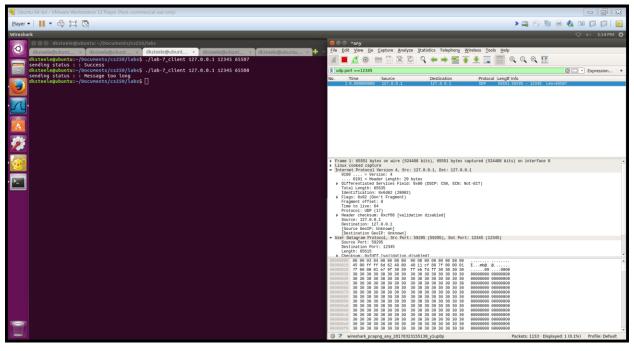
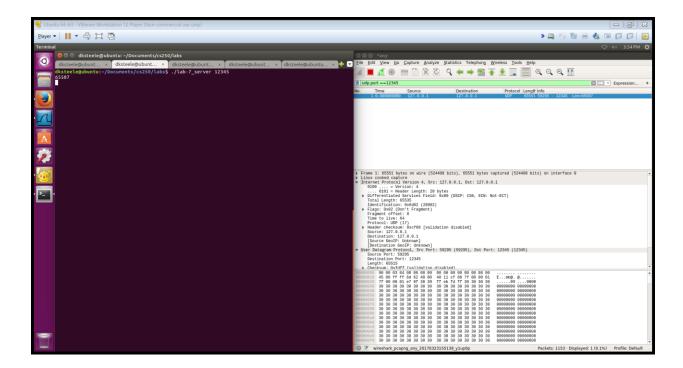
Daniel Steele Lab #7

Verify experimentally the size of the largest datagram you can send and receive using a UDP socket (without fragmentation). What is the size? What happens if the datagram is larger than the limit?

My testing indicated that I could on my system send a UDP packet of 65507 bits but was not able to exceed that limit. After this limit the messages would have to fragmented to be able to send. I believe that the way that the it is implemented such that oversize messages cannot be sent is important to make sure that a receiver of the data does not receive incomplete data. I also imagine that this number may differ between system dependent on hardware and operating system. As seen below that information was both sent and received successfully but I was not able to exceed the bounds of 65507 bits of data.





The code to run the test is below.

```
Player ▼ ■ ▼ 🖧 💢
                                                                                                                                                                                                                                                                               » 👊 🕤 🖫 🔒 🐧 🗵 🗊 🗊
Text Editor
                         lab-7_client.c
<sys/socket.h>
<netinet//
                                                                                                                                                                           int main(int argc, char* argv[]){
            nt main(int argc, char* argv[]){
                                                                                                                                                                                       int s;
struct sockaddr_in addr, cli_addr;
socklen_t slen=stzeof(cli_addr);
                       int s;
struct sockaddr_in addr;
 P
                      if(argc != 4){
    printf("Usage: %s <server_ip_address> <server_port> <number_of_bytes_to_send>\n", argv
                                                                                                                                                                                       tf(argc != 2){
    printf("Usage: %s <server_port>\n", argv[0]);
    return 0;
            0]);
                                                                                                                                                                                       memset(&addr, '0', sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(atoi(argv[i]));
addr.sin_addr.s_addr = htoni(INADDR_ANY);
                      memset(&addr, '0', sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(atoi(argv[2]));
                      tf(inet_pton(AF_INET, argv[1], &addr.sin_addr) < 0){
    printf("Error Parsing Ip Address: %s\n", argv[1]);
    return 0;</pre>
                                                                                                                                                                                      tf((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0){
    printf("Error Creating Socket\n");
    return 0;
}</pre>
                                                                                                                                                                                      tf(bind(s, (struct sockaddr* ) &addr, streef(addr)) < 0){
    printf("Error Binding Socket\n");
    return 0;
}</pre>
                      char *out = (char *) malloc(streof(char) * atol(argv[3]));
nemset(&out[0], '0', atol(argv[3]));
sendto(s, out, atol(argv[3]), 0, (struct sockaddr*) &addr, streof(addr));
perror('sending status : ");
close(s);
                                                                                                                                                                                      white(1){
    char *in = (char*) malloc(sizeof(char) * INT_MAX);
    int n = recvfron(s, in, INT_MAX, 0, (struct sockaddr*) &cli_addr, &slen);
    printf("%d\n", n);
}
                      return 1;
                                                                                                                                                                                      close(s):
                                                                                                                                                                                       return 1;
```