Assignment 3

Write up

12/1/2015

Mangirish Waghle and Siddharth Jain

Contents

Graph Data Structure

We use a standard 2-D integer array for representing the graph vertices and edges. To identify which row or column represents which node, a separate data structure is used. It is an array of following structure:

```
typedef struct {
        //identifies if this node is a neighbour or not
        unsigned char isNeighbour;
        // Flag to check if any msg has been received from the node.
        int msg_recvd;
        //ip of the node
        char *ip;
} GraphNode;
```

For identifying which row/column index correspond to which node in graph, we lookup for the node at that index in the array of GraphNode.

Routing Table Data Structure

We use the following structure for storing routing table.

```
typedef struct {
        //desitination of the route
        char *destination;
        //next hop of the route
        char *next_hop;
        //cost for this route
        int cost;
        //ttl for this route
        int ttl;
        //for split horizon-no more used
        char *learnt_from;
} RouteEntry;
```

Again we use the GraphNode array to map the rows and columns with the node.

Multi-threaded Implementation

Threads

Three threads are being used in our application:

1. Receiver Thread: It just receives a packet from the UDP socket and enqueues it in the common data structure
2. Main/Processing Thread: Initially it initializes the application and once it is done with it, it acts as a thread to purely process the messages received.
3. Periodic Update Thread: This thread sends the periodic advertisement to its neighbours and decrements the TTL.

Common Data Structure

A queue of following structure is used for storing the messages received:

```
typedef struct QueueNode {

        //message received

        char *msg;

        //ip from which the message was received

        char *from;

        //next node

        struct QueueNode *next;

} QueueNode;
```

Mutex

Two mutex are used in our implementation:

1. For limiting access to the shared queue
   The mutex lock is used while enqueuing from the receiving thread and while dequeing from the message processing thread, so as to keep the data structure consistent
2. For limiting acess to the routing table
   The mutex lock is used while making changes in the routing table, i.e. during executing distance vector/bellman ford algo. As well as when the advertisements are being sent to the neighbour. This is to avoid sending inconsistent data to the neighbours

Counting Semaphore

A counting semaphore is used to avoid busy waiting in the message processing thread. For every message received, signal() is invoked on the semaphore. Before processing any message in the message processing thread, wait() is invoked on the semaphore. This ensures the minimal wastage of CPU cycles.

Analysis

Graph 1 and 2 have comparison between split-horizon & non split-horizon cases wherein default infinity is 16

Graph 3, 4 and 5 have comparisons between split-horizon & non split-horizon cases wherein default infinity is 16 and 50

Following is the node legends used in the graph topologies below:

1- lh008linux-01.soic.indiana.edu

2- lh008linux-02.soic.indiana.edu

3- lh008linux-03.soic.indiana.edu

4- lh008linux-04.soic.indiana.edu

5- lh008linux-05.soic.indiana.edu

D- degu.soic.indiana.edu

Graph 1

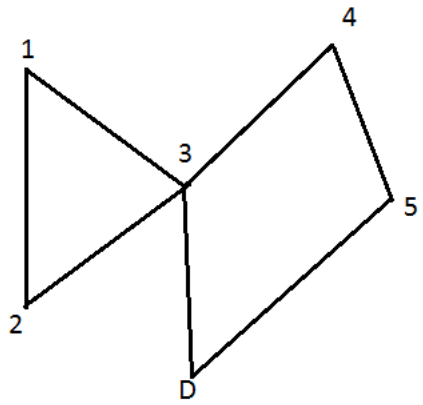
Data & Observation

Topology: D-1-2-3-4-5

Default TTL	90	90
Update Period	30	30
Default Infinity	16	16
Split Horizon	0	1
Initial Convergence Time (s)	33	33
Convergence Time After Failure (s)	239	38
Node killed	D	D
Observation	• Convergence Time After Failure (s) is substantially higher • Initial Convergence time is the same	• Convergence Time After Failure (s) is substantially lower • Initial Convergence time is the same

Graph 2

Topology:

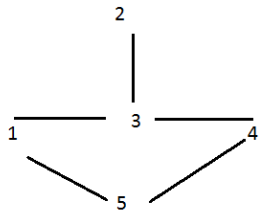


Data and Observation

Default TTL	90	90
Update Period	30	30
Default Infinity	16	16
Split Horizon	0	1
Initial Convergence Time (s)	30.4	31.8
Convergence Time After Failure	215	92
Node killed	3	3
Observation	• Convergence Time After Failure (s) is substantially higher • Initial Convergence time is about the same	• Convergence Time After Failure (s) is substantially lower • Initial Convergence time is about the same

Graph 3

Topology:



Data

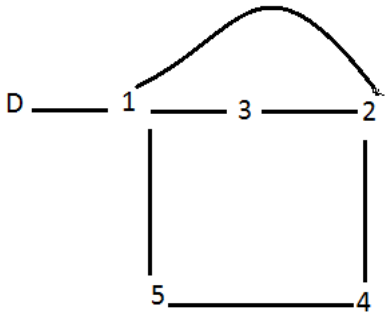
Default TTL	90	90	90	90
Update Period	30	30	30	30
Default Infinity	16	50	16	50
Split Horizon	0	0	1	1
Initial Convergence Time (s)	33.2	32	32.6	32
Convergence Time After Failure	274	787	85	86
Node killed	3	3	3	3

Observation

- Initial convergence time for all the cases are almost the same
- Convergence time after node failure is as expected lower in cases with split-horizon turned on
- Convergence time after node failure in case of non-split-horizon cases:
    - Convergence time after node failure is higher in case where default infinity value is higher, i.e. 50. This is due to the count to infinity behaviour
- The default infinity value does not affect the convergence time in split horizon cases, as count to infinity does not occur.

Graph 4

Topology



Data

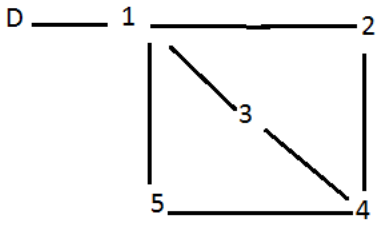
Default TTL	90	90	90	90
Update Period	30	30	30	30
Default Infinity	16	50	16	50
Split Horizon	0	0	1	1
Initial Convergence Time (s)	33	31.9	32.7	32
Convergence Time After Failure	72	78	Does not converge in reasonable time	Does not converge in reasonable time
Node killed	5	5	5	5

Observation

- Initial convergence time is almost the same for all the cases, as split horizon and default infinity do not factor in it
- Convergence time after failure in case of non-split-horizon cases was almost the same with different values of default infinity
- We could not observe convergence after failure in case of split-horizon cases, it gets stuck. We think the reason behind it is the cycle formed between 1, 3 and 2. When one of the nodes tries to detect the node failure with expiry of TTL, it does not succeed, because as soon as it does that the other node advertises it a lower value.

Graph 5

Topology:



Data

Default TTL	90	90	90	90
Update Period	30	30	30	30
Default Infinity	16	50	16	50
Split Horizon	0	0	1	1
Initial Convergence Time (s)	33	32.7	33	33
Convergence Time After Failure	353	1109	660	2686
Node killed	3	3	3	3

Observation

- Initial convergence time remains almost the same across all the cases
- Convergence time after failure in cases with split horizon is higher, due to count to infinity exhibited because of the presence of a loop. Reference: count to infinity is possible with split horizon as well,https://courses.engr.illinois.edu/cs438/sp2010/slides/lec08_routing.pdf
- Convergence time after failure with lower default infinity is lower in the respective category of comparisons of split-horizon and non-split-horizon cases