**Programming Assignment 3 : Distance Vector**

**Multi-threaded routing application and supporting triggered updates:** The Application implements a multi-threaded distributed asynchronous distance vector routing algorithm. Following are the primary characteristics:

1) **Multi-threaded**- <shared variables, mutex and conditional variables>

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 \*/**

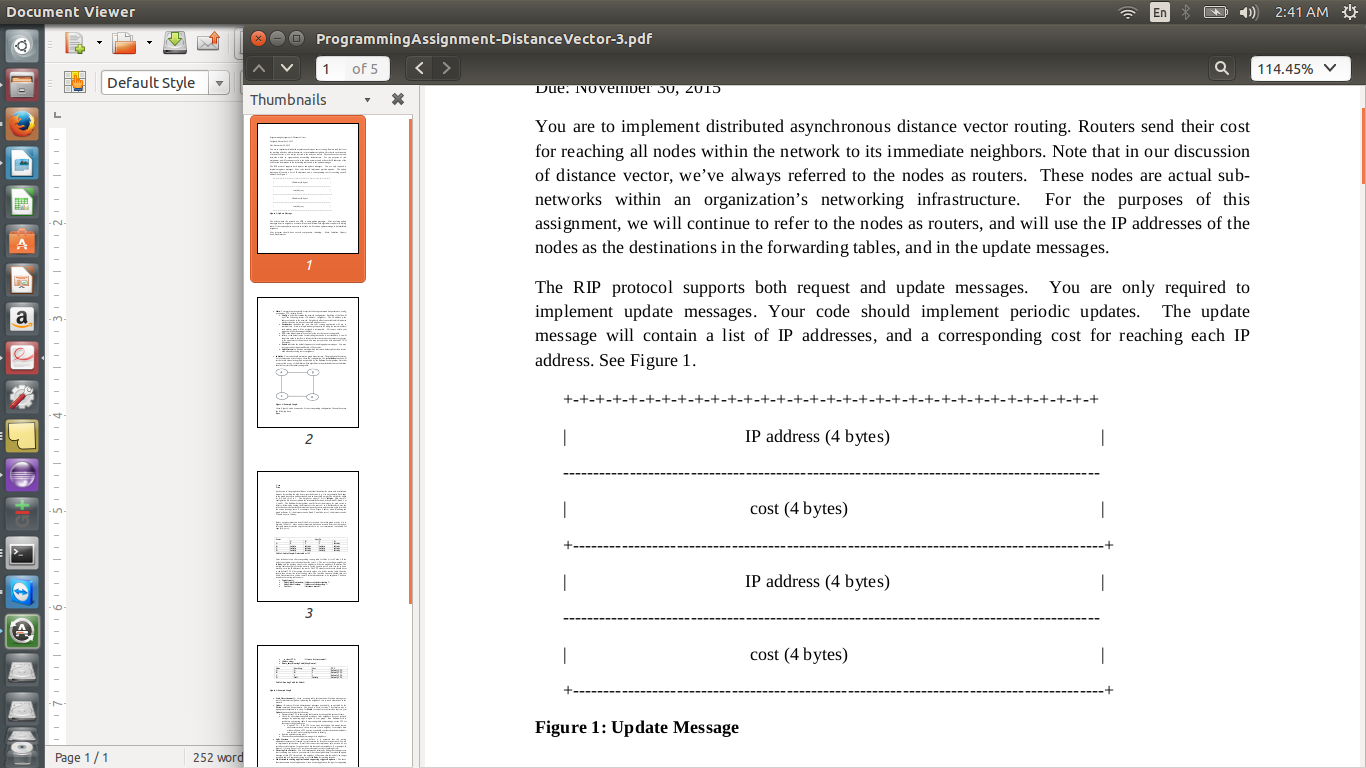
The distance-vector algorithm uses the graph to find the smallest paths to the destinations represented by the columns and updates the same in the RouteEntryVector, where the next hop and ttl information per destination is maintained.The rows in the graph also give a count to the number of vertices (nodes)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

**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 the following, as indicated in the problem statement : <need to enlarge and crop image>



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**

**8) Detecting Node Failures**

**Analysis:**

**1) 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.**

**2) Split Horizon and convergence -**