COMP 9331

Computer Networks and Applications

Assignment: Link State Routing Protocol

Language and Platform: Python3

Implementation:

1. Broadcast Thread：

Every second, each router broadcasts its information which contains source node, neighbors and path to its neighbors.

2. Listen Thread:

Listening link-state message from other nodes. Listen Thread is also responsible for updating the network topology and transfers the message to its neighbors when necessary.

2.1 Heartbeat:

The heartbeat can help the current route to have a certain understanding of the current network topology. Heartbeat updates within 3 seconds are acceptable. If message arrivals beyond this time are considered "Node Failures".

2.1 Retransfer:

Transferring the message to the current router's neighbors when necessary.

3. Dijkstra Thread:

Calculating the least cost and the path to the router (except the router itself).

4. Processing File Thread:

Processing each instance of the routing protocol's data.

```
{'source': 'A', 'neighbour': {'B': ['6.5', '5001'], 'F': ['2.2', '5005']}, 'path': ['A', 'B', 'F']}
```

Features:

1	Correctly parsing and processing the information of each router.
2	The broadcast mechanism is capable of transmitting a link state packet to each neighbor of the current route every UPDATE_INTERVAL.
3	The retransferring mechanism is capable of retransferring messages. contained in its neighbor broadcast packets which indicates that the node will regard this message as a forwarded message.
4	All Information sent and received by using UDP as the transport protocol.
5	Setting up a global view to illustrate the whole network topology.
6	Applying Dijkstra algorithm to calculate the shortest path and cost of the path every ROUTE_UPDATE_INTERVAL.
7	Restricting Link-state Broadcasts by using a path in the message to reduce unnecessary retransferring.
8	Handling routing dynamics by using
9	Least-cost path computation from alive nodes is able to remove failed nodes.
10	Least-cost path computation from alive nodes is able to add failed nodes.

Data structure:

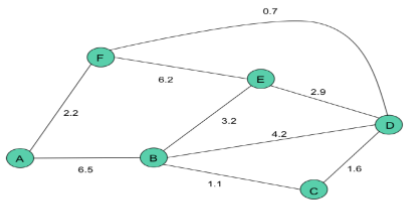

Figure 1: Test topology

Network topology:

The data structure used to represent the network topology is a global dictionary called graph which contains alive node's information.

{ sourceNode: { neighbor1 : [cost, portNum], neighbor2 : [cost, portNum] }}

```
graph: {'A': {'B': ['6.5', '5001'], 'F': ['2.2', '5005']}, 'B': {'A': ['6.5', '5000'], 'C': ['1
.1', '5002'], 'D': ['4.2', '5003'], 'E': ['3.2', '5004']}, 'C': {'B': ['1.1', '5001'], 'D': ['1
.6', '5003']}, 'D': {'F': ['0.7', '5005'], 'B': ['4.2', '5001'], 'C': ['1.6', '5002'], 'E': ['2
.9', '5004']}, 'E': {'B': ['3.2', '5001'], 'D': ['2.9', '5003'], 'F': ['6.2', '5005']}, 'F': {'
A': ['2.2', '5000'], 'D': ['0.7', '5003'], 'E': ['6.2', '5004']}}
```

Node Failures:

If a node fails in sending its message to its neighbors, its neighbor can detect this situation and quickly remove this node's information in the global view by using heartbeat approach. There is a dictionary called heartbeat whose key are the current node's name and received nodes' name and values are the time when the current node listened the other nodes' messages.

```
heartbeat:  {'A': 1565143051.089731, 'C': 1565143051.14013, 'D': 15
65143050.866137, 'E': 1565143050.4414341, 'F': 1565143049.992301}
```

The value of the heartbeat dictionary is incremented by one every second. When the value of a key has not been updated for more than three seconds, it indicates that the key is a failed node. When a node fails, this node's information will be automatically deleted by the graph by using a heartbeat method.

Restricts excessive link-state broadcasts:

A broadcasts its information to B with the path [A, B, F] that A needs to send. When B listens A's information, it will transfer to it neighbors [A, E, D, C]. I create a new list call transNodes[] for B to transfer. If B's neighbors are not in A's path, I add these neighbors in in transNodes [E, D, C] and add neighbors in path in A's information. The path in A's information becomes [A, B, F, E, D, C]. After that, B will transfer the nodes which in the transNodes list. When E listens the information from source A, which retransferred by B, it will find the path in this information's path contains [A, B, F, E, D, C]. It will not retransfer A's information to D. This will restrict excessive link-state broadcasts.

Design trade-offs considered and made:

1. Use of the Pickle module：

Almost all data types, for example list, classes, dictionaries, etc., can be serialized by pickle. The pickle.dumps used before transmission compresses the dictionary information of the current router. On the receiving side, pickle.load is used to extract the obtained packet information.

2. Use of the Time module：

The heartbeat method uses the time module to process time-related tasks to determine the state of each node. This state refers to whether the node is active or failed.

Improvement:

In a real network, if a node sends the same message only once, UDP cannot ensure that the forwarded router receives the retransferred message. No ACK is added to the programming to ensure that the sender knows that the message of this router has been retransferred to the forwarded router. An ACK needs to be considered.

Learning from lecture notes:

UDP socket programming

```
sender = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)#ipv4 and UDP
```