Name: SGT Schoenwald-Oberbeck, Jesse
Date: 20FEB2016
Current Module: Data Structures and Algorithms
Project Name: Zergmap

Project Goals:
Create graph from packets received, make sure the graph is properly and effectively dispersed, and that every node has at least two disjoint paths to every other node.

Considerations:
o Packets can be in one or multiple files.
o Units can be anywhere on the globe, and latitude/longitude lines change depending on distance from polls.
o No more than half the nodes should be removed from the available nodes.
o Nodes must all have 2 paths to other nodes without any collisions. So a pathing algorithm to find disjoint paths must be found or created.

Initial Design:
Main exists within decode.c, all other functions are in codec_functions.c, or mapperB.c, with some structures and prototypes being defined in structures.h.

Files will be read in one at a time. The file contents will be read in as packets, into corresponding structures which contain the data.

Data Flow:
Main will send the data to handler functions for each portion. Separating the file header, packet header, ethernet header, and so forth.

The zerg packet data, once encapsulation is removed, will be placed into Node structures.

These node structures are built to be handled as both a graph (utilizing adjacency lists) and a tree, while still being only one set of nodes.

Nodes will be built when the packet is read in, and placed into the graph and tree using separate functions. The graph function will be followed by a function to iterate over existing nodes and build the adjacency list, to be attached to each respective node, made of Edge structures, which point to each node to which the node in question is adjacent.

Paths and collisions are found via a custom algorithm which runs two path checks concurrently. Each path will check if the node it is on has a node in common with the other, and avoid that shared node while still moving toward the end-point. Where the shared node cannot be avoided, only one path should take it. If both paths are required to use the same node, a collision is found, and redundant paths are impossible.

Communication Protocol:

Ethernet, IPv4, IPv6, and UDP data are to be handled. Sockets are not to be handled.

Potential Pitfalls:

- o Multiple data structures have to be employed and navigated.
- o Data structures must be build from partially unknown data, and assessed on the fly.
- o Existing algorithms are either inappropriate for the goal, or difficult to implement.
- o "Edge cases" for possible graph configurations raise large concerns.
- o Side issues like handling IPv6, filtering UDP based on port, etc, will take time away from the main issues.

Test Plan:

User Test:

Execute the binary on provided test files, observing outcome and comparing to expected correct outcome.

Test Cases:

"Tails" are removed correctly from graphs. Straight line graphs are reduced, inappropriately down to one node. Collisions (points where two paths MUST collide) are found correctly, but not yet handled. UDP not filtered on port.

Conclusion:

The planning phase took too long, to put it simply. Conceiving a data structure to handle graphing, when graphing was a new concept, as well as building an algorithm in concept took enough days that I ran out of time to implement the ideas fully.

None of the existing algorithms I found were suitable to my project, so I took it upon myself to create one of my own. I should instead perhaps have modified an existing pathing algorithm to suit my needs, but I was convinced I had thought of something I could implement faster than I could comprehend any of the found algorithms sufficiently to tailor them to my needs.

Ultimately, my downfall with this project was taking too much of the allotted time to plan how to do things rather than find something that existed and worked. Partially because I feel I still have virtually zero experience or knowledge with algorithms and needed to invent something. Proper research requires somewhat of an existing knowledge base, and the class repeatedly going over Dijkstra's pathing algorithm did not prepare me for any other algorithm or variant.