Computer Networks and Applications

COMP3331 19T2

LSR REPORT **#python2.7**

Implementation of the LSR protocol.

There are two classes and four main threads for LSR protocol implementation.

- **Node Class:**
  A blueprint for creating node entity with required properties (i.e. node id, node port, related costs and ports of all neighbors) and methods.
- **Graph Class:**
  A blueprint for creating graph list with required properties (i.e. all existing nodes and related updated time of node) and methods. Graph list is a list with all node information and make node id to be the key.
- **Loop Thread:**
  Encode all required information of the node that gets from command config file as link-state packets and send it to localhost. And this target address session will run every UPDATE_INTERVAL.
- **Listen Thread:**
  Receive data from client, add the node information into graph class. And then transmit route to indirect connected neighbors.
- **Route Thread:**
  Every RPUTE_UPDATE_INTERVAL, use graph class to find if the node is the only node in network. If not, use Dijkstra Tread to calculate the least cost path to other routers in network and print it out.
- **Dijkstra Tread:**
  Calculate the least cost path to other routers in network and make other nodes in the network become aware that the failed node is unreachable.

Implemented Features

- Correct operation of the link state protocol
- Mechanism to restrict link-state broadcasts
- Appropriate handling of dead nodes, whereby the least-cost paths are updated to reflect the change in topology

- Detect the id and port number of the dead node that joins back
- Other nodes can detect the node when a dead node joins back the topology
- But I did not find the join back node cost and position in topology.

Data structure used to represent the network topology and the link-state packet format

I use graph class to represent the network topology, all node and its required information is recorded here. The sample data structure is:

{
Source: {Destation:cost, Destation:cost}
Source:{ Destation:cost, Destation:cost, Destation:cost, Destation:cost, Destation:cost }
}
	{
	'A': {'B': 6.5, 'F': 2.2}
	'B': {''A': 6.5, 'C': 1.1, 'D': 1.6, 'E': 3.2}
	}

Data structure used to represent the link-state packet format:

id	port	neighbors	status

Id refers to the id od node, port is the node port integer, neighbors refers all neighbors of the node with their id to be key, status can show if the node is failed.

Method to deal with node failures and restricts excessive link-state broadcasts

To deal with **node failures**, in update graph function, if another node information has not updated in 3 second, the node is removed. Neighbors of failed node can detect node failure and they will broadcast this info to their neighbors.

To **Restricting Link-state Broadcasts**, in transmit message function, we transmit route to in direct connected neighbors. For example, A->B->C, B will transmit A's route to C.

In receive excessive link-state broadcast case, when D also received packet from A, and will send it to B(A->D->B). B will not send it to C because C has already recorded it in function.

Discuss any design trade-offs considered and made. List what you consider is special about your implementation.

Indirect adjacent node can not detect node failure. Therefore, I make neighbors of the failed node broadcast this node failure info to their neighbors.

Deadlock can occur because multiple shared variables always be manipulated simultaneously. Thus, I sort resource requests in order which make the thread require node one by one in order. This is helpful to avoid deadlock.

Describe possible improvements and extensions to your program and indicate how you could realize them

When a dead node joins back the topology, its cost and relationship with other nodes may not be the same. I can realize it by read the config file and encode the new info and send it as link state packet.

Segments of code that have borrowed from the Web or other books.

UDPClient.py and UDPServer.py that provided in webcms3.

URL: https://webcms3.cse.unsw.edu.au/COMP3331/19T2/resources/27704