1   Introduction and Environment Setup

Our setup for the environment is a fresh install of Ubuntu 14.04 x64. There were no special libraries used in the making of the application. In order to run the program, the following steps should be taken:

- Unzip the project ZIP folder to your desired location

- Run *./build_script.sh* which will compile the program using gcc with all required parameters

- Run the *proj.out* executable using sudo along with arguments

- If no arguments are given, the program will output the correct usage

A sample run:

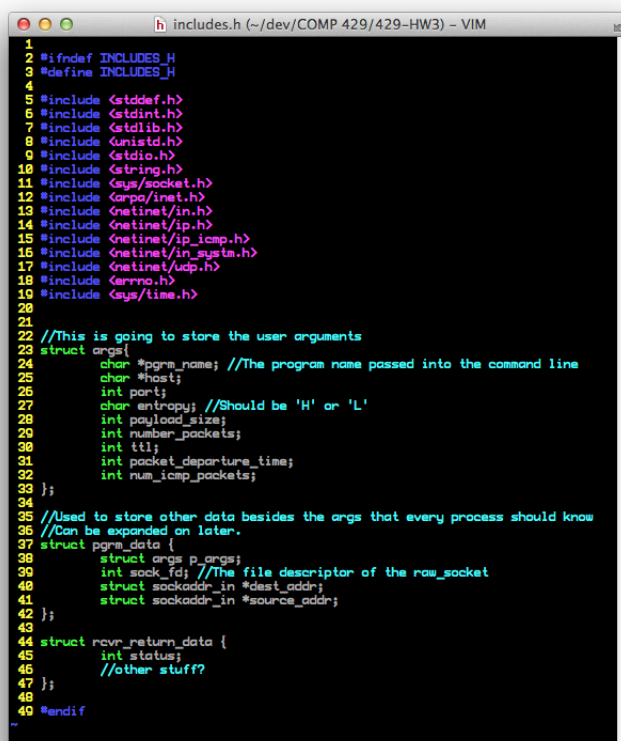- sudo *./project2.out* 8.8.8.8 9876 *L* 1100 6000 255 10 50

2   Challenges

A big problem that we faced during this project was the issue of cross compatbility between operating systems. Varatep Buranintu was developing on OS X, John-Luke Laue on Linux 14.04, and Jeffrey Limbacher on Windows. This was due to a lack of consideration for compatibility on Linux and we encountered some issues with the *udphdr* struct having more than one definition and there was ambiguity when switching the source files to a different operating system.

3   Correct Implementations

4   Problematic Implementations

5   Design Decisions

The greatest design decision we had to conclude on was how to split the work up. For a project like this, modularity is key in developing a reusable codebase with independent segments in itself allowing multiple developers to work simultaneously on similar parts of the program without impeding each other's work. We agreed that the application would use multithreading in order to provide a seamless user experience and performance limiting possible gridlocks. The first thing we did was creating a header file called *"includes.h"* that would import all of the required C standard libraries as well as act as the interface for our common program data. Such program data includes the Sender and Receiver functions *sendto* and *recvfrom* on the same socket. Another common program data used is the idea that both the Sender and Receiver need to be aware of the entropy type. Our user arguments struct was also stored in the *"includes.h"* file so that we would not have to re-declare or re-define it for every file in which it is used.

```
1
2  #ifndef INCLUDES_H
3  #define INCLUDES_H
4
5  #include <stddef.h>
6  #include <stdint.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <stdio.h>
10 #include <string.h>
11 #include <sys/socket.h>
12 #include <arpa/inet.h>
13 #include <netinet/in.h>
14 #include <netinet/ip.h>
15 #include <netinet/ip_icmp.h>
16 #include <netinet/in_systm.h>
17 #include <netinet/udp.h>
18 #include <errno.h>
19 #include <sys/time.h>
20
21
22 //This is going to store the user arguments
23 struct args{
24         char *pgrm_name; //The program name passed into the command line
25         char *host;
26         int port;
27         char entropy; //Should be 'H' or 'L'
28         int payload_size;
29         int number_packets;
30         int ttl;
31         int packet_departure_time;
32         int num_icmp_packets;
33 };
34
35 //Used to store other data besides the args that every process should know
36 //Can be expanded on later.
37 struct pgrm_data {
38         struct args p_args;
39         int sock_fd; //The file descriptor of the raw_socket
40         struct sockaddr_in *dest_addr;
41         struct sockaddr_in *source_addr;
42 };
43
44 struct rcvr_return_data {
45         int status;
46         //other stuff?
47 };
48
49 #endif
```

5.1   Sender

We designated the Sender to stay on the main thread as this is the core of our project that requires the most attention from the operating system. The sender is broken up into two categories: send functions *("sender.c")* and fill-out functions *("network.c")*.

Sender

include <network.c>

sender.c   network.c
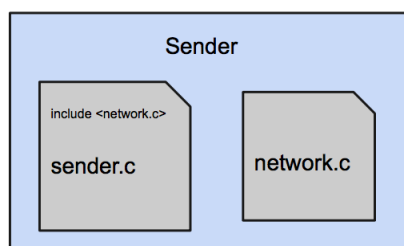
5.2   Receiver

John-Luke Laue intended the Receiver to have the same design as the Sender, although he quickly realized that this was unnecessary as a result of unpacking not requiring a multitude of functions in order to work properly.

Receiver

receiver.c
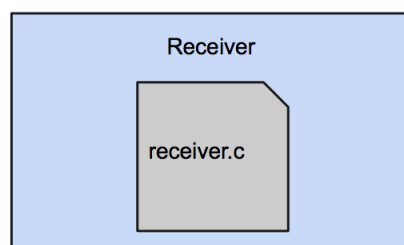
6   Project Hindsight

7   Allocation of Work

The collaborators of project collectively decided on how the work would be split up. The individuals of the team came up with different designs for the task, but we conjointly decided to capitalize on Jeffrey Limbacher's well-thought architecture. Upon general completion of the architecture design, we realized the significance of modularity in this project. Since there are two main hubs (the sender and the receiver) in this project, it made sense to assign at least one person per hub. John-Luke Laue was assigned the task of bringing together the receiver by himself, since this part was relatively straightforward. Varatep Buranintu and Jeffrey Limbacher modularly designed the sender hub and pooled resources together. Varatep Buranintu was responsible for the ICMP (head and tails) segment whilst Jeffrey Limbacher developed the UDP segment. Jeffrey Limbacher also built the raw socket and IP header as his tasks. Varatep Buranintu generated the project documentation and ensured the source code documentation (comments) is superlative so that another developer would be able to pick up where the project was left off.