**Programming Assignment 3 : Distance Vector**

**Multi-threaded routing application with triggered and periodic updates:** The Application implements a multi-threaded distributed asynchronous distance vector routing algorithm with support for both triggered and periodic updates.

Following are the primary characteristics:

1) **Multi-threaded :** 2 threads are implemented to achieve asynchronous system. Thread 1 is dedicated for periodic updates and Thread 2 receives Advertisement from other sockets, process them and send triggered updates. Mutual Exclusion is applied by using a mutex variable “rtmutex” on Routing table/Graph entries. Every time an Advertisement is received mutex lock is enabled until Bellman ford algorithm is run and routing table is updated with the final values. Also, mutex is applied when an Advertisement has to be sent until it has been sent over the socket. This enables 2 things:

a) Nobody can write when it is being read

b) Nobody can read when it is being written.

2) **Graph and Routing Table data-structures:** The appropriate choice for a data structure for the graph used for the purposes of distance vector algorithms used in the application was a 2-dimensional array. The rows represent the routing tables of the respective nodes as known by the current node maintaining the graph. The columns are the destinations with the cell values the costs to the destinations.

The routing table data structure is a vector (as the number of nodes read from the configuration file could be variable) of components with the following attributes:

in\_addr destination **/\* address of the destination \*/**

in\_addr nexthop **/\* address of the next hop \*/**

int Cost **/\* distance metric \*/**

long TTL **/\* time to live in seconds \*/**

**typedef std::vector<RouteEntry> RouteEntryVector;/\* Declaration of the RouteEntry vector \*/**

The distance-vector algorithm uses the graph to find the smallest paths to the destinations represented by the columns and updates the same in the RouteEntryVector, where the next hop and ttl information per destination is maintained.The rows in the graph also give a count to the number of vertices (nodes) distributed in the network.

3) **Main –** The MainClass uses the following arguments as indicated in the problem statement and can be run as follows:

**./distanceVector <config filename> <portnumber: 65531> <ttl> <infinity value> <update period for advertisments> <split-horizon boolean: true/false>**

Also , the program can be run by using the script ./run1.sh for node A, run2.sh for node B and run3.sh for node C for a three node topology.

**4) Initialize :** The **void** **initialize**(string fileName) and **void** **initiaizeGraph**() functions initialize the RouteEntryVector and the graph respectively. The RoutingTableVector is initialized to values read from the **config** file and the graph values are initialized to INFINITY\_VALUE as input during the execution of the program.

5) **Send\_Advertisement() :** The advertisement is serialized by the node and sent every **period** seconds and is composed of a stream of IP addresses and cost values for each other node in the topology.

6) **Update :** The update function in the application performs the following tasks as specified in the problem statement:

* Decrements the TTL in the routing table entries by the specified amount of time as input initially.
* Checks for advertisement/update messages from neighbors. Process received messages by updating edge weights in your graph. Run Bellman-Ford to recalculate your routing table. If any routing table entries change, set the TTL for that entry to the default value.

**Expired TTL :** If the TTL for an entry has expired, this means that an advertisement hasn’t been received from a neighbor for multiple time windows. When a TTL expires, the node is considered unreachable, and the **Cost** for reaching the node is set to INFINITY\_VALUE followed by :

* Printing of the updated routing table.
* An update message is sent to the node's neighbors.

7) **Split Horizon :** This has been implemented taking into consideration routing loops such that if A can reach C via B it should not update this route to B as B is a part of it , in the topology A---> B ---> C

8) **Detecting Node Failures :** If there is no periodic update from the node for 'period' seconds , the node is considered to have gone down and the cost of reaching it from neighboring nodes is updated to INFINITY\_VALUE on its neighboring nodes.

**Analysis:**

**Time taken to :**

**a) Establish routes to all nodes during initialization**

**b ) Converge to a steady state after a node goes down. Your analysis should vary the**

**Infinity and describe the effect that this variable has on the time it takes to**

**converge**

**This is discussed with the help of following test cases -**

**Period: 30s**

**Default TTL: 100s**

**Infinity: 16**

**The Topology :**

A: 129.79.242.101

B: 129.79.242.102

C: 129.79.242.103

D: 129.79.242.104

**Node A** **Node B**

**Node D** **Node C**

**Case 1: 4 nodes without split horizon: (Simulating Node A Failure)**

**Log file entries from -**

**Node A :**

12-04-2015 20:34:49.331850[DEBUG] 129.79.242.101 **stablilized in 4 sec**

**Node B :**

12-04-2015 20:34:49.333130[DEBUG] 129.79.242.102 **stablilized in 0 sec (less than 1 sec)**

12-04-2015 21:38:29.454221[DEBUG] Routing table before sending advertisement

Destination:129.79.242.102 nextHop:129.79.242.102 cost:0 ttl:1449282927

Destination:129.79.242.101 nextHop:129.79.242.103 cost:3 ttl:1449283209

Destination:129.79.242.103 nextHop:129.79.242.103 cost:1 ttl:1449283209

Destination:129.79.242.104 nextHop:129.79.242.103 cost:2 ttl:1449283147

12-04-2015 21:38:29.500453[DEBUG] Routing table before sending advertisement

Destination:129.79.242.102 nextHop:129.79.242.102 cost:0 ttl:1449282927

Destination:129.79.242.101 nextHop:129.79.242.103 cost:16 ttl:1449283209

Destination:129.79.242.103 nextHop:129.79.242.103 cost:1 ttl:1449283209

Destination:129.79.242.104 nextHop:129.79.242.103 cost:2 ttl:1449283147

12-04-2015 21:38:29.454077[DEBUG] 129.79.242.101 **value changed time since from stable is 62**

12-04-2015 21:38:29.454090[DEBUG] 129.79.242.101 **value changed time since ttl is 125**

**Node C :**

12-04-2015 20:34:54.641904[DEBUG] 129.79.242.103 **stablilized in 2 sec**

12-04-2015 21:38:29.463568[DEBUG] Routing table before sending advertisement

Destination:129.79.242.103 nextHop:129.79.242.103 cost:0 ttl:1449282929

Destination:129.79.242.101 nextHop:129.79.242.102 cost:4 ttl:1449283209

Destination:129.79.242.102 nextHop:129.79.242.102 cost:1 ttl:1449283209

Destination:129.79.242.104 nextHop:129.79.242.104 cost:1 ttl:1449283209

12-04-2015 21:38:29.498343[DEBUG] Routing table before sending advertisement

Destination:129.79.242.103 nextHop:129.79.242.103 cost:0 ttl:1449282929

Destination:129.79.242.101 nextHop:129.79.242.104 cost:16 ttl:1449283209

Destination:129.79.242.102 nextHop:129.79.242.102 cost:1 ttl:1449283209

Destination:129.79.242.104 nextHop:129.79.242.104 cost:1 ttl:1449283209

12-04-2015 21:38:29.463433[DEBUG] 129.79.242.101 **value changed time since from stable is 62**

12-04-2015 21:38:29.463445[DEBUG] 129.79.242.101 **value changed time since ttl is 62**

**Node D :**

12-04-2015 20:34:54.649654[DEBUG] 129.79.242.104 **stablilized in 0 sec (less than 1 sec)**

12-04-2015 21:38:29.461533[DEBUG] Routing table before sending advertisement

Destination:129.79.242.104 nextHop:129.79.242.104 cost:0 ttl:1449282932

Destination:129.79.242.101 nextHop:129.79.242.103 cost:3 ttl:1449283209

Destination:129.79.242.102 nextHop:129.79.242.103 cost:2 ttl:1449283147

Destination:129.79.242.103 nextHop:129.79.242.103 cost:1 ttl:1449283209

12-04-2015 21:38:29.502143[DEBUG] Routing table before sending advertisement

Destination:129.79.242.104 nextHop:129.79.242.104 cost:0 ttl:1449282932

Destination:129.79.242.101 nextHop:129.79.242.103 cost:16 ttl:1449283209

Destination:129.79.242.102 nextHop:129.79.242.103 cost:2 ttl:1449283147

Destination:129.79.242.103 nextHop:129.79.242.103 cost:1 ttl:1449283209

12-04-2015 21:38:29.461391[DEBUG] 129.79.242.101 **value changed time since from stable is 62**

12-04-2015 21:38:29.461404[DEBUG] 129.79.242.101 **value changed time since ttl is 125**

**Looking at the above test case logs we can say that after the node is down, approx. time taken to converge to infinity is around 125s which is closer to the Default TTL of 100s.**

**Case 2: 4 nodes with split horizon: (Simulating Node A failure)**

**Node A :**

12-04-2015 20:09:02.898568[DEBUG] 129.79.242.101 **stablilized in 4 sec**

**Node B :**

12-04-2015 20:09:02.900116[DEBUG] 129.79.242.102 **stablilized in 0 sec (less than 1 sec)**

Destination:129.79.242.102 nextHop:129.79.242.102 cost:0 ttl:1449277742

Destination:129.79.242.101 nextHop:129.79.242.103 cost:3 ttl:1449278389

Destination:129.79.242.103 nextHop:129.79.242.103 cost:1 ttl:1449278479

Destination:129.79.242.104 nextHop:129.79.242.103 cost:2 ttl:1449278449

**Node C :**

12-04-2015 20:09:09.44204[DEBUG] 129.79.242.103 **stablilized in 3 sec**

Destination:129.79.242.103 nextHop:129.79.242.103 cost:0 ttl:1449277746

Destination:129.79.242.101 nextHop:129.79.242.102 cost:2 ttl:1449278449

Destination:129.79.242.102 nextHop:129.79.242.102 cost:1 ttl:1449278539

Destination:129.79.242.104 nextHop:129.79.242.104 cost:1 ttl:1449278510

**Node D :**

12-04-2015 20:09:09.48324[DEBUG] 129.79.242.104 **stablilized in 0 sec (less than 1 sec)**

Destination:129.79.242.104 nextHop:129.79.242.104 cost:0 ttl:1449277749

Destination:129.79.242.101 nextHop:129.79.242.103 cost:3 ttl:1449278569

Destination:129.79.242.102 nextHop:129.79.242.103 cost:2 ttl:1449278569

Destination:129.79.242.103 nextHop:129.79.242.103 cost:1 ttl:1449278569