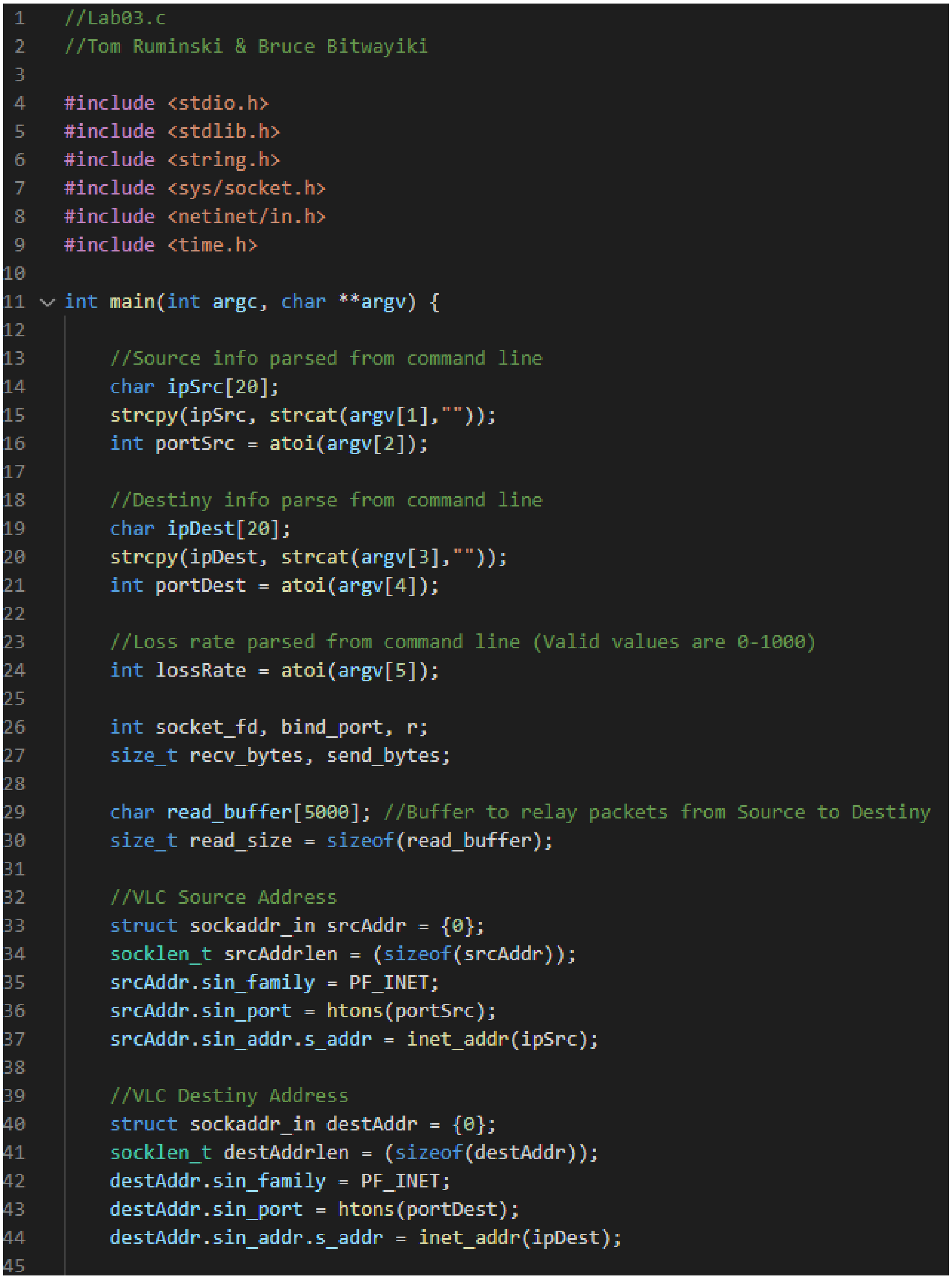
Summary

In this lab we further explore network sockets, primarily UDP sockets. The goal is to write a program that would forward UDP packets from a given source to a given destination while introducing packet loss. We begin by implementing a socket and binding it to an IP address and port, the process is very similar to TCP sockets. Our program stores source and destination addresses and ports through command line arguments. Once the original socket is binded; we enter an infinite while loop which allows us to keep forwarding packets until the user ends the transmission.
Once packets are received, we decide whether or not to transmit the packets to the destination port and address based on a specific loss rate implemented using a random number generator. Last but not least, we tested our program using VLC media player by having two windows and streaming the video as data packets from one port to another.

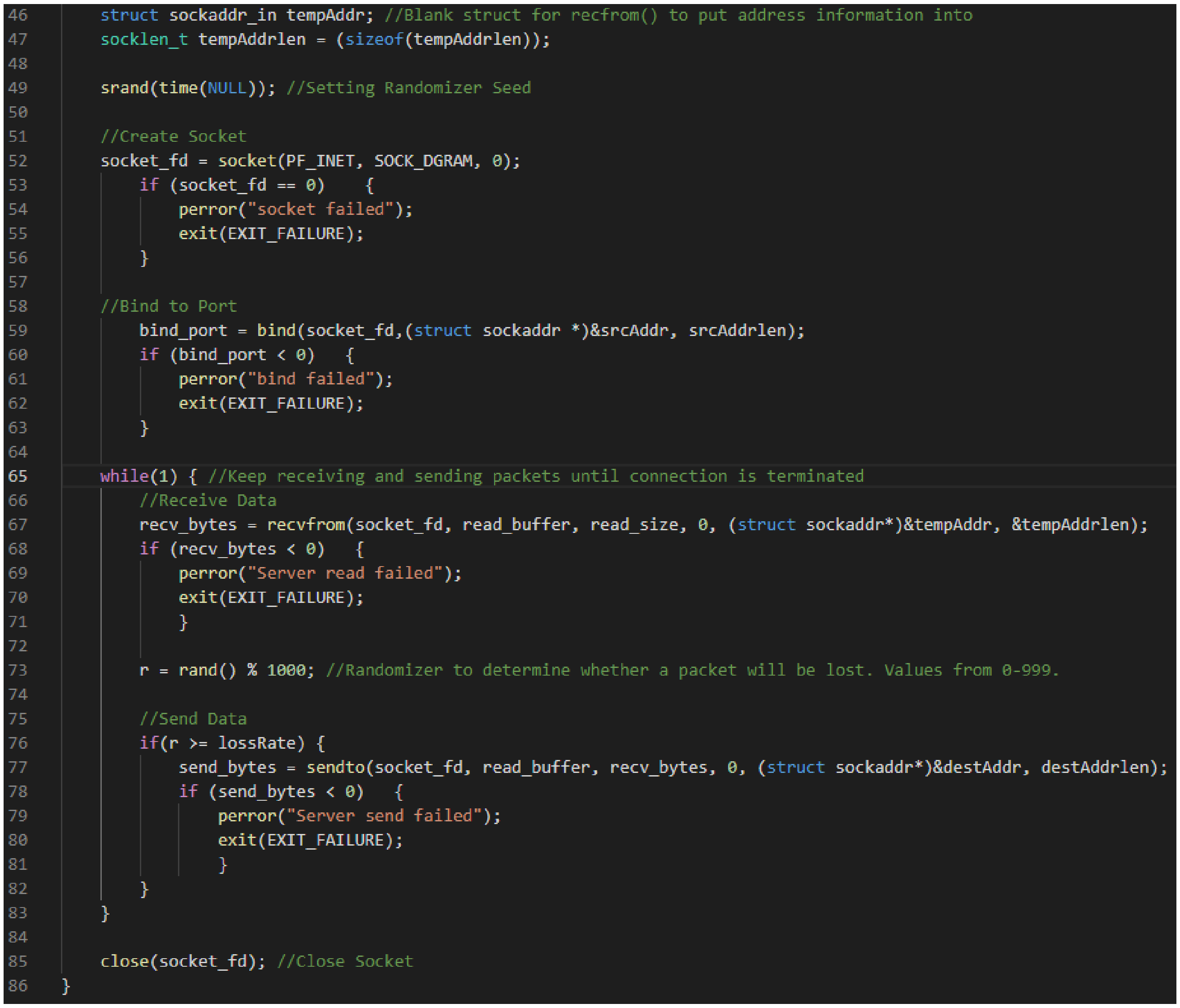
Exercises

1. **What was the effect of increasing the loss rate on the video quality?**
   By increasing the loss rate, we noticed significant damage to video quality, this included jittery images and total stops at times.
2. **To counter the effect of network loss, someone suggested using TCP instead of UDP for**
   multimedia communication. Do you agree? Why? Why not? (Answer the question in terms of
   buffer space, jitter, and retransmission time.)
   TCP would help our use case as we would have a better image due to the handshake that occurs through TCP. However, since we purposely “lost” the package, TCP would request retransmission of the lost packets which would lead to extended retransmission time, and possibly additional video jitter. So we would increase image/video quality using UDP but at the expense of greater transmission time.

Well Commented Code



```c
//Lab03.c
//Tom Ruminski & Bruce Bitwayiki

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <time.h>

int main(int argc, char **argv) {

    //Source info parsed from command line
    char ipSrc[20];
    strcpy(ipSrc, strcat(argv[1],""));
    int portSrc = atoi(argv[2]);

    //Destiny info parse from command line
    char ipDest[20];
    strcpy(ipDest, strcat(argv[3],""));
    int portDest = atoi(argv[4]);

    //Loss rate parsed from command line (Valid values are 0-1000)
    int lossRate = atoi(argv[5]);

    int socket_fd, bind_port, r;
    size_t recv_bytes, send_bytes;

    char read_buffer[5000]; //Buffer to relay packets from Source to Destiny
    size_t read_size = sizeof(read_buffer);

    //VLC Source Address
    struct sockaddr_in srcAddr = {0};
    socklen_t srcAddrlen = (sizeof(srcAddr));
    srcAddr.sin_family = PF_INET;
    srcAddr.sin_port = htons(portSrc);
    srcAddr.sin_addr.s_addr = inet_addr(ipSrc);

    //VLC Destiny Address
    struct sockaddr_in destAddr = {0};
    socklen_t destAddrlen = (sizeof(destAddr));
    destAddr.sin_family = PF_INET;
    destAddr.sin_port = htons(portDest);
    destAddr.sin_addr.s_addr = inet_addr(ipDest);

```



```c
    struct sockaddr_in tempAddr; //Blank struct for recfrom() to put address information into
    socklen_t tempAddrlen = (sizeof(tempAddrlen));

    srand(time(NULL)); //Setting Randomizer Seed

    //Create Socket
    socket_fd = socket(PF_INET, SOCK_DGRAM, 0);
        if (socket_fd == 0)    {
            perror("socket failed");
            exit(EXIT_FAILURE);
        }

    //Bind to Port
        bind_port = bind(socket_fd,(struct sockaddr *)&srcAddr, srcAddrlen);
        if (bind_port < 0)   {
            perror("bind failed");
            exit(EXIT_FAILURE);
        }

    while(1) { //Keep receiving and sending packets until connection is terminated
        //Receive Data
        recv_bytes = recvfrom(socket_fd, read_buffer, read_size, 0, (struct sockaddr*)&tempAddr, &tempAddrlen);
        if (recv_bytes < 0)   {
            perror("Server read failed");
            exit(EXIT_FAILURE);
            }

        r = rand() % 1000; //Randomizer to determine whether a packet will be lost. Values from 0-999.

        //Send Data
        if(r >= lossRate) {
            send_bytes = sendto(socket_fd, read_buffer, recv_bytes, 0, (struct sockaddr*)&destAddr, destAddrlen);
            if (send_bytes < 0)   {
                perror("Server send failed");
                exit(EXIT_FAILURE);
                }
        }
    }

    close(socket_fd); //Close Socket
}
```