CSE 589 - Modern Networking Concepts
Project II

IMPLEMENTATION OF DISTANCE VECTOR PROTOCOL

Subhranil Banerjee
UB # : 50096800
Email: subhrani@buffalo.edu

Implementation Details

1. Data structures involved:

- **Node:** Holds the details of a server on the network.
  - id : The serverid for this node as mentioned in the topology file.
  - IP : The IP address of the node.
  - port : Port on which the node receives incoming data.
  - cost : Cost of the link (or shortest path) to the node from the current host.
  - isActiveNeighbour: To determine if it is an active neighbour* of the host.
  - updateCount : To keep track of the **messages received from this node.

  * A neighbour whose distance vector updates the current host will process and to whom it will send it's own distance vector.

  ** When this count reaches 3, the link cost with that node is set to infinity.

  This data-structure has been initialized on **line 39.**

- **Node costList[] :** An array of Node objects. Although it contains entries for all the nodes in the network , the cost parameter is set only for the neighbouring nodes and is set to -1(infinity) for the others.

  This data-structure has been initialized on **line 64.**

- **Node distanceVector[] :** An array of Node objects. It is the implementation of the routing table. Contains the shortest path to a specific node from the current node.

  This data-structure has been initialized on **line 65.**

- **costMatrix[][]**: A two-dimensional array for storing the minimum costs of traversal between any two nodes. costMatrix[i][j] refers to the minimum cost of traversal between the ith node and the jth node. The zeroth row contains the same costs as the distance vector of the current node.

  This data-structure has been initialized on **line 66.**

- **mssg[]:** An array of bytes(char) inside the function *createMessageFormatFromVector()*. The function iterates over the entries in the array distanceVector[] and stores the details into this array, using memcpy(). This array is sent over the socket by the system call *sendto()*.

  This data-structure has been initialized on **line 150.**

- **tempVector[]:** An array of Node objects inside function *convertMessageIntoDistanceVector().* It stores the distance vector received on the socket.

  This data-structure has been initialized on **line 188.**

2. Explanation of Source Code:

The program takes 2 parameters:

- ➔ Path of topology file.
- ➔ Timer interval for periodic updates.

The major modules of this project have been explained below.

- **Parsing Topology File:** The topology file passed as input to the process is opened and scanned line-by-line inside function *initialiseLists().* The initial set of values (details of nodes) are used to fill the entries into the costList[] and distanceVector[] arrays. The next set of values are used to fill the cost parameter in the costList[] array.

Note: Index 0 for both the arrays contains details of the current host reading the file.

- **Receiver :** Implemented inside function *setupReceiver()*. We create a socket and bind it to the IP address and port for the current host , read from the topology file. We then run the system call select() inside a while loop with the following parameters:
    - Socket id of the main socket.
    - STDIN (0) for accepting user commands.
    - Timeout value specified as command line arguments.

- **Generation of Message:** Implemented inside function *createMessageFormatFromVector()*. The function iterates over the entries in the array distanceVector[] and stores the details into the structure **mssg[]**, using memcpy().

- **Parsing received message:** Implemented inside function *convertMessageIntoDistanceVector()*. The function creates an array **vector[]** with node objects to hold the distance vector received on the socket. Since the message has a definite format , we use memcpy to store the details of the message into vector[]. We also update costMatrix[][] accordingly.

- **Broadcast** : Implemented inside function *broadcastDistanceVectorToNeighbours()*. It calls *createMessageFormatFromVector()* to generate the message. It then iterates over all neighbours, checks for nodes which have three failed updates, updates the cost accordingly and then sends them the message over an UDP socket.

- **Timer :** Implemented using the timeout parameter in select(). When select() returns with 0, we perform the timer handler i.e. *broadcastDistanceVectorToNeighbours()*. The timer gets reset everytime we execute the “step” command successfully.

- **Updating the distance vector:** This is implemented using two functions:
    - *resetDistanceVector():* We iterate over all items in distanceVector[] and costMatrix[0] and set them to -1 (infinity). This function is called everytume we want to forcefully update our distance vector e.g. cost changes of a link (using update command) or disabling link.

    - *updateSelfDistanceVectorWithVector():* The *isinit* parameter helps distinguish between the initialization case i.e. copying cost values from costList[] to distanceVector[] and the regular update case i.e. applying the Bellman Ford algorithm for determining shortest link path. For the latter case, we run a pair of nested loops . the outer loop iterates over all nodes of the distance vector and the inner loop runs over all neighbours. We use the bellman ford equation to determine the minimum cost for a particular destination node and update distanceVector[] and costMatrix[][] accordingly.