Nathan Ackerman, Paul Elder

2520 Project 1 sLSRP Design

Version 2

Introduction

The functionality will be separated between into two distinct parts, the routing component (routing.c) and the ui component (ui.c). routing.c will start based on a configuration file that will specify its hello_interval, update_interval, and any neighbors it should start with. These may be changed at a later time through the ui.c component.

Routing Component

The routing component will split into different threads that will each be responsible for different activities. One thread will be the listener. The listener will continuously read from the socket and will be responsible for updating information and dispatching threads to deal with work. All communication coming into a router will be sent through the same socket. Communication going out to each of the routers neighbors will have a different socket for each neighbor. The listener will receive the following types of packets: neighbor requests, responses to neighbor requests, lsa messages, acknowledgements of lsa's, alive messages, acks of its own alive messages, link down messages, link cost ack messages, and control tasks from the ui component.

Actions for the Listener

When the listener receives a neighbor request, it accepts or denies based on config settings. If it accepts, it must send the response to that new neighbor and add the neighbor to its list of neighbors. When the listener receives a response to a neighbor request, it will do the same thing if the response was a confirmation of its neighbor request. This will also spawn a thread that does link cost messages to measure the link between the router and that neighbor. Another thread will spawn that sends alive messages on a timer to the neighbor. If 10 alive messages time out, the neighbor will be considered dead, or at least the link between them is dead, so the link will be removed and the neighbor will be removed from the list of neighbors. Other routers will then be told that this link is down.

The thread that is spawned to do link cost messages will be set on a timer to continuously send messages and record the time between when they were sent and when the ack was received. When it is time to update, it will average the message times and use that as the link cost for its own LSA. We decided to include the queueing time at the other router in this measurement. While this can be somewhat bursty, we felt it was a more accurate representation of the performance that was being seen, and it would be averaged among other messages anyway.

Upon receipt of an LSA, the listener will first check in the LSA hashmap to see if there is an LSA for this router ID. If there is no LSA, the listener will create an entry that maps to a pointer to a control struct (sends and acks, includes sequence numbers and age) for sending the LSA and the LSA itself. A thread will be spawned which will be responsible for sending the LSA to each of its neighbors and making sure it gets an ack. An age field will be in the LSA which will be continuously decremented and eventually when it hits 0, if a newer LSA has not arrived for this router ID, then this LSA will be discarded. If an LSA for this router ID is received, and its sequence number is newer than the one stored in the hashmap, then the LSA will be replaced and we will start sending out the new one. When a newer LSA is received, the information in the LSD will also be updated. When the listener receives an acknowledgement of an LSA, it will acquire the lock on that LSA's control struct and specify that the ack was received.

When the listener receives an alive message, it will immediately send a response. If it receives an ack of one of its own alive messages, it will update the control struct for the thread that is in charge of messaging that neighbor. These structs can be found through a hashmap that maps the router ID to the pointer to its control struct. If the listener receives a message about a link being down, it will immediately update the LSD and remove the link.

UI Component

The ui component, ui.c, interacts with the routing.c through the listening socket. The router component will receive messages asking for the routing table information, or to remove or add a neighbor. Upon the listener receiving these, it will process the request and return the results to the ui component. The ui component will be a simple repl interface.

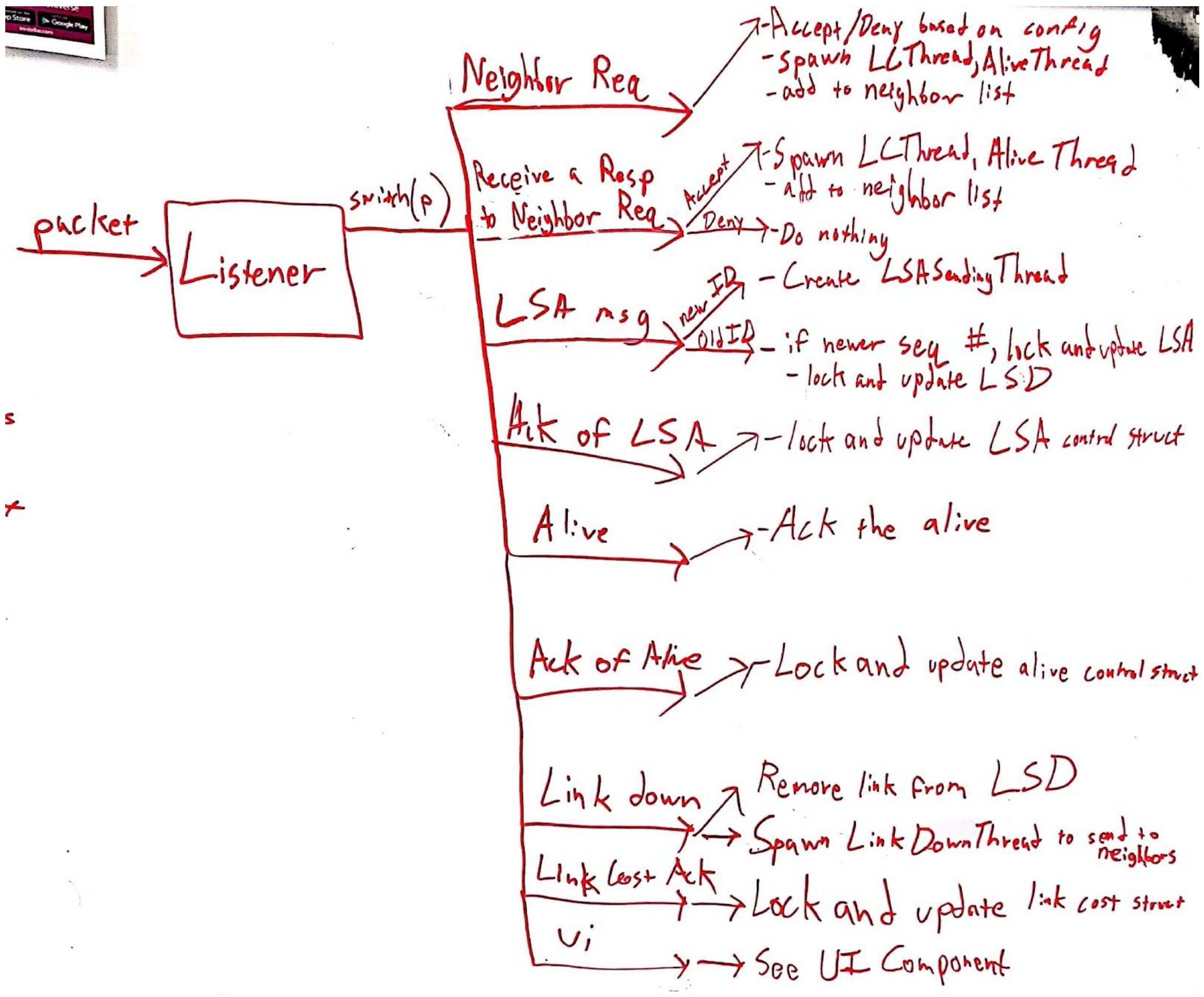
Routing Table/LSD

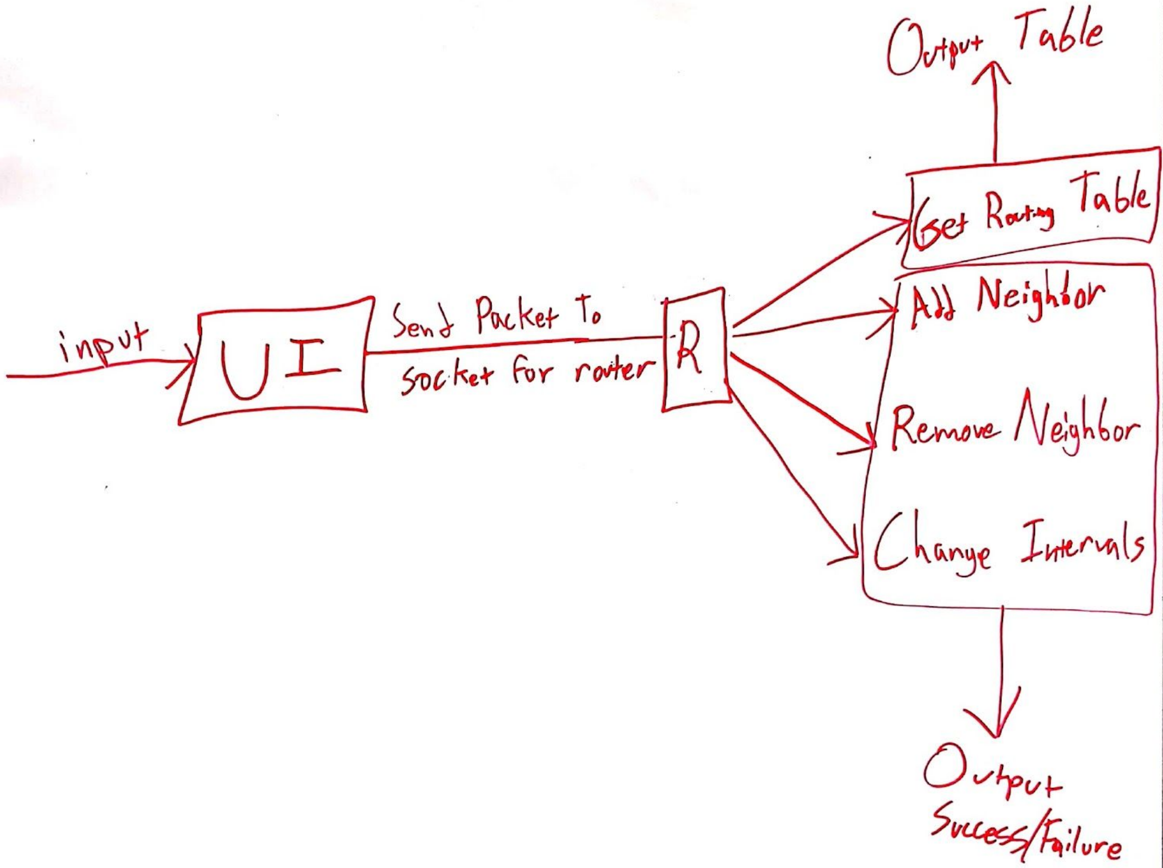
The routing component will have a routing table on hand to use. It will keep a separate routing table that is being built behind the scenes (in a separate thread spawned in the beginning) and the pointer will be swapped upon completion of the new table. The table will be rebuilt when the timer goes off signifying that the interval for the next update is now. It will build the table by acquiring a lock on the LSD (so new LSA don't get written). The LSD will be (edge, weight) pairs that will be used to build the data structures that dijkstras will need to build the routing table. The routing table itself will be built into a hashmap that maps the desired destination ID to the ID that the packet should be sent to.

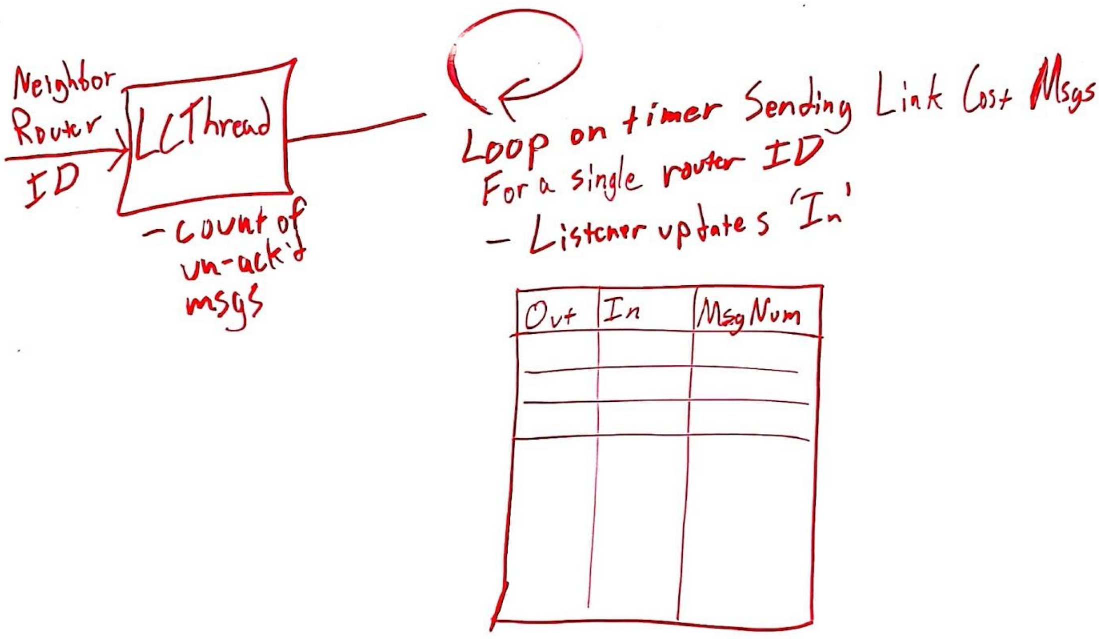
Diagrams (should be self explanatory):

Supplemental information:

- Alive thread should LinkCostThread if neighbor is declared dead
- LinkCostThread should not have a counter for un-acked messages (alive thread will take care of this and kill the link cost thread, as mentioned in the above point)
- LinkDownThread is responsible for propagating Link Down messages - so its behavior and diagram is exactly the same as LSASendingThread, except that it is dealing with link down messages rather than LSAs

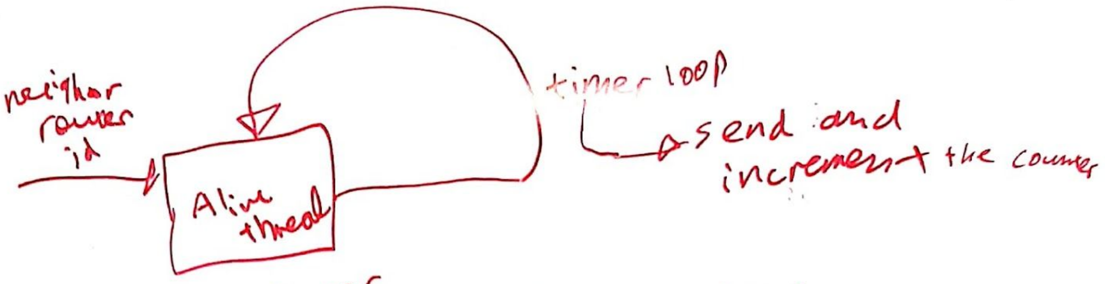

packet
Listener
switch(p)
Neighbor Req
-Accept/Deny based on config
-Spawn LCThread, AliveThread
-add to neighbor list
Receive a Resp to Neighbor Req
Accept
-Spawn LCThread, Alive Thread
-add to neighbor list
Deny
-Do nothing
LSA msg
new ID
-Create LSASendingThread
Old ID
-if newer seq #, lock and update LSA
-lock and update LSD
Ack of LSA
-lock and update LSA control struct
Alive
-Ack the alive
Ack of Alive
-Lock and update alive control struct
Link down
Remove link from LSD
Spawn LinkDownThread to send to neighbors
Link Cost Ack
Lock and update link cost struct
Ui
See UI Component


input
UI
Send Packet To
socket for router
R
Get Routing Table
Output Table
Add Neighbor
Remove Neighbor
Change Intervals
Output
Success/Failure



Loop on timer Sending Link Cost Msgs
For a single router ID
- Listener updates 'In'

Out	In	MsgNum

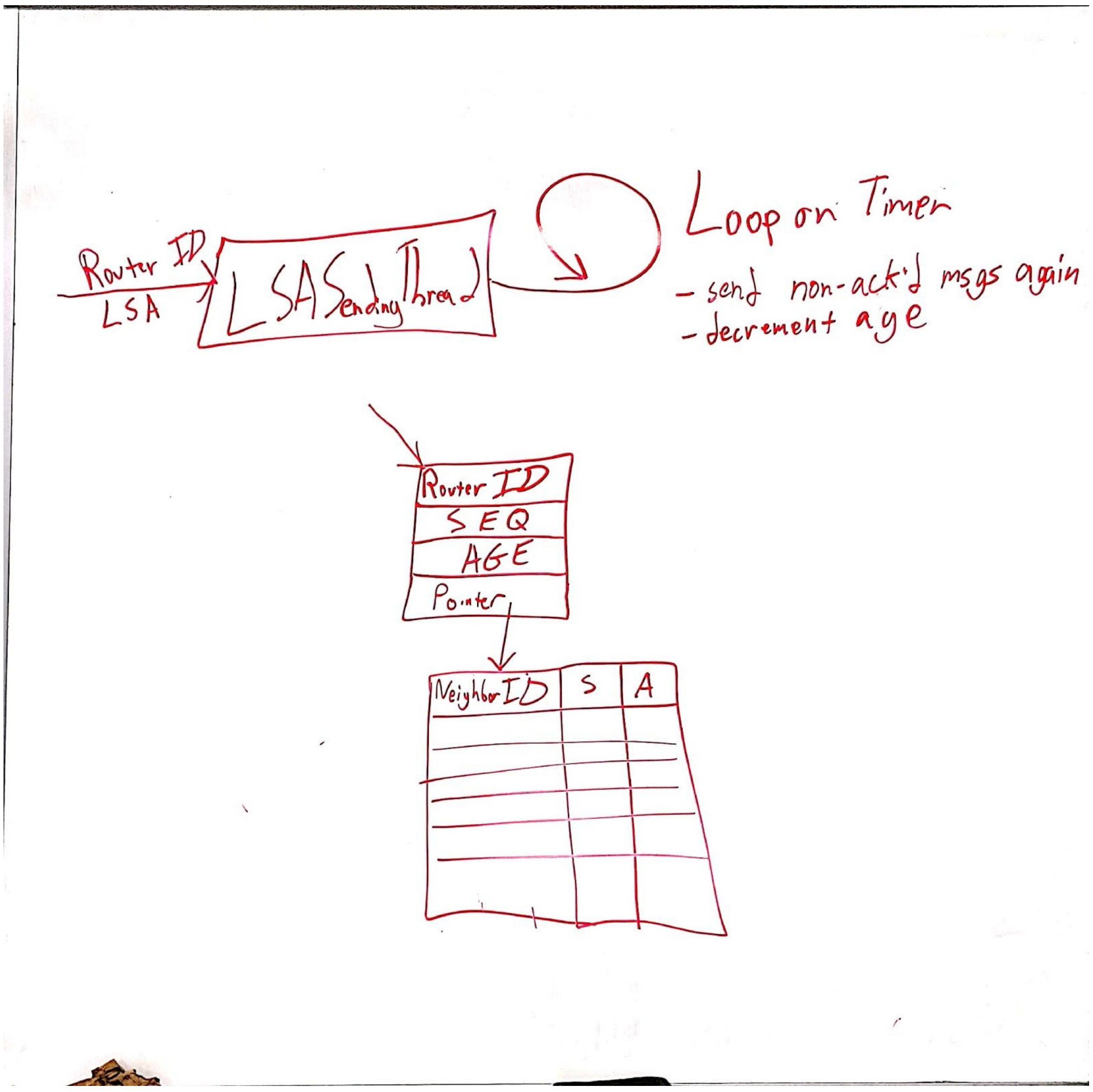


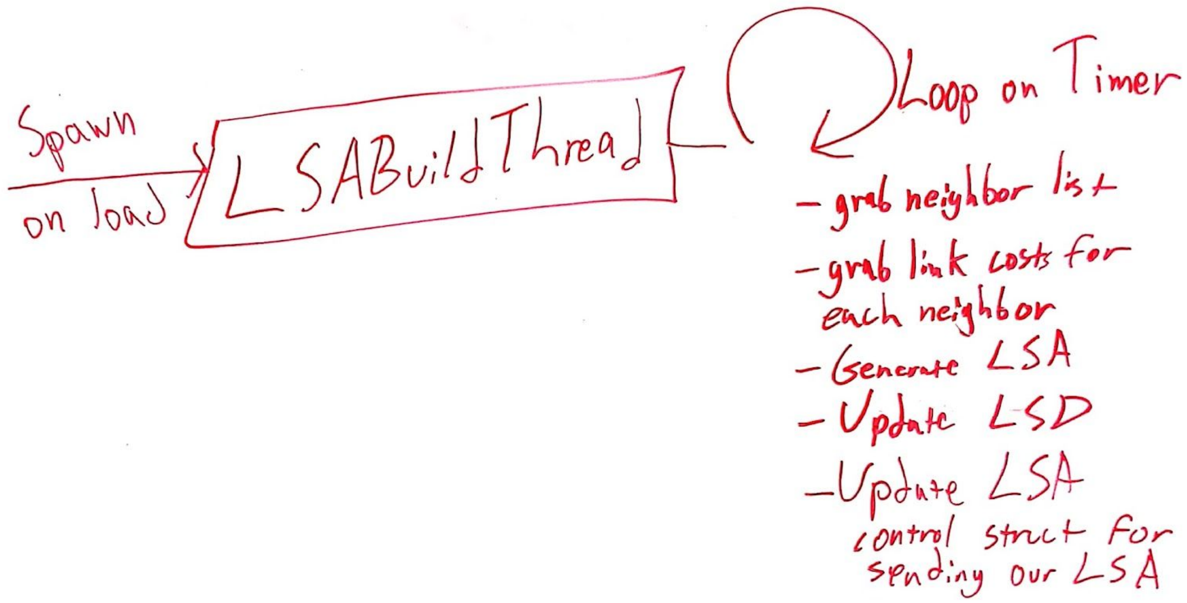
- counter
  ↳ # of unacked alives ← updated (set to 0) by listener upon alive ack

↓

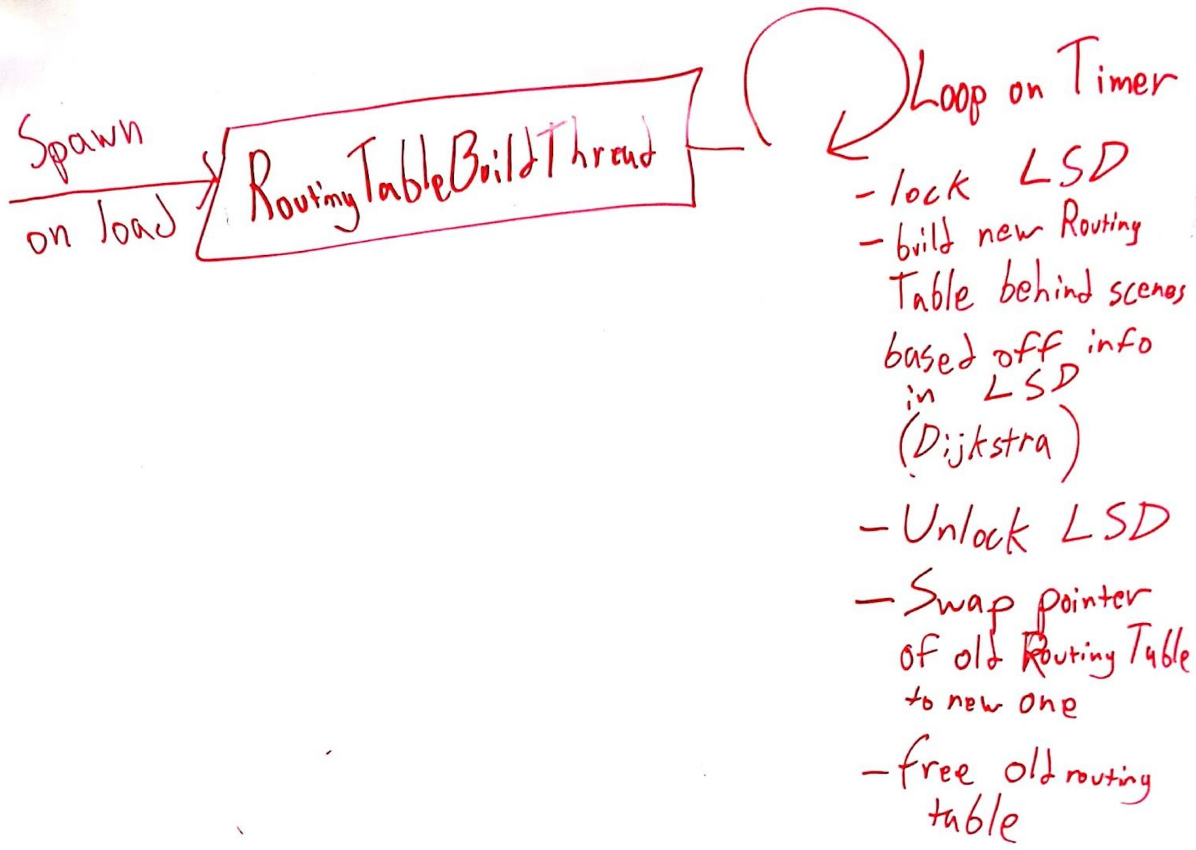
if > 10,

- delete neighbor from neighbor list, LSD
- spawn LinkDownThread
- die


Router ID
LSA
LSASendingThread
Loop on Timer
- send non-ack'd msgs again
- decrement age
Router ID
SEQ
AGE
Pointer
Neighbor ID
S
A



- grab neighbor list
- grab link costs for each neighbor
- Generate LSA
- Update LSD
- Update LSA control struct for sending our LSA



- lock LSD
- build new Routing Table behind scenes based off info in LSD (Dijkstra)
- Unlock LSD
- Swap pointer of old Routing Table to new one
- free old routing table