Zergmap Design Plan

Jonathan Butler, Adam Matthes, Mark Workman

January 31st, 2022

1 Introduction

The Zerg, uplifted race of the Xel'Naga, communicate via latent psychic abilities. Surprisingly, these communications can be picked up via psychic antennae and recorded as standard UDP traffic destined for port 3751.

This document describes a design plan to write a program (Zergmap) that will work with the Psychic Captures (PCAPs) of network traffic involving the Zerg fell race. When given a PCAP file, Zergmap will determine where the Zerg are located on the battlfield. The program will recommend a list of Zerg to kill in order to form a fully-connected network of the survivors.

2 Features Targeted

2.1 Man Page

Write man(1) pages to document the program.

2.2 Big Endian

Support Big-endian `.pcap` formats.

2.3 Additional Protocol Support

Add support for one of the following: `IP Options`, `Ethernet 802x` optional headers.

2.4 Other Additional Protocol Support

Add support for one of the following: `4in6`, `Teredo`, or `6in4`.

3 Architecture

3.1 Data

`Zergmap` will be required to accept multiple `.pcap` files via the command line and parse each `.pcap` file. It will support parsing both Big and Little-Endian .pcap file headers. Zergmap will read in Big Endian formatted network data and convert it to Little-Endian format for proccessing.

`Zergmap` will incorporate the use of multiple data structures. The most notable of these data structures will be a directed graph and a `Octree`. The `Octree` will be used to store and retrieve multi-dimensional spatial data of a given Zerg (e.g. latitude, longitude, altitude), while the graph will be used to store and represent a fully connected Zerg network. A depiction of the file structure and structs used in `Zergmap` are depicted Figures 1, 2, and 3.

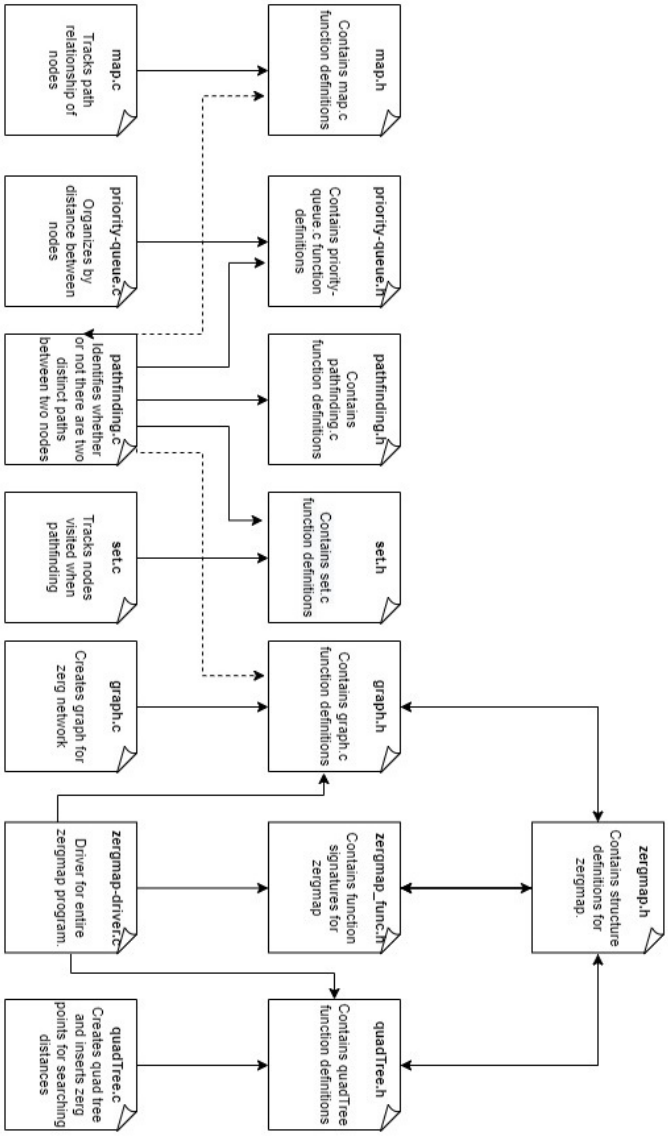

Figure 1: Zergmap Structs

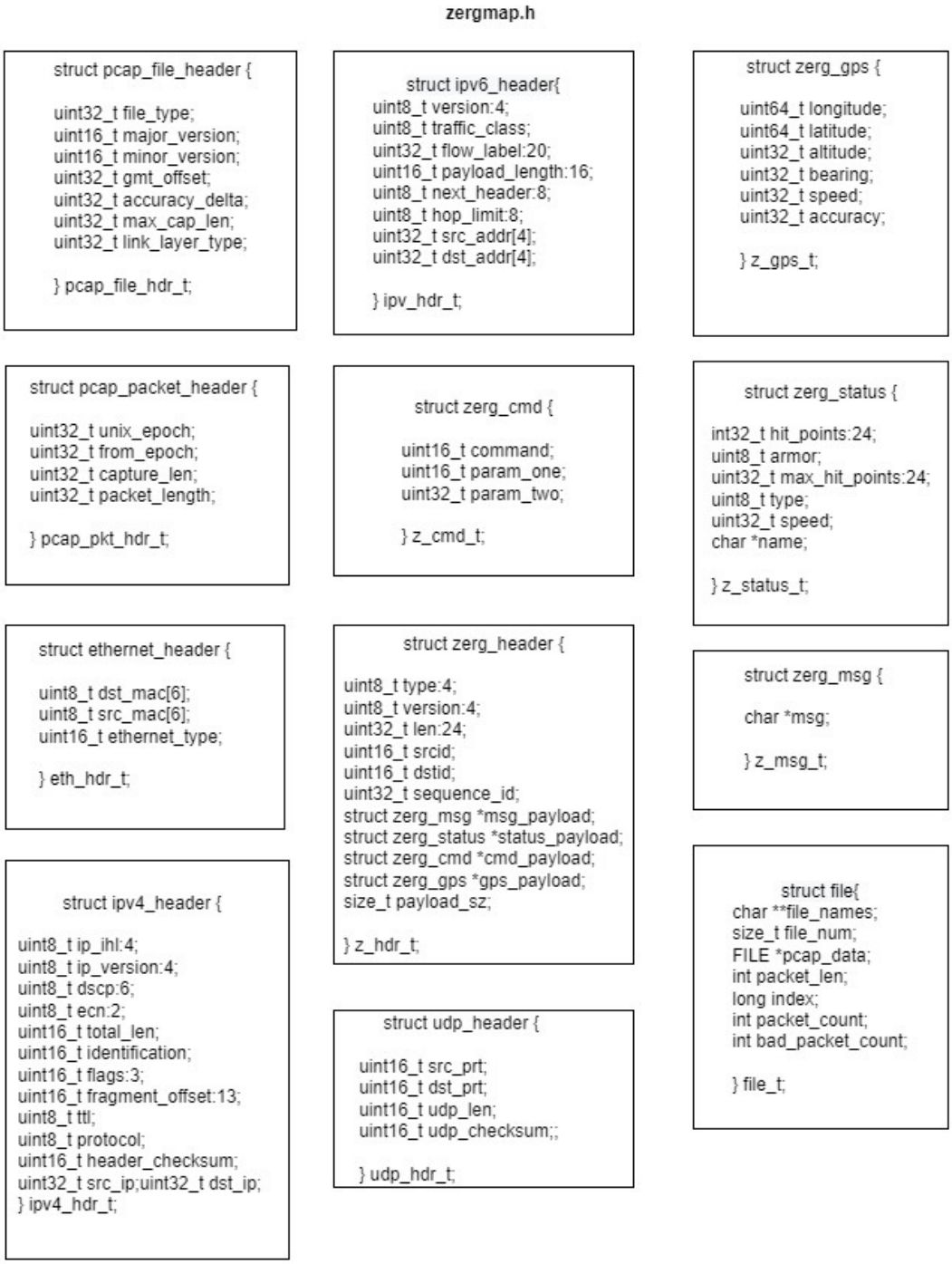

Figure 2: Zergmap.h Header File Structs

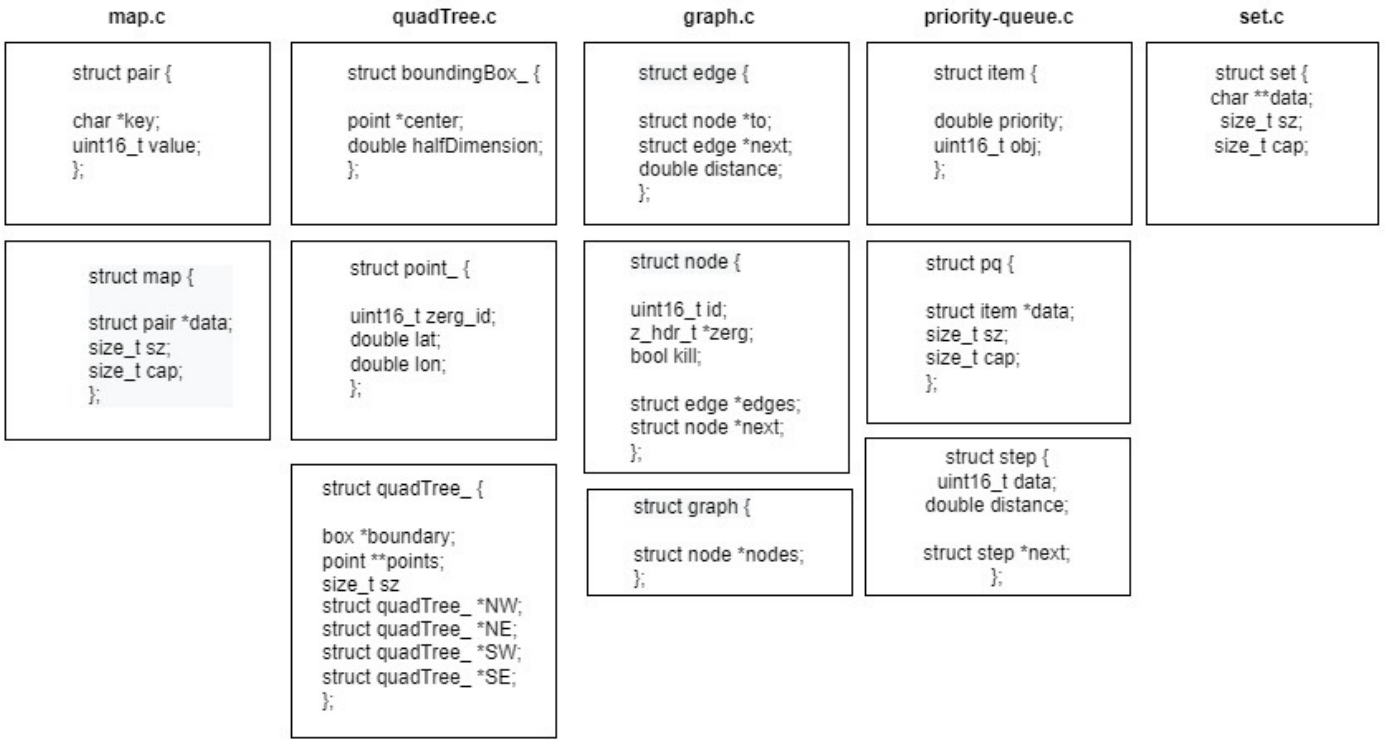

Figure 3: Map, QuadTree, Graph, Priority Queue, Set Header File Structs

4 Significant Functions

4.1 zergmap-driver.c

1. `int main (argc, argv[])`

   `Main` is the driving function for Zergmap. `Main` will parse all command line arguments, as well as the `.pcap` files that are provided by the user. `main` will also conduct command line argument input validation, as well as file input validation.

4.2 zergmap.c

1. `char **file_names(struct strings_array *sa, char *argv[])`
   Dynamically allocates memory for an array of strings and stores all `.,pcap` names passed at the command line within each element of the array.

2. `int file_hdr_parse(file_t * file, pcap_file_hdr_t * pcap_fh)`
   Populates appropriate `.pcap` file header struct fields with values read in from `.pcap` file.

3. `int packet_hdr_parse(file_t * file, pcap_pkt_hdr_t * pcap_ph, eth_hdr_t * eth, ipv4_hdr_t * ipv4 , udp_hdr_t * udp, z_hdr_t * zhdr, pcap_file_hdr_t * pcap_fh)`
   Populates appropriate `.pcap` packet, `ethernet`, `IPv4`, and `UDP` struct fields with values read in from `.pcap` files.

4. `void print_breed(z_hdr_t * zhdr, file_t * file)`
   Converts Zerg breed type ID to Zerg breed string name.

5. `int zerg_gps_parse(file_t * file, z_hdr_t * zhdr)`
   Parses `.pcap` file, converts values from Big to Little Endian, populates Zerg GPS struct fields, and prints values to `stdout`.

6. `int zerg_status_parse(file_t * file, z_hdr_t * zhdr)`
   Parses `.pcap` file, converts values from Big to Little Endian, populates Zerg status struct fields, and prints values to `.stdout`.

7. `int zerg_command_parse(file_t * file, z_hdr_t * zhdr)`
   Parses `.pcap` file, converts values from Big to Little Endian, populates Zerg Command struct fields, and prints values to `stdout`.

8. `int zerg_msg_parse(file_t * file, z_hdr_t * zhdr)`
   Parses `.pcap` file, converts values from Big to Little Endian, populates Zerg Message struct field, and prints values to `stdout`.

4.3 graph.c

1. `graph *graph_create(int (*cmp)(const void *, const void *))`
   Creates and returns a graph object.

2. `bool graph_add_vertex(graph *g, void *data)`
   Creates vertexes and stores appropriate data within graph nodes.

3. `bool graph_add_edge(graph *g, const void *src, const void *dst, double weight)`
   Contains logic to iterate over all vertexes and add appropriate edges between vertexes to build out fully connected Zerg network.

4. `double graph_edge_weight`(const graph *g, const void *src, const void *dst)—
   Adds weight to each edge within graph.

5. `bool graph_contains(const graph *g, const void *data)`
   Searches for a given piece of data within the graph.

6. `bool graph_destroy(graph *g)`
   Iterates over graph vertexes and edges and frees all dynamically allocated memory.

4.4 pathfinding.c

1. `void graph_surrbale(graph *g)`
   Utilizes Surrbale algorithm to find the two shortest paths between two Zergs.

4.5 quadTree.c

1. `point *create_point(double lat, double lon)`
   Creates and returns a point object.

2. `point *create_zerg_point(z_hdr_t *z_hdr)`
   Creates and returns a point object for Zerg points.

3. `void point_print(point *p)`
   Prints the point object to `stdout`.

4. `box *create_box(point *center, double halfDimension)`
   Creates and returns a new bounding box object.

5. `bool box_contains_point(box *boundary, point *point)`
   Returns true or false depending on if the point is located in the bounding box.

6. `bool box_intersects(box *self, box *check)`
   Returns true or false if one bounding box intersects the second bounding box.

7. `qt *create_quadTree(box *boundary)`
   Creates and returns a new `quadTree` object.

8. `size_t number_of_points(qt *qt)`
   Returns a `size_t` count of how many points are in the `quadTree`.

9. `qt *subdivide(qt *root)`
   This function is used with the `quadTree` to divide into regions.

10. `bool qt_insert(qt *root, point *point)`
    Returns true or false if a point is inserted into the `quadTree`.

11. `point **quadTree_search(qt *root, box *range)`
    Returns an array of point objects that are in the specified search box.

12. `void point_destroy(point *p)`
    Frees a point object.

13. `void box_destroy(box *b)`
    Frees a box object.

14. `void qt_destroy(qt *qt)`
    Frees a `quadTree` object.

5 Command Line Arguments

The program will be designed to accept multiple `.pcap` files which are passed as command line arguments. If too many or too few arguments are passed, a usage message will be displayed. If a PCAP file of the incorrect type (e.g. anything but version 2.4) is passed, a usage message will be displayed.

6 Developmental Approach

Given that this is a group project, development will tasks will be executed simultaneously. All development tasks will be grouped within three Lines of Effort (LOEs), with each LOE being executed concurrently. Within each LOE, each task will be executed in priority order as listed below. The LOEs are as follows: LOE 1 (Graph), LOE 2 (Tree), and LOE 3 (Decode IPV6 / Suggested Features) .

6.1 LOE 1: Graph

1. `graph_create`: Develop `graph_create`.
2. `graph_add_vertex`: Develop the `graph_add_vertex`.
3. `graph_add_edge`: Develop `graph_add_edge`.
4. `graph_add_weight`: Develop `graph_add_weight`.
5. `graph_contains`: Develop `graph_contains`.
6. `graph_destroy`: Develop `graph_destroy`.
7. `graph_surrbale`: Develop `graph_surrbale`.

6.2 LOE 2: Tree

1. `point_create`: Develop `point_create` function.
2. `point_print`: Develop `point_print` function.
3. `create_box`: Develop `create_box` function.
4. `box_contains`: Develop `box_contains` function.
5. `box_intersects`: Develop `create_quadTree` function.
6. `create_box`: Develop `create_box` function.
7. `qt_insert`: Develop `qt_insert` function.
8. `subdivide`: Develop `subdivide` function.
9. `qt_destroy`: Develop `qt_destroy` function.

6.3 LOE 3: Decode / Suggested Features

1. `main`: Update `main` to take into account multiple `.PCAP` files being passed via the command line.
2. `file_names`: Develop `file_names`.
3. `Decode IPv6`: Update `Decode` to support `IPv6`.
4. `Suggested Features: Decode IP Options / Ethernet 802.x`: Update `Decode` to support either IP Options or Ethernet 802.x headers—.
5. `Suggested Features: Decode 4in6 / Teredo / 6in4`: Update `Decode` to support either 4in6, Teredo, or 6in4—.