# Distributed Systems Report

**Communication** Nodes communicate by entering a message onto their outgoing messages queue. A message is a pair of the addressee and the payload, the string. The payload is usually three things; the type of message, e.g. ELECT, FAILURE_CHECK, the argument — e.g the node that is being elected as leader — and the final destination. The importance of the final destination will be clear later.

Node delivery is as suggested; nodes add outgoing messages to a queue. Once all nodes have ran, the network takes all of the messages and delivers them to their destinations, by putting them in a list of incoming messages for each node. If a node has two messages addressed to the same node, they may only send one per round.

One particular choice is that nodes may communicate with the network itself. We model the network as running on its own thread. The nodes send messages as standard but with address -1. The network intercepts and processes these messages, rather than routing them to their destination as is normal. This is done so that the network can handle events such as a leader being elected, which it logs, and a node failing.

**Leader Election** This follows the Chang-Roberts algorithm, without anything interesting or notable happening. When a node receives an ELECT message with its own id as parameter, it declares itself leader. At this point it sends a LEADER message to the next node. Once it receives a LEADER message with its own id, it messages the network to say that it has successfully been declared leader.

**Failure Detection** Every T rounds since the last successful failure check, the nodes start a new check. This consists of sending a `FAILURE_CHECK` message to the next node and awaiting a response. Whenever a node receives a FAILURE_CHECK message, it returns a FAILURE_RESPONSE message. The nodes wait 2D rounds for a response; if a response is not received in this time then the node is marked as failed. Our constants T and D were set to 12 and 5 respectively.

**Failure Recovery** When a node fails, a message is sent to the network. Upon receipt, the network has to try and maintain the ring structure of the network itself. The way it does this is by checking which node had its next pointer set to the failing node. It then sets the next pointer of this node to the next pointer of the node that failed.

However, we still need to obey the rules of the network; there may not necessarily be a connection between these two nodes. Hence the network performs a breadth-first search from the former node to try and find the latter. If this search fails, the network has become disconnected as a result of failure; if this is the case we log that this has happened and end the simulation. If this is not the case, we now need to forward messages from the latter node to the destination.

**Redirection** Each node has a map, from integers to integers, that stores where it should redirect a message upon receipt of a message not delivered to itself. When a path is found between two nodes, the network updates this map for each of the nodes with the redirection information. Then, finally, when a node receives a message not addressed to itself it forwards it on to the corresponding address in the map. In this manner, any “holes” in the network caused by failures get patched up.