Assignment Report

WEI SONG
z5198433
02/08/2019

Part1. Overview

Pyhton3 is used to complete this assignment. The goal of the assignment is to implement the LSR protocol. The most import modular I use is Thread from Python3 standard library. And use JSON format data to implement the data transmission, and transport layer is UDP protocol. There are total five threads implementing different tasks simultaneously, like sending data, receiving data and other tasks.

How to implement the LSR protocol?

In my implementation, except reading configure file and handle the config info, I totally defined five thread tasks to perform different tasks at the same time.
The first one is sending LS packet thread, which sends LS packet to all neighbors every second.
The second one is receiving packet thread, which receives packet including LS packet and heartbeat packet from neighbors, and then transmit to other neighbors except the sending one if the packet is latest.
The third one is sending heartbeat thread, which sends heartbeat packet to neighbors every second.
The fourth one is handling the failure routers and new join routers, which should check the topology graph and update the topology if failure events happen.
The fifth thread is running the Dijkstra Algorithm and finding the smallest cost path every 30 seconds.

Some features of my implementation

1.Basic LSR 2. Sending and updating link state periodically 3. The timestamp mechanism to reduce the packet sending 4. The Heartbeat mechanism to detect the failed nodes and handling. 5. UDP protocol for data packets transmission.

Part2. Data Structure

For a node, its link state data structure is a dictionary, the key is its ID, the value is a list including a dictionary (store all linked nodes, key is linked node ID, value is all information of this node), timestamp (time of creating this data packet) and alive flag:

```
# a router link state:
# is a dictionary, the key is the router ID and the value is a list
# three parts in the list:
# 1. a dictionary including all linked nodes:ID, port, timestamp, alive flag for link node
# 2. timestamp for this link state
# 3. alive flag for this router
{'A':[ {'B': [cost, port, timestamp, alive],  'C': [cost, port, timestamp, alive]}, timestamp, aliveFlag]}
```

But when a node sends or transmits data packets, it will encapsulate the data with its ID (sender ID), which receiver should know the sender and will not transmit the packet back. And when receiver receive this packet, it will take the data out, update its topology

and transmit this data if necessary.

```
# original LS data structure
data = {'A':[{'B': [cost, port, timestamp, alive], 'C': [cost, port, timestamp, alive]}, timestamp, ailveFlag]}
# packet after encapsulation
{'senderID': data}
```

Network topology data structure:
A list contains all nodes' LS, which after stable all online nodes should have the same topology. An example like below:

```
# router A's topology graph
[
    {'A':[ {'B': [cost, port, timestamp, alive],  'C': [cost, port, timestamp, alive]}, timestamp, aliveFlag]},
    {'B':[ {'A': [cost, port, timestamp, alive]}, timestamp, aliveFlag] },
    {'C':[ {'A': [cost, port, timestamp, alive]}, timestamp, aliveFlag] }
]
```

Deal with failure: every second, the router should receive one heartbeat packet from its each neighbor, and update the value of alive (alive += 1), and transmit the packet to other neighbors immediately if the packet is latest, and every 3 seconds the router will check how many heartbeat packets of its immediate neighbors send to it (record in alive filed), if the number is 0, that means the router have not received 3 consecutive heartbeat packets of its neighbor, neighbor is down. The router will delete that neighbor's information from its link state, but if the number is not 0 the router will reset it to 0 and waiting for new heartbeat. While for its indirect linked nodes, when the router receive their heartbeat packets from its neighbors, the router will update the value of aliveFlag (aliveFlag += 1), and every 3 seconds the router will check the alive flag of its linked state packet, if alive flag is 0, means this node is down, the router will delete its link state data from the topology, else the router will reset this value to 0 and waiting the new heartbeat.

When a node restart, it should begin sending heartbeat packets and LS packets to its neighbors again, and its neighbors will add its information to their LS again, and they will transmit these packets to other nodes. And finally, all other online nodes should know the node restart information because of beginning to receive its heartbeat packets again, and also they will get its LS from its neighbors and update the new join node's LS into their topology graph again.

Restrict excessive link-state packet broadcast: as you can see from data structure, there is a timestamp field, which records the latest time of creating the packet (from system time). Thus, when the router receives a packet, if the router does not hold this packet, it will put this packet into its topology directly, but if the router has this packet, it will check the timestamp, and if the packet is a newer packet (its timestamp bigger than the same packet the router holds) then the router update the old packet with the new packet and transmit the new packet to other neighbors else the router will discard this packet and will not transmit.

Part3. Trade-Off and Future Improvements

Trade-Off: as you can see from the above, the data structure is a little complex and

redundant, which use much memory to store almost all original info. This can improve the accuracy, stability and sensitivity of my program because every node exactly knows other nodes information, alive state and timestamp, but it cost memory.

Special implementation: data structure can make sure the data is organized properly and integrally, which provide the foundation for LSR protocol, five simultaneous thread tasks can make sure all the tasks can run simultaneously, and heartbeat mechanism help to sense the failure nodes and update the information very timely, which can make sure that all nodes can compute a correct routing table using the latest information.

Possible improvements: in the Dijkstra Algorithm, I use the iteration to find the current least cost and unvisited node, which make the time complexity is $O(n^2)$.This can be improved by using the heap data structure, which can improve the time complexity of the Dijkstra Algorithm to $O(\text{nlog(n)})$.

Part4. Reference

[1] The idea of Dijkstra Algorithm borrowed from lecture note
https://webcms3.cse.unsw.edu.au/COMP3331/19T2/resources/26451
[2] Python3 UDP socket programming
http://www.runoob.com/python3/python3-socket.html