This project implements a flooding protocol and neighbor discovery system for TinyOS to enable multi-hop communication in both linear and mesh network topologies. The design utilizes split horizon, duplicate suppression, and modular separation of flooding and neighbor discovery. Control (beacon) and data traffic are handled on separate channels, preventing interference.

Neighbor discovery operates through a beacon-based protocol using dedicated control packets with protocol=0 and broadcast destination. Each node periodically transmits beacons with TTL=1 on the AM_PACK channel, ensuring these discovery packets can only reach directly adjacent nodes. When a node receives a beacon, it extracts the source address and adds or updates that neighbor in its local neighbor table, recording the current timestamp for aging purposes. The neighbor table maintains up to 32 entries, each containing the neighbor's node ID, last-seen timestamp, and validity flag. The system uses aggressive aging with a 10-second timeout. Any neighbor not heard from within this window is marked invalid and removed from the table. This rapid aging prevents stale entries from persisting when nodes move or fail. A cached neighbor list is maintained for fast access during flooding operations, updated whenever the neighbor table changes.

The flooding algorithm sends packets to all its neighbors simultaneously. When a ping command is issued, the source node creates a packet with TTL=30 and broadcasts it to every neighbor in its table. Each intermediate node that receives this packet performs several critical operations: duplicate detection using a (source, sequence, protocol) cache, TTL validation and decrement, and forwarding to all neighbors except the immediate sender. The algorithm employs split horizon to identify the previous hop, preventing packets from being sent back to the node they just came from. This eliminates immediate loops while still allowing packets to explore multiple paths through the network. When forwarding, each node creates packet copies for every valid neighbor (excluding the sender), decrements the TTL, and queues them for transmission through SimpleSend. Packet delivery occurs when the destination address matches the current node's ID, triggering a DELIVERY event and stopping further forwarding. For ping packets, the destination node generates a reply packet with the source and destination addresses swapped, then initiates its own flooding process to return the response. This bidirectional flooding ensures reliable round-trip communication even in complex topologies where the return path may differ from the forward path.

Several critical issues emerged during development. The most significant challenge was false neighbor discovery, where nodes incorrectly identified distant nodes as neighbors. Initially, all received packets triggered neighbor discovery, causing Node 9's ping packets to be received by every node in the network, leading each node to erroneously add Node 9 to their neighbor table. This created invalid network topology maps where distant nodes appeared directly connected. The fix restricted neighbor discovery to only beacon packets, while flooding packets were processed separately for data forwarding. Another issue was asymmetric flooding behavior, where packets could successfully travel forward through the topology (1>9) but failed when traveling backwards (9>1). This happened during ping testing, where Node 9 could receive pings from Node 1, but its replies never reached Node 1, creating one-way communication. The issue was that relay nodes were only attempting single-neighbor forwarding. One more issue was that packets would go back the way they came, resulting in unnecessary bouncing, which would also clog up the channel. This was solved with the split-horizon algorithm, which prohibited nodes from sending packets to the neighbor from which they arrived.