18441/18741: Design report for content distribution project

Name: Xinyu Bao Andrew ID: xinyub

1. Project structure

The main function and large bulk of codes are in the contentserver.cc file. At the same time, there are 4 helper classes, CurrentNode, PeerNode, MapItem, and PeerInfo, all of which are in the corresponding hpp files. There are 3 helper functions, too (just want to make sure that the main file doesn't have too many lines), *sendMessage* for sending udp packages, *recvMessage* for receiving udp packages, and *rankByMetric* for running the shortest path algorithm. There is also a Makefile and several configuration files for compiling and testing purpose.

2. Basic running logic

The main program first loads the configuration file based on the input filename. The information is stored in a CurrentNode instance, myHost, along with neighbor nodes' information. Then four child threads are created and run: one for the timer purpose of the keepalive message, one for sending keepalive messages, one for sending link-state messages, and one for receiving all messages. Particularly in the thread that receives messages, new grandchild threads will be created once a new message comes in. This design is intended to separate receiving and handling process, which can ensure much more messages to be received at the same time in case there's a burst of messages in the network. Finally the main program listens input from keyboard and perform different actions based on the input command.

3. Some implementation details

For keepalive message, a neighbor is considered unreachable if three consecutive messages are lost, and keepalive is sent every 10 seconds. This is implemented by the timer thread. The keepalive message contains all necessary information for a host to be added as a neighbor for another host. Therefore, if a host receives a keepalive from a previously unknown node, the host is able to add this new node to its neighbors and starts tracking its reachability. Also the name of each neighbor is sent through keepalive messages so that each node should get its neighbors' names pretty fast. Keepalive messages are sent to all potential neighbors, regardless of their reachabilities.

A link-state message is sent both by triggering and periodically. There are several situations that can trigger a node to send a new link-state message. For example, a new neighbor found, a previous neighbor lost, new items added in the network map, and old items deleted in the map. By triggering, generally all nodes are able to get the entire map in less about 5 seconds in most cases. However, in some cases, it is possible that several messages come at the port at the same time, and the host is not able to get them all. This situation is greatly prevented by separating receiving and handling messages, but just in case link-state messages are also sent periodically, with a longer interval (60 seconds in my implementation). Link-state messages are sent to

active neighbors only.
Each node maintains a network map which is comprised of multiple MapItem instances. Every time a host detects change in the network, such as a new link-state message, adding/deleting of neighbors, the map is updated and a link-state message is sent. When “rank” command is invoked, the host would generate a newest routing table based on its current network map, with each node and their least cost respectively, ranked by the cost. Each node is able to detect any node/link’s addition and deletion in the network, and is designed to be able to update their map quickly enough, generally in less than a few seconds.

4. A note about the design

In response to questions mentioned in handout, communication between nodes is enabled using the udp protocol. Since this protocol is light and handy, I consider it better than tcp in this application where lots of messages are supposed to be in the network.

5. Extra instructions on code execution

The executable file should be easy to generate with a simple “make” command. The configuration file should be strictly formatted as it is in the handout (including whitespaces). Also, the addneighbor command should have the same format as specified in the handout. I didn’t add much format check to both the configuration file reading and addneighbor command processing.
My nodes should be able to detect neighbors, response to nodes adding/deleting all over the network, and generating the newest map pretty quickly. However, if there’s something wrong with one or two node’s map output, please give it another 60 seconds. This may be due to a packet lost or port collision, which is rare though. Thank you in advance.
I didn’t implement the active metric due to time constrain. I think I spend too much time on basic thread and network programming in C, and the debug process combined with keepalive and link-state message is a lot of pain. I’m glad I made it to the end.