Nathan Ackerman, Paul Elder
2520 Project 1 sLSRP Design

Introduction

There will be two separate main components, one for the user interface interactions and one for administering the routing protocol. These will both be implemented in C. We will refer to the routing component as routing.c and the user interface as ui.c. ui.c will interact with routing.c through a socket. routing.c will be in charge of everything related to the routing protocol.

routing.c will have three ways to start up. It can be started with a config file, with an interactive mode to set configurations, or default configurations. In all cases (either via config or cli option), the id of the router must be specified, and it must be unique. The id cannot be changed at runtime. The configuration will contain whitelist, blacklist, router_id, slsrp version number, hello_interval, update_interval, etc. These configuration option may be changed during runtime. Neighbours must be specified at runtime (ie. they cannot be specified at startup time).

Routing component

routing.c will have components that will be implemented as handlers. The handlers are for updating the routing table, sending Link State Advertisements to neighbors, handling data arriving in a socket from a neighbor or a potential neighbor, sending link cost messages, sending alive messages, and dealing with ui.c requests.

There will be an ack waitlist, called ackwaitlist, which is a table of messages that we are waiting for an ack for (implemented as list of timers with callbacks and corresponding data). Each entry will contain the function to be run when the timer expires, the timer expiration time, and number of attempted tries (die after 3 tries). These entries will be replayed if past their expiration. Upon receipt of an ack, the entry will be removed from the ackwaitlist. Any messages sent will be added to the ackwaitlist with the proper function to be re-run and for brevity we will not discuss this again when overviewing the handlers and the messages they will send. This functionality will ensure that each of the messages are correctly delivered. The handlers will also send ack's of any messages they receive. Right before the rerun, the liveness of the neighbour will be confirmed, and the entry removed if the neighbour is dead.

The handler for updating the routing table will be triggered by a timer. This will use the information in the Link State Database to run Dijkstra's to compute the routing table.

The handler for sending Link State Advertisements will be based on a timer set to Update_Interval. This will cause routing.c to send its updated information about the links to all of its neighbors. The router will keep track of its most recently sent out sequence number, and if an older LSA is attempted to be sent out due to retrying for an ack, then drop the old LSA.

The handler for sending the alive messages will be a timer based on the Hello_Interval that it needs to send to each of its neighbors. Upon not receiving acknowledgements of 3 such messages, the router will assume its neighbor is down. If the neighbor wishes to use a connection with the router it must establish a new connection. The interval between sending a new alive message instance must be greater than 3 times the hello_interval (to prevent multiple concurrent alive messages waiting to be acked).

The handler for sending link cost messages will also run on a timer. This will map out the cost of the links for all of the router's neighbors and will update the Link State Database respectively which will be spread through the next LSA. The update will happen after a timer set to Update_Interval. An entry will be added to the ackwaitlist, but the timer interval will be a timeout value, and the handler will kill the LCM. The interval between LCMs must be greater than the timeout value.

Routing component - socket listener

The handler for data arriving at the socket will be triggered by that event. When the event triggers, the header will be read from the packet and the type of message will be determined. This will determine how much more to read from the socket and what to do. This will include the processing of LSA's, receiving and responding to alive messages, receiving neighbor requests, and receiving link cost messages.

- Processing of an ack:
  - Upon receiving an ack, the router must remove the corresponding entry in ackwaitlist.

- Processing of LSA's:
  - Upon receiving an LSA, the handler will update its information in its LSD (described below) and send to all of its neighbors except for the one who sent it to this router.
  - Drop LSA's that we've already seen (ie. same router id and sequence number) since multiple neighbors could send the same LSA if they are both neighbors of another node.

- Processing of alive messages:
  - Upon receiving an alive message, the router must respond to that router with an ack of that message.

- Processing of neighbor requests:
  - Upon receiving a neighbor request, the router must send a confirm or refuse to the neighbor based on its configuration.

- Processing of link cost messages:
  - Upon receiving a link cost message, the router will add an entry to a table corresponding to this link which will be averaged upon the Update_Interval completing.

UI component

The user interface will be a simple repl (and therefore will be interactive). The commands that will be available are configuration management (eg. adding/removing items from the white/blacklist, changing values, etc), configuration loading, configuration saving, dumping the routing table (both in stdout and to a file), and dumping the LSD (also both to stdout and to a file). The ui will not support single-command via cli parameter because it can be specified by stdin redirection. The ui will communicate with routing.c via a socket, so a router can be administered either remotely or through localhost. A help menu will be available for any of the functionalities provided.

LSD

For the Link State Database, we will use a library called igraph to store a local copy of the LSD. Upon receiving an LSA, the router will update its LSD with the required information. Link State Advertisements will send the information about that router's edges (its local topology), that it has calculated through the link cost messages, to its neighbors. Upon receiving an LSA, the LSD is updated if the sequence number is greater (newer) than the sequence number previously stored with that edge. This ensures the LSD always has the newest information for when it needs to build the routing table. Our motivation for this design of the LSD was simplicity of maintenance and computation of the routing table.