Kyle Shyffer, Zach English

For this project, the natural place to start was with a LinkState interface and module. LinkState would be relatively analogous to NeighborDiscovery: It was going to send out a special type of packet, and wait for its neighbors to send it packets of their own, before incorporating the data from those packets into a table that other parts of the program can draw from. So, in broad strokes, that module is based on the NeighborDiscovery one. However, LSR actually requires information to be communicated by the packets, rather than just their existence. So we have a struct for a LSA payload that gives all of that information, we’re using the special protocol, and we accounted for those factors in our IP module and Flooding, so the payload can be delivered to LinkState and the information can be updated.

The IP module is as simple as possible, which matches the philosophy of IP in the “real world.” It consists of two commands: starting a new message, and forwarding an existing one. In either case, it does little more than pack up the message properly, refer to LinkState for where it should be routed, and use the existing SimpleSend system to send it along the next hop. The primary benefit of defining an IP header is the ability to track both the ultimate source/destination as well as the immediate one, similarly to keeping track of both an IP address and a MAC address in an Ethernetwork.

In order to handle changes in topography, we have NeighborDiscovery re-running on a timer, and whenever there is a change in the list of neighbors, we signal LinkState to begin the routing process again(technically we are signaling Node.nc which is then commanding LinkState, but the result is the same). We track new neighbors being added trivially, and we track dropped neighbors with a simple timeout system, where we drop any neighbor who hasn’t responded to NeighborDiscovery messages in the past several cycles of discovery(the actual number of cycles is arbitrary and easily modified; as we write this it’s set to only two). This system does work, but it can take a significant amount of time to reconverge after a change occurs. It seems that this may just be a necessary evil of the system, as our implementation does not seem to be glaringly inefficient.

The final item of note is the LinkStateDatabase within the routing module: it keeps track of the most recent LinkState message received from each node, and therefore makes it possible to see how the routing tables were constructed. This uses a sequenceNumber system not dissimilar to the one we use in Flooding.