Routing Algorithm Report

490210055

1 Network Topology

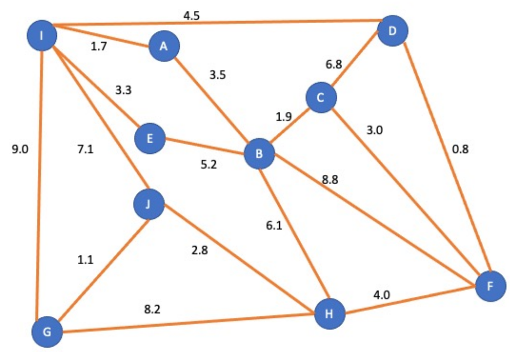

Figure 1: The topology built upon loading the default node configuration files

2 Routing Algorithm

Distributed, asynchorous Bellman-Ford algorithm from the lecture slide.
$D_x(y) = min_v(c(x, v) + D_v(y)), \forall y \in$ network

3 Implementation

The *main* program calls *manager* to create threads for each specific task. The threads use some shared memory like *neighbor_status* and *routing_table* to handle data concurrency.
A *send* thread of a node to one of its neighbors encodes the node's *routing_table* in such format:
node name >>> [**destination node** : **least cost distance** , **next hop port number**;]>>> [**activated nodes**]
A *receive* thread for the neighbor decodes the incoming messages by the delimiters above.

The *receive* and *send* threads can be regarded as two ends of a pipe which is labelled by a *port* number since they share the same ip address. After the main program has collapsed for 60s, the *receive* threads will chain new routing calculation threads(*dv_routing*)

4 Limitations

- Concrete least cost path can't be generated from only one routing table.
  While it is possible to output the whole path by keeping a record of other nodes' routing table, I decide not to do so, since the key feature of a distance vector algorithm is to track local information at each node, in this case, information about one node's neighbors. If global information is tracked, it is better to use a link-cost algorithm. However, a whole path could be think of jumping from one node to it's next hop towards its destination. For example, a path should be A-B-C-F-D. Starting from A, we could see an entry in the routing table of A, which, as an instance, is D: 3.5, B. We know that the next hop node is B, thus, we refer to the routing table of B of which one destination is D. Repeat such a chain in routing tables, a complete path could be achieved.

- The implementation failure to reset and converge to another state if any node is disconnected/ have increased costs.
  My implementation only has a mechanism to check new connections/smaller costs changes due to the implementation of *update_node_status* function. If the codes could be reconstructed, a more suitable logic pattern could be similar to the sample pseudo code from the textbook, which, at each node, x, there is an initialization and an update.
  During initialization, $D_x(y) = c(x, y)$ if y is a neighbor. ∞ otherwise. Send the D matrix to each neighbor.
  During the update loop, a node waits for a link cost change(where my implementation doesn't fully detect) or a distance vector from a neighbor(in my implementation, vectors are received every 10 s, because of the following step). Then use the Bellman-Ford algorithm to find the least cost. If the distance vector for the node changes, broadcast to its neighbors.(but my implementation keeps broadcasting every 10s no matter there's a change or not).

- Missing implementation of command line modifying edge length(cost between two neighboring nodes)
  This is a direct side effect of the item just mentioned above.