Computer Networks: Project Report – Assignment 3

12/05/2015

Thanmai Mallesha Bindi and Abhinandan Sampathkumar

# Structure of Messages

The message sent and received by routers contain IP address and cost in a sequential order. Below is the description of how an advertisement looks.

|  |
| --- |
| IP ADDRESS(4 BYTES) |
| COST(4 BYTES) |
| IP ADDRESS(4 BYTES) |
| COST(4 BYTES) |
| ………………………………. |
| ………………………………. |

# Data Structures

## Node list

**vector<Node\*> node\_list;**

Node list is a vector of Nodes. The Node class is shown below, we parse the configuration file and store the initial values in the list of nodes.

**Class Node {**

**int index;**

**bool neighbor;**

**struct sockaddr\_in node\_address;**

**String address;**

**bool available;**

**}**

## Graph

**vector<vector<int>> graph;**

We used a vector of vectors to represent the graph table. This holds the value of cost from source node to destination nodes. Below is a depiction of the graph table.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **A** | **B** | **C** |
| **A** | 0 | 1 | 1 |
| **B** | Infinity | Infinity | Infinity |
| **C** | Infinity | Infinity | Infinity |

## Routing Table

**vector<RouteEntry\*> routeTable;**

Routing table is a vector of RouteEntry. The RouteEntry class has address of the destination, the next hop to reach a node, cost to the destination and the time to live.

**Class RouteEntry {**

**String destination;**

**String next\_hop;**

**int cost;**

**long time\_to\_live;**

**}**

|  |  |  |  |
| --- | --- | --- | --- |
| **Destination** | **Next\_hop** | **Cost** | **Time\_to\_live** |
| **A** | **A** | **0** | **TTL** |
| **B** | **B** | **1** | **TTL** |
| **C** | **B** | **2** | **TTL** |

## Distance vector

**Vector<int> distance\_vector;**

We reinitialize the Distance vector to infinity, every time we get an update. This vector of integers is used to update the cost to nodes after running the bellman ford algorithm and use this cost to update the routing table whenever there is an update.

|  |  |  |  |
| --- | --- | --- | --- |
| Advertise / Dest | A | B | C |
| B | **1** | **0** | **1** |

# Multi-threading

Two Threads are used to handle the routing updates and receive. Updates include the triggered and the periodic Update

## Main thread

The main thread of the program takes input parameters for the program from console, reads config file, creates a **pthread** and handles the periodic updates. The periodic update occurs once every Period\_time seconds, where Period\_time is specified by the user

## pthread

This thread handles all the received updates from other routers and it sends a triggered update for every change in the routing table

## Mutex

**pthread\_mutex\_t mutex1;**

To make sure read and write operation does not occur together on resources shared by threads, we use mutex lock and mutex unlock on code blocks as shown below.

**pthread\_mutex\_lock (&mutex1 );**

**{**

**/\*Code block\*/**

**}**

**pthread\_mutex\_unlock(&mutex1 );**

# Class files/header files

## Socket Class

Low level Class that handles socket level functionalities. This contains the socket file descriptor and port number that is being communicated with.

## Node Class

Stores information about Router/Node such as: Neighbor, Index in the Graph, Address in String & SockAddr\_in format. Config will be parsed and Node information about the network will be stored in this class.

## Advertise Class

Does not have a member function that is used. An interface which handles Sending & Receiving of Advertisement. Prepares Message to be sent to a node. Also does Parsing of Messages received from a node.

## RouteEntry Class

Stores information about Destination Router and next hop to reach it as well as the cost of reaching that destination. Also, has Time to Live, number of seconds that a Node is considered to be alive.

# Graphs For analysis

# Analysis

Test cases:

1. 3 – Node linear
2. 4 – Node Square
3. 5 – Node Square with diagonal Node

Test case 1:

Command: /Main config 65520 30 16 10 1

Description: Taking out A & C

Time-to-Live: 30 seconds

Infinity: 16

Period: 10 seconds

* With Split Horizon
  + Initial Time: 0 for B, 11 for A and C
  + Convergence Time: 30 for C when A was killed but 10 since the last stable update recvd from B.
* Without Split Horizon
  + Initial Time: 0 for B, 16 for C, 14 for B
  + Convergence Time: 30 for A when C was killed but 6 for A & 10 for B since the last stable update recvd.

Test case 2:

Command: /Main config 65520 90 16 10 1

Description: Taking out D & C

Time-to-Live: 90 seconds

Infinity: 16

Period: 10 seconds

* Without Split Horizon
  + Initial Time: 4 for D, 1 for A, 3 for B & 11 for C
  + Convergence Time: 90 for A, B & C when D was killed but 10 for A, B & C since the last stable update recvd.
* With Split Horizon
  + Initial Time: 0 for A, 1 for B, 3 for C & 4 for D
  + Convergence Time: 90 for A, B & D when C was killed but 7 for A, B & D since the last stable update

Test case 2:

Command: /Main config 65520 90 999 10 1

Description: Taking out E

* Without Split Horizon
  + Initial Time: 3 for A, 5 for B, 6 for C, 6 for D & 13 for E
  + Convergence Time: 90 for A, B, C & D when E was killed but 12 for A, 11 for B, 10 for C and 0 for D since the last stable update
* With Split Horizon
  + Initial Time: 5 for A, 5 for B, 8 for C, 6 for D & 15 for E
  + Convergence Time: 90 for A, B, C & D when E was killed but lesser time than without split horizon

This concludes that it takes very less time to converge in the case of Split Horizon but it will take a bit more time in the case of regular network. Time taken to initialize and stabilize the network will depend on all the nodes in the network. Initial time is different for each node in the network. Please check logs to know more about time. As we increased the Infinity value there was a slight increase in the convergence time but not significant enough.

Triggered Update: