COMP3221 Assignment №1

Tianyi Chen, 490165227

One of the topology used is the same as the example, and the other two are generated randomly by **topologyGenerator.py**. This program first randomly creates a net typology with 10 nodes and 15 edges with weight which you can modify. Then it outputs configuration files of each nodes in the same directory. In addition, it gives you the shortest path between each nodes by Dijkstra and visualise the graph for you.

The routing algorithm used in this program is based on Bellman-ford algorithm, storing distance vectors and the next hop along the shortest path of each nodes in the routing table as well as sending routing tables between neighbours to update and communicate. For each node x, it stores following variables and the last four items are included in the update packet as the routing table:

- For each neighbour v, the cost $c(x,v)$ and port number from x to a directly attached neighbour node v
- Node x's distance vector, i.e. the least cost from x to all other nodes in the net topology
- Node x's next hops, i.e. the next node on the least-cost path from x to another node v in the net topology for all v in the net topology
- The distance vector of each of other nodes in the net topology
- The next hops of each of other nodes in the net topology

There are four threads for each nodes, one for taking user's input from the interface and handling the corresponding responses; one for sending the update packet to its neighbours every 10 seconds; one for receiving the update packet from neighbours and detect if there should be an update for the current node's routing table, if so, then updates the least cost and the next hop; one for calculating the least-cost path and output the results.

The least cost is calculated based on the formula: $D_x(y) = min_v\{c(x,v) + D_v(y)\}$ for all nodes y and all neighbours v when detecting an update. The least-cost path is calculated by iterating on the next hop of each nodes for a shortest path to the target node, since for each node x, it stores the next hop of all other nodes on a shortest path to all nodes. For example, for a target node y, if $nextHop(x,y) = v$, then the algorithm puts v in the path and finds $nextHop(v,y)$ until reaching $nextHop(u,y) = u$. Then the path $x, v, ..., u, y$ is the shortest path from x to y.

What makes my system special is how it correctly stores the routing tables of other nodes in the current node and how it handles link cost changes and node failures. There are 3 different packets corresponding to different situations and 2 types of sequence numbers to make sure that the algorithm is correct.

- When there is no link-cost change and no node failures, the update packet from nodes x to v contains the distance vectors of all nodes (included in the routing table) currently stored in x, and each distance vector has a sequence number to indicate the time of the last updates of that node's distance vector. Upon receiving, node v only updates it's own distance vector and next hop based on the routing table of v, link cost of (v, x) and the distance vector of x in the update packet. If there is an update then the sequence number of distance vector of v increases by 1. Node v also updates the distance vectors of other nodes i stored in v by directly replacing the old one by the one in update packet, if the sequence number of the node i's distance vector in node v is less than the sequence number of node i's distance vector in node x.

- When there is a link-cost change, the new link-cost is included in the update packet and sent to the neighbour nodes. Both nodes will update the config file and do the same operating mentioned above. Then both nodes will broadcast whether there is an change in their distance vector.

- When there is a node failure of an arbitrary node x, the neighbour nodes of x will firstly remove all edge towards x as well as the distance vector of x, then format the routing table, restart the algorithm as if node x doesn't exist in the net topology. All neighbour nodes increase the node sequence number by 1 (N.B. this is different from the distance vector sequence number) to indicate that there is 1 more node failure. This new node sequence number will be included in the packet (before the first node failure, all node sequence numbers are 0). The failed node is also included in the packet for them to broadcast. When every other nodes receive this packet, they as well format routing table, remove failed node and restart. Since all nodes have a new node sequence numberthey only update their own distance vector if the Node sequence number matches each other, to make sure that they have incorporated the same information. For each node, it only restarts the algorithm when they firstly receive the packet of node failure of a particular node and ignore following packets of the same type. Each node also sends node failure packet once after they restarted. Node that no matter what packet types they are, they always contain the routing tables for neighbour nodes to update.

Node that the first output of a node's shortest path will be displayed $60s$ after the start-up, hence, user needs to wait for the program to output. In addition, since "bad news travel slows" in a distributed system, it takes time for the network to re-converge after link-cost changes or a node failures. Although the program outputs the shortest path of each node every time it detects a least cost change, user is expected to wait for the network to re-converge itself. The network is guaranteed be converged when the routing tables of each nodes in a connected graph are the same: distance vectors, next hops, sequence number of each distance vectors and sequence number of the node.