Christopher Di Bella (chrisdb)

Data structure

The network topology is structured in a graph format that consists of three data structures: the node, the edge and the graph.

The node (struct)

The node simply represents a vertex in the graph. It is not used to store vertices in the graph, but rather to store visited nodes in search algorithms. The structure consists of a value attribute (a character), which represents the vertex, and a pointer to the parent node (i.e. the vertex that suggested visiting this vertex).

The edge (struct)

Unlike nodes, edges are used in the graph's underlying structure. They consist of two vertices (both characters), two constant integers (one for weight, one for the maximum number of connections), and a mutable integer for the current load.

The graph (class)

This data structure is the backbone of the entire simulation. This *graph* data structure directly represents the network and consists of:

- An associative array that maps characters (vertices) to a list of edge pointers. The list represents an adjacency list. Pointers to edges were necessary because the graph is bidirectional, and so a modification for A -> B implies a modification for B -> A. Since using values would require looking up A -> B and B -> A, and then modifying the appropriate data, pointers were used so that only one edge would ever need to be modified per two vertices.
- An associative array that maps end-of-connection times to an array of edge pointers. This is what stores the virtual connections. The array simply keeps track of which edges need to be updated when the connection is terminated.
- A random number engine, used for randomisation.
- The three search algorithms, each of which accept a source and destination (characters) and return an array of node pointers.
- A sanitisation function that cleans up the array from the search algorithms by removing anything that was visited but not a part of the final path.
- An establishment function, which creates the connection (if there is one), and
- An abolish function, which terminates the connection after the given time.

Tabulation

	Requests	Success	Success %	Blocked	Blocked %	Avg Hops	Avg Prop
SHP	5884	3380	57.44	2504	42.56	4.15	3.49
SDP	5884	4188	71.18	1696	28.82	6.01	6.37
LLP	5884	4759	80.88	1125	19.12	3.49	2.98

Discussion

Success/Blocks

It appears that SHP seems to block many more routes than the other two, which are comparable. This might be because there are fewer paths to choose from and so the randomisation was more biased toward blocked paths (e.g. 2 paths = 50% chance to select one, and it might be blocked).

Average hops

It is interesting that the SHP algorithm was not the most optimal when concerning the average number of hops. This may be because the SHP algorithm is an uninformed search and therefore does not select things (remotely) intelligently.

It is not surprising that SDP has the most number of hops, as it might be less costly to navigate through many nodes to get to the destination if the delay is much less.

Average propagation

This is the most surprising set of results. Not only did SDP *not* perform best, it actually performed the worst. No reason can be provided as to why.

Conclusion

Although further study is necessary, from the above data, it is apparent that LLP is a strong contender as a routing algorithm and is recommended for the time being.