Link State Routing Protocol

By Joshua Shim z5479929

Implementation of the LSR protocol (1):

I have implemented the LSR protocol in python where each node reads its given configuration file and then repetitively sends and receives Link State Packets to and from its neighbours such that a Graph of the topology which the node is a part of can be formed. Then through a modified version of Djikstra’s algorithm it prints out the fastest route that can be taken to transfer a packet through the nodes.

Features Implemented: (all features successfully implemented)

- Link State Packet Sending: every one second the node sends a link state packet to each of its neighbours.
- Listening and relaying of Link State Packets: every node is able to receive and relay link state packets from and to its neighbours.
- Restricted Broadcasting: to reduce overhead each link state packet is only able to visit each node once.
- Route calculation: Djikstra’s algorithm is utilised to calculate the shortest route to each node.
- Node failure handling: failed nodes are addressed in route calculation.
- Node restart handling: previously dead nodes can come back alive and be re-added to the shortest route calculations.

Network topology, link state packets and broadcasting (2):

The data structure used to represent the network topology was an adjacency list implemented through a python dictionary such that information about each router could be received through the router’s id as the key. Within the adjacency list contains all of the necessary information about each node, including the adjacent nodes, their port numbers and the distance to the respective nodes.

The link state packet is formatted as a python object that is created with the fields origin (the router at which the link state packet is created), adjInfo (the information given in the configuration file). Furthermore, to check for dead nodes a time field is generated upon creation which contains the current system time which is used to calculate the disparity between the current packet and the last received packet to check whether certain nodes are down or not. There is also a visited dictionary which makes it so that the packet can only be sent to unvisited nodes.

Since there is a time field in the link state packet, every time a node receives a link state packet, it updates the time field in the adjacency list to the given time. Hence whether a node is currently dead or alive can be checked by subtracting the current time by the given time to see whether the disparity is greater than 3, at which point it can be concluded that the node is dead. In the dIjkstra's algorithm, there are certain checks to exclude dead nodes from being included in the calculations of shortest paths.

Excessive link-state broadcasts are prevented as whenever a node is visited, the link state packet object is modified such that the router is included in the visited dictionary. When relaying the link state packet is checked to see that the destination is not included in the visited dictionary.

Reflection (3):

I believe that my implementation of the LSR protocol fulfils all requirements without any downsides in respect to the assignment specifications. However when considering possible extensions and utilisations of this code for larger projects, there are some disadvantages to the way that I have written my code.

Trade-offs:

- Incoherent code style: In order to code this relatively simple project quickly, I have opted to disregard code style conventions as they were not necessary as stated in the forum.
- Correct system time is a requirement for the nodes to recognise whether a node is dead or not (this should not be a problem as if a router is connected to a network then the system time should be correct).
- Inextensibility: the way my nodes are programmed only allows dynamic change if all of the configuration files are changed to reflect said change, therefore it is not possible to randomly insert a node and to expect the program to deal with it. There is no system in place to change the topography (other than killing and restarting nodes) on runtime.
- Scalability: my implementation of this project is not very scalable, the algorithm used to calculate the shortest route and the relaying action becomes very inefficient when the network topology becomes much larger.

Why is my implementation Special?

My implementation is special due to its simplicity and efficiency in achieving its goal of emulating the link state routing protocol. My use of dictionaries in representing the network topology allowed me to form a good basis on which the further algorithms and features of this project could be implemented.

Furthermore my modification of Djikstra's shortest path algorithm was very effective as it allowed for the easy consideration of nodes turning off and on as I used a double check with a onNodes and offNodes list such that only nodes within onNodes could be included in the calculations and nodes in offNodes would be avoided every time.

Possible improvements and extensions:

- Improved code style - increased legibility and maintainability of code.
- Heartbeat packets - to increase reactivity to dead or newly alive nodes. To realise this a new thread could be formed to specifically send heartbeat packets at a faster rate than LSP’s such that it is more evident when a router is dead.
- Dynamism - I could possibly change my implementation such that if a new node was to be added on runtime, then all of the present nodes could override their current information to include the new node. This would be done by changing the listening function such that the relayed information would update adjacencies instead of merely establishing adjacencies as a graph (adjacency list) when the program begins.

Code Attribution (4):

None of the code that I have written has been borrowed from the web or any books.