18741 Design Document for Project 2
Content Distribution Design Document

Chenxi Li (AndrewID: chenxili)

1. **Read Configuration File:**
   In this part, a configuration file will be read based on file path provided by the command line. As some of the options may not exist in the configuration file, a HashMap would be used to store options like "uuid" as key and its exact value as value. After reading the file, the existence of "uuid" would be checked. A new "uuid" would be generated if this option is missing and configuration file would be updated.

   A class called "Node", which is used for representing the router/node in the network, has the function of serializing and de-serializing so that a node object can be generated through String format. A "uuid" is required for constructing a "Node" object, while there are some other "setter" functions to set the values for other options. The naming rule for each node is "node" plus the first three character of its uuid.

2. **Node Identifier:**
   A "Scanner" object is created for accepting the "System Input" from the command line. When user input "uuid", the information of the current node's "uuid" can be extracted from the HashMap we created before and would be printed in "JSON" format.

3. **Reachability:**
   In this part, each node is required to send a "keepalive" message to its neighbors to ask them if they are still reachable, and should also be able to listen the message sent from its neighbors at the same time. A HashMap called "aliveNeighbors" in the Node class is used to store the current reachable node as "uuid – Node" pair.

   Here the UDP backend protocol is used to handle the reachability messages. Each node would have one thread to run as a server to keep listening the "keepalive" message sent from its neighbors in its own backend port. Every time the server receives a message, it should send back a confirmation message back to the sender to tell them it is still reachable. Each node would have multiple threads to run as clients to send "keepalive" message to its neighbors every 10 seconds. If the client does not receive the confirmation message from the target server for three times, this neighbor would be declared as unreachable and this neighbor would be removed from the "aliveNeighbors" and the configuration files would be updated at the same time. If there is a new alive neighbor that does not exist in the configuration file but start to send message to the node's server,

the server would detect this and add this new neighbor into “aliveNeighbors” and update the configuration files at the same time.

User can also type “addneighbor” into the command line to manually add a new neighbor to the current node by providing the new neighbor’s information.

4. **Peer Discovery and Link State Advertisement**

In order to construct a complete network map, each node in this network should not only know its neighbors’ information, but also know its neighbors’ neighbors’ information. In order to do that, “link-state” protocol is used by implementing UDP as backend.

First, a class called LSPacket is used for representing the packet that would be sent in the “link-state” protocol. An attribute called “router” that served as the sender of this packet. An attribute called “seqNum” that represent the sequence number of this packet. And an attribute called “nodeList” stores the current alive neighbors of the router. Sequence number is used for keeping the freshness of the message as each node needs to know the most recent relationship condition of other nodes. A server would not accept a packet with a smaller sequence number to avoid accepting outdated message.

An attribute called LSDB in Node class is used to store the most recent packet belongs to other nodes. LSDB stores packets as HashMap uuid-packet pair format.

A class called “LSPThread” is used to send node’s own packet to its neighbor. Each node would have multiple “LSPThread” served as clients to send packets to its neighbors every 10 seconds with ascending sequence number. Also, a neighbor would be declared unreachable if it missed the packet for three times.

In order to efficiently distribute the LSPackets of all routers, the “Flooding” algorithm is used here to complete this task. When router receive a LSPacket, it first verifies if this LSPacket has already stored in the LSDB (by comparing the sequence number). If not, the router would first update the LSDB and then forward this packet to other links it has. In this way, each router can not only learn its direct neighbors, but also can get other nodes’ information by the packet that forwarded by its direct neighbors. So that each node can receive all the LSPackets in the network and be able to construct a complete map of the whole network.

To implement the “Flooding” algorithm, a class called “FloodThread” is designed for forwarding all the packets in LSDB to the node’s neighbors. As LSPacket class has

serialization and de-serialization method, a packet can be sent in String format through UDP backend protocol. A main thread class called “ForwardThread” is used to generate multiple new client “FloodThread” threads to send all the packets to each of its neighbors. The server thread of each nodes would receive these forwarded packets and update their own LSDB when they receive the new packets.

5. **Priority Selection**

   In this part, a routing table that shows the shortest distance between current node and all other nodes would be generated based on “Dijkstra” algorithm. As the distance metric between two nodes has already been given in the configuration file and the complete map of the entire network can be constructed in the previous task, we have enough required information to implement “Dijkstra” algorithm to calculate the shortest path. Based on the “Dijkstra” algorithm, we can have an array to store the current shortest path from the current node to all other nodes and a “PriorityQueue” to sort the paths. In order to sort the nodes based on distance metric, we can implement “Comparator” interface and override the “compare” method.

6. **Kill**

   Before exit the program, we need manually kill all the server, client or any other running threads. As the threads are designed to keep running while a Boolean variable “flag” is true. So simply set the “flag” to be false to stop all the threads. Same with the “Scanner” object that keeps receiving the input from the command line, simply set the “flag” to be false to jump out of the while loop and exit the program.

7. **About the specific questions:**

   a. ***Did you use the backend protocol (udp) to handle advertisement/reachability messages or something else?***

      Yes, both advertisement and reachability are used UDP as the backend protocol. Please see details about handling process in the previous sections.

   b. ***What method did you use to perform link-state advertisements and priority selection?***

      In link-state advertisements, the “Flooding” algorithm is used to efficiently distribute the LSPackets of all routers. While in priority selection, “Dijkstra” algorithm is used to calculate the shortest path between target node and all other nodes to construct the routing table.