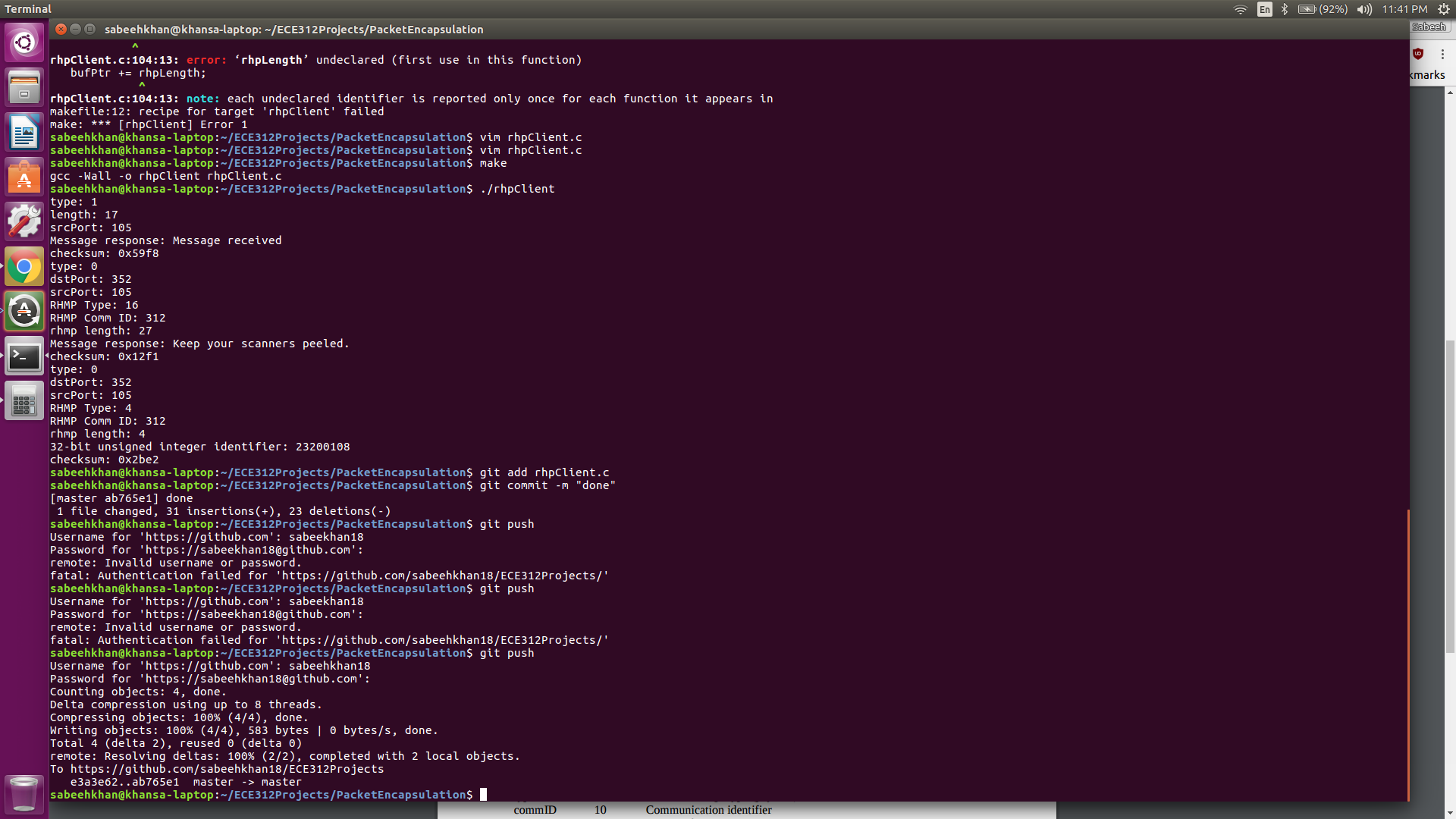
**Sabeeh Khan & Nick Kiesel**

**ECE 312-01**

**Project 2: Packet Encapsulation**

**Figure 1: Send of 3 Types of Packets and Results**



**Code for rhpClient.c:**

/\*\*\*\*\*\*\*\*\*\*\*\*\* UDP CLIENT CODE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <string.h>

#include <unistd.h>

#define SERVER "137.112.38.47"

#define MESSAGE "hello"

#define PORT 1874

#define BUFSIZE 1024

uint16\_t getChecksum(char\* data, int numBytes) {

uint16\_t\* checkPtr = (uint16\_t\*) data;

uint32\_t sum = 0;

for (int i = 0; i < numBytes/2; i++) {

sum += \*checkPtr;

if (sum >= 0x10000) {

sum = sum & 0xFFFF;

sum += 1;

}

checkPtr++;

}

return (0xFFFF - (uint16\_t)sum);

}

int recvRHP(int socket) {

char buffer[BUFSIZE];

int nBytes, pad = 0;

uint8\_t\* bufPtr;

uint8\_t type, rhmpLength;

uint16\_t rhpLength;

memset(buffer, 0, BUFSIZE);

/\* Receive message from server \*/

nBytes = recvfrom(socket, buffer, BUFSIZE, 0, NULL, NULL);

uint16\_t recvChecksum = \*((uint16\_t\*)(buffer+nBytes-2));

uint16\_t calcChecksum = getChecksum(buffer, nBytes-2);

if (recvChecksum != calcChecksum) {

printf("CHECKSUM MISMATCH\n");

return 1;

}

bufPtr = (uint8\_t\*) buffer;

printf("type: %d\n", \*bufPtr);

type = \*bufPtr;

bufPtr++;

if (type == 1) {

rhpLength = \*((uint16\_t\*)bufPtr);

printf("length: %d\n", \*((uint16\_t\*)bufPtr));

}

else

printf("dstPort: %d\n", \*((uint16\_t\*)bufPtr));

if (type == 1) {

if (\*((uint16\_t\*)bufPtr) % 2 == 0) {

pad = 1;

printf("adding buffer on received data\n");

}

} else {

bufPtr+=6;

rhmpLength = \*bufPtr;

if (rhmpLength % 2 != 0)

pad = 1;

bufPtr-=6;

}

bufPtr+=2;

printf("srcPort: %d\n", \*((uint16\_t\*)bufPtr));

bufPtr += 2;

if (type == 0) {

uint16\_t rhmpTypeCommID;

memcpy(&rhmpTypeCommID, bufPtr, 2);

uint8\_t rhmpType = (rhmpTypeCommID & 0x003F);

printf("RHMP Type: %d\n", rhmpType);

uint16\_t commID = (rhmpTypeCommID & 0xFFC0) >> 6;

printf("RHMP Comm ID: %d\n", commID);

bufPtr+=2;

printf("rhmp length: %d\n", rhmpLength);

bufPtr++;

if (rhmpType == 1) {

} else if (rhmpType == 2) {

} else if (rhmpType == 4) {

uint32\_t num = \*((uint32\_t\*)bufPtr);

printf("32-bit unsigned integer identifier: %d\n",num);

bufPtr+=4;

} else if (rhmpType == 8) {

} else {

printf("Message response: %s\n",bufPtr);

bufPtr += rhmpLength;

}

} else {

printf("Message response: %s\n", bufPtr);

bufPtr += rhpLength;

}

bufPtr += pad;

printf("checksum: 0x%04x\n", \*((uint16\_t\*)bufPtr));

return 0;

}

int sendRHP(int socket, uint8\_t type, char\* toSend, uint32\_t length) {

char buffer[BUFSIZE];

uint16\_t dstPort\_Length = 0;

int includePad = 0;

struct sockaddr\_in serverAddr;

memset(buffer, 0, BUFSIZE);

uint8\_t\* bufPtr = (uint8\_t\*) buffer;

\*bufPtr = type;

bufPtr++;

if (type == 0) {

dstPort\_Length = 105;

} else {

dstPort\_Length = length;

}

memcpy(bufPtr, &dstPort\_Length, 2);

bufPtr += 2;

uint16\_t srcPort = 352;

memcpy(bufPtr, &srcPort, 2);

bufPtr+=2;

memcpy(bufPtr, toSend, length);

bufPtr+=length;

includePad = 0;

if (length % 2 == 0) {

includePad = 1;

bufPtr++;

}

uint16\_t checksum = getChecksum(buffer, 5+length+includePad);

memcpy(bufPtr, &checksum, sizeof(checksum));

/\* Configure settings in server address struct \*/

memset((char\*) &serverAddr, 0, sizeof (serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(PORT);

serverAddr.sin\_addr.s\_addr = inet\_addr(SERVER);

memset(serverAddr.sin\_zero, '\0', sizeof serverAddr.sin\_zero);

/\* send a message to the server \*/

if (sendto(socket, buffer, 7+length+includePad, 0,

(struct sockaddr \*) &serverAddr, sizeof (serverAddr)) < 0) {

perror("sendto failed");

return 1;

}

return 0;

}

int main() {

int clientSocket, result;

char buffer[BUFSIZE] = {0};

struct sockaddr\_in clientAddr;

/\*Create UDP socket\*/

if ((clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("cannot create socket");

return 0;

}

/\* Bind to an arbitrary return address.

\* Because this is the client side, we don't care about the address

\* since no application will initiate communication here - it will

\* just send responses

\* INADDR\_ANY is the IP address and 0 is the port (allow OS to select port)

\* htonl converts a long integer (e.g. address) to a network representation

\* htons converts a short integer (e.g. port) to a network representation \*/

memset((char \*) &clientAddr, 0, sizeof (clientAddr));

clientAddr.sin\_family = AF\_INET;

clientAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

clientAddr.sin\_port = htons(0);

if (bind(clientSocket, (struct sockaddr \*) &clientAddr, sizeof (clientAddr)) < 0) {

perror("bind failed");

return 0;

}

// send RHP with hello

do {

result = sendRHP(clientSocket, 1, "hello\0", 6);

} while ((result = recvRHP(clientSocket)) != 0);

memset(buffer, 0, BUFSIZE);

uint32\_t data = 8;

data = data | (312 << 6);

memcpy(buffer, &data, 3);

// send RHMP with message request

do {

result = sendRHP(clientSocket, 0, buffer, 3);

} while ((result = recvRHP(clientSocket)) != 0);

data = 2;

data |= (312 << 6);

memset(buffer, 0, BUFSIZE);

memcpy(buffer, &data, 3);

// send RHP with ID request

do {

sendRHP(clientSocket, 0, buffer, 3);

} while (recvRHP(clientSocket) != 0);

return 0;

}