## 1. A brief discussion of how you have implemented the LSR protocol. Provide a list of features that you have successfully implemented.

1.1. LSR is implemented with Python 3.6. Please use Python 3.6 to run the codes.

The main component of my implementation is the Router class. It provides high APIs such as broadcast\_link\_state\_packets, check\_node\_failure and report\_shortest\_path. Codes in Lsr.py will read a configuration txt file and control periodic events using these APIs, namely neighbor information broadcast, node failure detection and shortest path report.

The process and replay of any incoming UDP packet however, is governed by a Server object within each Router instance.

1.2. ALL the features mentioned in the assignment pdf file are implemented, which are listed as follows:

1. Send link-state packet to a neighbor
2. Replay link-state packet
3. Avoid redundant replay
4. Construct network topology
5. Compute shortest path with Dijkstra’s algorithm
6. Deal with failed nodes and rejoining nodes

## 2. Describe the data structure used to represent the network topology and the link-state packet format. Comment on how your program deals with node failures and restricts excessive link-state broadcasts.

2.1. A list of (from, to, cost) tuples are used to represent the edges as well as the network topology. Link-state packet is a json string in following format: {"sender": "A", "neighbours": [["B", 6.5, 5009], ["C", 2.2, 5010]], "timestamp": 1563794768.6184244}.

2.2. By utilizing sender and timestamp information in the packet, node failure detection is implemented and excessive link-state broadcasts is avoided.

When receiving a packet, the router first check whether this packet has been relayed by comparing sender and timestamp, if not, record it and relay. Notice that in such way new packet will not be ignored (and should not be).

Concretely, a map in which the key is sender and the value is the timestamp of the last relayed packet from that sender is used.

With this map, node failure can also be detected. For instance, at time t, scan every pairs in the map, for any timestamp such that t – timestamp > patient (in this assignment, 3 seconds), the corresponding sender is considered a failed one.

## 3. Discuss any design trade-offs considered and made. List what you consider is special about your implementation. Describe possible improvements and extensions to your program and indicate how you could realize them.

3.1. Sender and timestamp information is an overhead cost to deal with node failure and redundant packet relay. Considering that the neighbor number is small, they largen the packet size quite significantly.

I think a formal analysis of the total information that needs transmit with respect to the average neighbor number and the density of the network topology will be helpful.

3.2. The Server class is quite unique and difficult to implement. As a UDP server generally contains an infinite loop, Server object is a thread itself and this made the code cleaner than a non-blocking packet pooling design.

3.3. I think a switch to exclude sender information and timestamp would be conducive the analysis mentioned in 3.1 proves that sometimes it is more efficient to just let the redundant relay happens. Another side effect however, will be that the failure of nodes cannot be detected. It might be helpful in some unique robust networks.

To realize them, a function indicating which mode is better must be designed. When the switch is off, any incoming UDP packet will be relay by the router.

## 4. Indicate any segments of code that you have borrowed from the Web or other books.

4.1 None taken.