1. Overview

This LSR (Link State Router) is consists of a state class and router class. The router class is the main program which instantiates an immutable state and performs operations on the said state, returning a new state. Figure 1 below shows the relationship between the router and state. The state class contains the necessary utilities required to mutate the network topology (i.e. essentially all the graph operations). This design was inspired by the concept of state machine. I chose this design as it was easier when debugging.

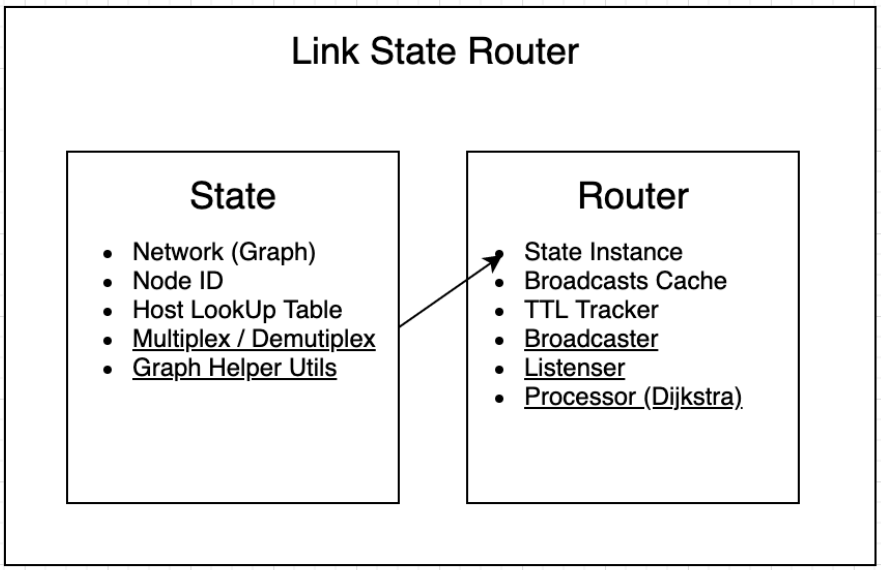

Figure 1. Program Representation

Upon initialisation, the router's config file is passed to the setup function, which parses the file and constructs the initial network topology and Link State Packet (LSP). After setup, we are now ready to run the router. On execution of the router, the four main services are launched on a new daemon threads until the user exits the main thread. Once the user terminates the main thread all its child services are shutdown automatically. This prevents the main router services from running in the background, after the program terminates. Below, we'll discuss the responsibilities of the router's four main services.

1. **Broadcaster**
    - Transmits the following packets to the router's neighbours via UDP
        - Link State Packet (of the current router, every UPDATE_INTERVAL)
        - Hello Packet (which tells router's neighbours that it's still alive)

2. **Listener**
    - Waits for link state packets, so we can update the network.
    - Forwards link state packets to other routers, other than the router in which the packet originated.
    - Resets the TTL of a given router if a Hello Packet has been detected.

3. **Updater**
    - Decrements the TTL of each neighbour.
    - Deleted a host from the network when a neighbour's TTL expires.
    - This service is run every UPDATE_INTERVAL.

4. **Processor**
    - Executes Dijkstra using a snapshot of the most recent network topology.
    - Prints out the response, according to the specifications.

- This service runs every ROUTE_UPDATE_INTERVAL (i.e. 30s)

Features

The following features have been implemented and tested on several topologies:

- Link State Packet Broadcasting
- Link State Packet Forwarding
- Byte level multiplexing/demultiplexing
- Caching broadcasts, to prevent repetitive transmissions.
  - Cache for specific hosts are reset when the host fails.
- One thread per main router service.

2. Data Structures

The following data structures described below were used to represent the network topology and other essential information.

Network Topology

An adjacency list, through the use of nested python dictionaries, was used to represent the network topology. We'll discuss why an adjacency list was preferred over an adjacency matrix and edge list in the design section. The graph below in the bottom left is the test topology extracted from the specifications. The json dump on the right, shows how the network is represented using dictionaries.

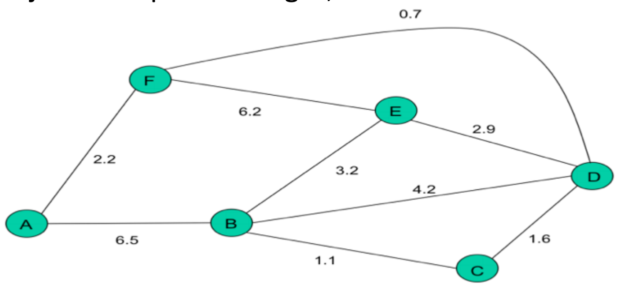

Figure 1: Test topology

```
{
   "A": {"F": 2.2},
   "B": {"C": 1.1, D": 4.2},
   "C": {"B": 1.1, "D": 1.6},
   "D": {"B": 4.2, "C": 1.6, "E": 2.9, "F": 0.7},
   "E": {"D": 2.9, "F": 6.2},
   "F": {"A": 2.2, "D": 0.7, "E": 6.2}
}
```

Packet Format

Hello packets and LSPs (i.e. `MSG_TYPE_HELL` and `MSG_TYPE_STATE` respectively) are the two types of messages the router produces. The router encodes every message using python's pickle data structure, which transforms the message into a byte stream. This data format is more efficient when transmitting over the network. The encoded message is later decoded using pickle's loads function. Both message types (hello and LSP) are wrapped in the following json

{ “msg_type”: <HELLO | LSP>, “payload”: <NULL | Serialised LSP>, “owner”: <MSG_OWNER_ID> }

The listening service processes both types of messages, differently. The message owner id is used by the listener, when link state packets needs to be propagated immediately. We can prevent sending the packet back to the original owner using the MSG_OWNER_ID.

The link state packet's payload is wrapped in the following JSON:

{ “src”: <Router ID>, “links”: <Routers Neighbour Nodes + Weights>, “info”: <host look table> }

Handling Node Failures

The Hello Protocol is used to tell neighbouring routers that the current router is still alive. Every second, UPDATE_INTERVAL, the router broadcasts a hello packet (i.e. HELLO <routerID>), which tells neighbouring routers to reset their TTL back to the maximum value (i.e. TTL = 3).

Every second, every router decrements the TTL values of their neighbouring router. If a router expires (i.e. TTL = 0), the router deletes the node of the network. However, the host information remains, in case the router needs to retransmit this information to their neighbouring routers.

When a router re-enters the network, it starts sending its hello packet to other active nodes, notifying them to reset the router's TTL and re-add to the network.

Handling Excessive Broadcasts

All broadcasts to neighbours are cached using a hash table (i.e. Python dictionary). The hash table's keys are the host IDs. Every time, we check if we need to transmit a packet, we check if the current host's set contains the packet. If it does, we ignore transmission. The broadcast cache is reset if the router dies, because if that router respawns, we need to reforward its LSP.

3. Design Discussion

Design Trade-offs

In-order to represent the network topology, we had to use some sort of data structure to represent a graph. An adjacency matrix, adjacency list and edge list are common methods of representing graphs. Due to the dynamic nature of this system (i.e. Routers randomly connect and disconnect), the decision to go with an adjacency list was made. This is because creating and removing links are O(1) when using an adjacency list. However, finding the presence of a link was O(n), where n is the number of nodes in a link. However, for this use case, it was better to use an adjacency list over matrix, because we're are more likely to use those operations. If the system heavily replied on link lookups over dynamically adding routers, the choice of going with an adjacency matrix would have made more sense. Also, by using nest python dictionaries, we don't need to iterate over each neighbour, thus O(1) for lookups, insertions and deletions.

Implementation Extras

Below is a list of additional functional and non-functional requirements that had been implemented:

- Services queue, where we can easily create new threaded services on router start-up.
- Encapsulating the routers state and using private methods (in order to prevent misuse of the API)
- TTL updater running on a separate thread.
- State representation function <def __repr__> (so we can print out the network state when debugging).

Possible Improvements

Below is a list of several improvements that can be made to the system, in order to improve efficiency and code maintainability.

- Using a priority queue (using python's heapq library) for Dijkstra computation, to amortise the sorting cost, required to find the next minimum node to visit. This will greatly improve the time complexity of the Dijkstra algorithm as we don't need to sort the to_visit list on each iteration.
- Better Error Handling – instead of using try-catch blocks, actually take into account edge cases (i.e. when Dijkstra is run on a network with nodes that aren't connected, should be handled properly rather than ignoring the lookup key error).

- Broadcast Queue – all packet transmissions are added a priority queue where LSP forward transmissions are prioritised over regular transmissions. Each transmission can also run on a separate thread. Cache control can be handled on the main thread used to control the transmission threads (to prevent cache collision).
- More efficient method of taking snapshots of the network topology, rather than using a shallow copy of the dictionary. This was done to enforce state immutability.

4. Acknowledgements

Pickle Serialisation

- https://www.bogotobogo.com/python/python_serialization_pickle_json.php

Hello Protocol

- https://www.sciencedirect.com/topics/computer-science/hello-protocol
- https://www.quora.com/What-Is-The-Hello-Protocol-Used-For-in-Routers

Dijkstra

- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://stackoverflow.com/questions/40871864/dijkstra-s-algorithm-in-python