Comp3331 Assignment 2: Report

Matthew Moss
mdm@cse.unsw.edu.au
z3376173

June 3, 2014

Basic Design

The design of my program is fairly straightforward. A launcher (dv_router_ext.java) is responsible for parsing command line flags and configuration files, and instantiating a Router instance. Each Router is passed its direct neighbours on creation, and I assume that these do not change during operation of the program.

When a Router is created it schedules an event that sends the appropriate message to its direct neighbours at 5 second intervals. It also opens a socket on the port it was given at construction to listen for distance vector messages from its peers, and starts a thread listening for these messages. When a message is received, the thread locks the data structures in the router and inserts the new information. There may be race conditions when a packet arrives while the previous one is still processing, but the failure resistance nature of this implementation means this is not a concern.

Messages

The messages sent are produced by outputting Java objects in to a byte array, and reassembled using the reverse of this method at the other end of the connection. This is simple, and allows complex message contents with a minimum of cognitive overhead. All messages are sent over UDP, as is appropriate for routing information.

Distance Vector instances are sent directly as messages. These contain the id of the sending node, and a map of Characters to Distances of each node reachable by the sender. In addition, the Distance Vector class contains a map of Characters to Strings which are used to hold the path taken by the route in the extended version of the assignment. This field is null under basic and poisoned reverse conditions.

To construct the distance vector for sending, the router compares each of the most recent distance vectors received from its neighbours, and finds the minimal path to each known destination. Under special circumstances (poisoned reverse or path routing) there are additional restrictions on these messages.

For poisoned reverse, the messages propagate values of Float.FLOAT_MAX to neighbours that are the source of a current path to a location. in the extended version, Routers will not propagate routes to neighbours that are already a component of that route, as determined by the accompanying path.

Stabilisation

Determining when the network has stabilised is one of the more challenging components of this assignment. I finally settled on comparing messages received to the last three messages from a neighbour. If all these messages are the same, I consider the neighbour to be stable. Once all neighbours are stable, I consider the network as a whole stable. In practice this seemed to be a satisfactory approach.

Known Errata

- When running in either extended or poisoned routing mode, the network may not stabilise as expected in the spec. This has been addressed on Piazza (https://piazza.com/class/hs6ncxu2n1c1kg?cid=70) but is as yet unresolved.
- Under basic conditions, some node failures may result in the count-to-infinity problem. This is a known issue (https://piazza.com/class/hs6ncxu2n1c1kg?cid=67) and will should not be considered a violation of the spec.

Extension

Below is a topology that still suffers from the count-to-infinity problem, even when routers use reverse poisoning. The triple loop of A-B-C results in infinite increases in route costs, until a value of 1000 is reached.

```
Format: <node 1>--<pre>--<post>--<node 2>
Where:
pre = cost before stabilisation
post = cost after stabilisation

A -1-1- B
B -1-1- C
C -1-1- A
C -1-1- D
D -1-1000- E
```

Path Routing

To resolve the count to infinity issue, I propagate complete route information with each distance vector sent by a node. When assembling a distance vector, nodes will not send a route to a neighbour that is already included in the path. This stops any loops forming in the graph. This prevents the count to infinity problem at the cost of larger message sizes and increased complexity of distance vector assembly.

This approach appears to resolve the issues in all networks I have tested.

Message Sizes

The maximum message size is bounded by the number of nodes in a network. For a network with n nodes, the maximum message size is in the order of n^2. This it acceptable when the total number of nodes in the assignment is limited, as in this case.

Testing

1. Run the supplied topology with the '-p' flag as the fourth argument. Note that the network stabilises once, and then never again.
2. Run the supplied topology with the '-e' flag as the fourth argument. This time, the network stabilises a second time quickly.
3. The enhancement should also be resilient to node failures.

The supplied run_poison.sh and run_extended.sh scripts will run the program using the supplied network, and will demonstrate the problem and the result of the solution.