**CSE160 Project 2 – Link State Routing: Questions**

1. We use shortest path configuration only when a link exists in both directions, because then the shortest path *to* the destination isn’t necessarily the shortest path *from* the destination, and because of this we can have a situation where packets go *to* the destination and they’re cut off from coming back because the shortest path on the way back has a dropped node. However, the source node will not learn about this dropped node because the information cannot get back to the source from the destination and the algorithm, therefore, cannot find a new shortest path correctly.
2. Our algorithm produces symmetrical results due to the nature of the algorithm and our network topography. Aside from topography, the shortest path from A to B should be only one route (unless we find an equivalent route and we use that as backup in case of a dropped node). Due to the nature of the previous question (#1), our network must have links that go both ways in order to keep our algorithm in working order and insure stability and reliability.
3. If you never forwarded after sending out LSPs then you’re doing the process a disservice; the goal of flooding LSPs is so that you can reach certain nodes without flooding the whole network again; so you can simply forward your packet using a forwarding table. In order to implement that, you must have each node, after receiving neighbors from every other node, add its info to a universal page table (via Dijkstra’s algorithm) that can be accessed in the future to see where the next intermediate hop is in forwarding packets.
4. If LSPs are corrupted or lost, the page table (forwarding table) may not include the absolute *shortest* routes and future forwarding may not the most efficient.
5. If a node kept advertising and withdrawing a neighbor, Dijkstra’s algorithm would continuously be repeated and ran while the packets flood through the network. This would cause congestion and would make the shortest path algorithm never approach a result asymptotically. In order to alleviate this, we would have to trap a condition where if a node sends out a new LSP that’s includes the same “delta” as the last withdrawing LSP, we kill this packet until a new LSP wants to be advertised/withdrawn with a new delta.