Distance Vector in VNL Topology

Algorithm Overview

The Distance Vector algorithm is based on the Distributed Bellman-Ford Algorithm. Each router contains a routing table which has the following entries: destination, cost, and next hop. In this algorithm, a router periodically advertises its routing table (i.e., sending out routing updates) to neighbors (i.e., directly connected routers). Upon receiving the update, the router decides whether to update its routing table or not.

Updating entries in Routing Table

Given two routers *i* and *j*, and a destination *dest*, the following is the pseudo code for the algorithm to update routing table entry for (i, dest):

```
function updateCostAndNextHop(i, dest, j) {
  if (NextHop[i][dest] == j) {
    cost[i][dest] = cost[i][j] + cost[j][dest];
  } else if (cost[i][j] + cost[j][dest] < cost[i][dest]) {
    cost[i][dest] = cost[i][j] + cost[j][dest];
    NextHop[i][dest] = j;
  }
  // Else do nothing
}
```

This roughly translates to:

```
cost(i,dest)=min(cost(i,j)+cost(j,dest), cost(i,dest))
```

It tries to minimise the cost between a given router *i*, and a given destination *dest*. The algorithm checks whether updating the cost and next hop based on the intermediate node *j* would result in a lower cost, and if so, it makes the update. Otherwise, it leaves the current values unchanged.

VNL Topology

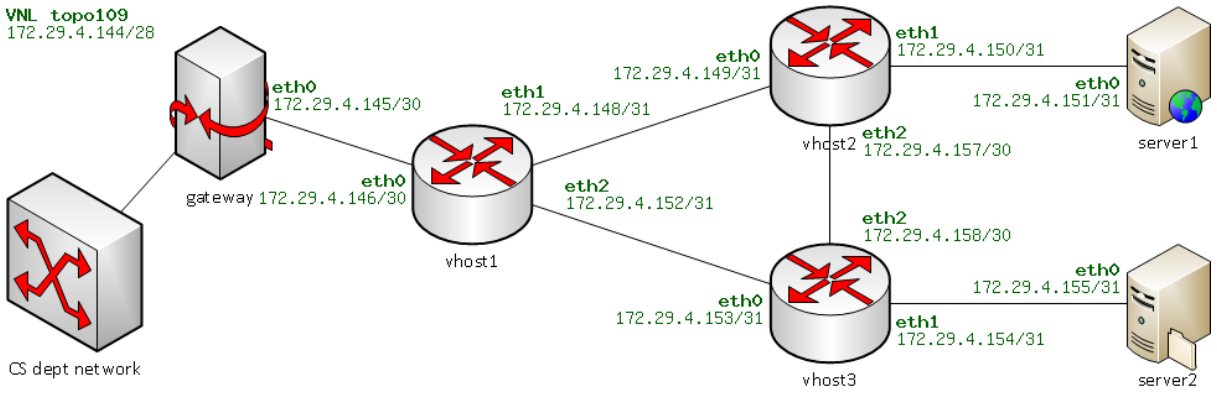

Fig. 1: Topology#109

This DV algorithm does not guarantee convergence to the shortest path, however, in our topology (three routers as shown in Fig. 1) the shortest path convergence is guaranteed as long as only one link fails among the three routers.

Implementation

The following are the high level details of the implementation.

- If the static rtable file is empty, each router fills entries in its routing table based on its interface entries.
- Each router sends periodic hello messages. The methods `send_hello_to_all_interfaces()` and `sendHelloPacket()` in sr_pwospf.c are responsible for this.
- Upon receiving a hello message from a neighbor, their details (time, sending-ip, interface) are stored in a neighbor struct. The method `add_neighbor()` in sr_router.c is responsible for this.
- Each router sends periodic LSU messages. These messages contain all the routing table entries of the router, which include destination, next-hop, and cost. The methods `send_all_lsu_updates()` and `sendupdatePacket()` in sr_pwospf.c are responsible for this.
- Upon receiving a lsu message from a neighbor, the routing table checks whether the received cost (destination cost + link cost between the router and the neighbor) is less or equal than the stored cost for the destination. If so, then the entry is updated with the received one with the neighbor as the next-hop. If there is no entry for the destination, then a new entry is added based on the received entry. The methods `add_lsu_to_rt()` in sr_router.c and `sr_add_rt_entry()` in sr_rt.c are responsible for this. *This step captures the essence of Distance Vector Algorithm.*
- If a neighbor timeout (i.e., its hello messages are not received after a timeout), then all its routing table entries (where this router is the next hop) are removed. A separate thread running the method `pwospf_run_thread_lsu()` in sr_pwospf.c is responsible for this. This helps the router to store update entries in the routing table in the case of a link failure.
-

The above implementation ensures that for a given destination, a shortest path is calculated in each of the routers.

Evaluation

Setup:

I tested the implementation using a testing script. The bash script ./check_link_failures.sh is designed to automate the testing of network topology changes by turning off links (vhost1-vhost2, vhost1-vhost3, and vhost2-vhost3) and pinging all the IP addresses in the topology.

Results:

All the 12 ip addresses were successfully pinged after each link-failure (vhost1-vhost2, vhost1-vhost3, and vhost2-vhost3). The routing tables of each router also reflected the updated shortest path after turning off and on each link.