1. **Introduction**

For this assignment we will build a basic distributed system model in a ring topology with 8 nodes and demonstrate a Leader Election protocol. Assuming unique IDs, the leader is the node with the smallest ID. All nodes start as ASLEEP, and end up either a FOLLOWER or a LEADER. In real-life, leader election can be used by nodes to self-determine who has the least ping, or the fastest processor; all without the need for a centralized server (i.e. a bottleneck) to compare them and make that decision.

Our simulator will help us test the AsFar protocol [1], where every ID is passed along around the ring, and only stopped when it comes across a Node with a lower (i.e. better) value. When a node receives its own ID back (i.e. it survived around the entire ring), it now knows that it’s the LEADER and we output its ID.

1. **System Modifications**

We initially had this configuration in mind (Figure 1), which included a queue and simulated network delays in each computing node. We however ran into several difficulties trying to instantiate the coupled Node model into 8 or more instances. We are still unaware of a non-manual way to instance coupled models and it has led to several human errors and hard-to-find bugs when designing the full the network of 6 to 12 nodes.

**L**

**R**

**Distributed System**

Computing Node

Network Delay

Queue

Network Delay

Message Processor

**L**

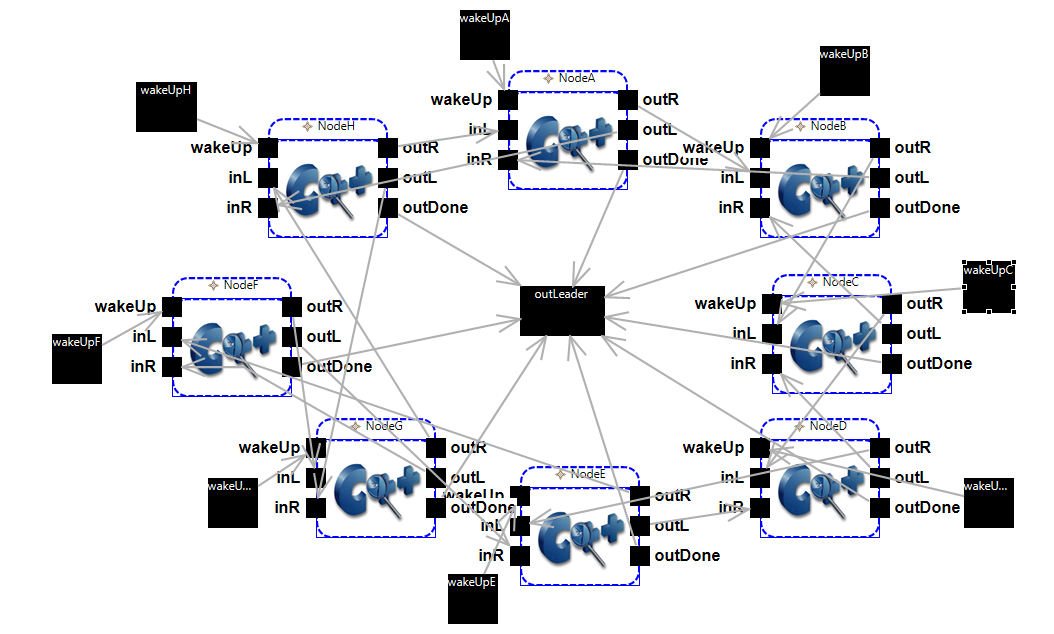
**R**

**Figure 1**. Initial design of a Ring Distributed Computing System.

For this reason we opted for a simpler system that might lose us some marks, in the interest of handing this assignment on time.

1. **Model Definitions**

The system consists of a network of 8 *RingNodes* arranged in a bidirectional ring topology. Each node has left and right input/output ports to indicate the direction along the ring. Left is equivalent to counterclockwise, and Right is clockwise.

 **Figure 2.** RingNetwork coupled model diagram.

**3.1 RingNode atomic model**

Behaviour

* Waking up: every node starts ASLEEP, and can be made AWAKE by any value input through the *wakeUp* port. Once AWAKE, the node will send its id through the *outR* port (start clockwise by default), and store its id in the ‘min’ variable. Any further input from *wakeUp* will be ignored.  
  The node can also be awakened by receiving a value through *inL* or *inR* ports. In this case, we first compare between the received id and the node’s own id, and only send the smaller one to the opposite port (receive from right🡪send to left, and vice versa). This value is now stored in ‘min’.
* While AWAKE, the node will receive messages from its neighbours. For each incoming message *x*, it compares it with ‘min’.
  + If *x* is lower, then ‘min’ is set to *x* and the node now knows it’s a FOLLOWER. It passes the value along to the opposite port ((receive from right🡪send to left, and vice versa). As a FOLLOWER, the node behaves exactly the same as AWAKE, except it now has this knowledge of being a follower.
  + If *x* is higher than or equal to ‘min’, the message is ignored (no output) and ‘min’ remains the same.
  + If *x* is equal to ‘id’, then I’ve received my own id back, so the node becomes a LEADER and outputs its ‘id’ through the *outdone* port. Once a LEADER, all future messages are ignored.

Specification  
State Variables:

* id // Node’s ID. Defined as a parameter in the parent coupled model definition.
* min = id // Initialize current known minimum value as my ID.
* srcPort = {NONE, WAKEUP, LEFT,RIGHT,INTERNAL}   
   //records the source of the current transition. Starts as NONE.
* nodeStatus = {ASLEEP, AWAKE, FOLLOWER, LEADER}  
   //current status of the node. All nodes start ASLEEP.

**S** = {id, min, srcPort {NONE,WAKEUP,LEFT,RIGHT,INTERNAL}, nodeStatus {ASLEEP, AWAKE, FOLLOWER, LEADER} }

**X** = {wakeUp, inL, inR}

**Y** = {outL, outR, outDone}

**δext** (id, min, srcPort, nodeStatus, x) **{**

if(port == wakeUp){

if (nodeStatus == ASLEEP){

nodeStatus = AWAKE;

srcPort = WAKEUP;

holdIn( active, Time(0,0,1,0) ) ;

}  
// otherwise, ignore messages, I’m already awake!

}else{

if(nodeStatus != LEADER){ // if I'm still a follower or candidate

if (x==id){ // I received my own id; I now know I'm the leader.

min = id;

nodeStatus = LEADER;

holdIn(active, Time(0,0,1,0)); // Send output in 1 second

}

else if (x<min){

min = x;

nodeStatus = FOLLOWER; //I've received a smaller id; I now know // that I'm a follower.

srcPort = NONE;

if(port== inL ) srcPort = LEFT; // record source port to decide out port

else if(port() == inR) srcPort = RIGHT;

holdIn(active, Time(0,0,1,0)); //Output the new ‘min’ in 1 second

}

else if(x>min){

nodeStatus = AWAKE; // just in case it was still ASLEEP

min = id;

if(msg.port() == inL ) srcPort = LEFT;

else if(msg.port() == inR) srcPort = RIGHT;

holdIn(active, Time(0,0,1,0)); //Output the ‘min’ in 1 second

}

} } }

λ(nodeStatus, srcPort){

if(nodeStatus == LEADER)

Send ‘id’ to port *outDone*

else{

if (srcPort == WAKEUP) // Initiate by sending to the right

Send ‘id’ to port *outR*

else if(srcPort == LEFT) // If received from left; pass on to right

Send ‘min’ to *outR*

else if(srcPort == RIGHT) // If received from right; pass on to left

Send ‘min’ to *outL*

}  
}

inL

outR

**3.2 RingNetwork coupled model**

outL

inR

Behaviour

Figure 3. Connection between 2 neighbours in the ring.

* The RingNetwork consists of 8 RingNodes, all initially ASLEEP.
* Each node is given a unique ‘id’ as a parameter in the MA definition. An exception will be thrown if the ‘id’ parameter is missing.
* Increasing the number of nodes is simple; just carefully connect the nodes as shown in Figure 3.
* The Leader Election algorithm, AsFar, assumes that we don’t know how many nodes are AWAKE at the beginning of the simulation. The AsFar protocol is designed to work regardless of how many are initially awake and sending messages. Even if only 1 node is initially awake, the rest will eventually wake up and we are guaranteed a correct Leader election output. To control this initialization, we have 8 *wakeUp* input ports, one per RingNode. Waking up events can be specified in the EV file.
* The network has an output port *outLeader* which will spit out the leader’s ‘id’ once the simulation is finished

Specification

**X** = {wakeUpA, wakeUpB, wakeUpC, wakeUpD, wakeUpE, wakeUpF, wakeUpG, wakeUpH}

Y = {outLeader}

D={NodeA, NodeB, NodeC, NodeD, NodeE, NodeF, NodeG, NodeH}

I(NodeA) = {self, NodeH, NodeB}; Z(NodeA) = self; Z(NodeA)=NodeH; Z(NodeA)=NodeB  
 I(NodeB) = {self, NodeA, NodeC}; Z(NodeB) = self; Z(NodeB)=NodeA; Z(NodeB)=NodeCI(NodeC) = {self, NodeB, NodeD}; Z(NodeC) = self; Z(NodeC)=NodeB; Z(NodeC)=NodeD  
 I(NodeD) = {self, NodeC, NodeE}; Z(NodeD) = self; Z(NodeD)=NodeC; Z(NodeD)=NodeE

I(NodeE) = {self, NodeD, NodeF}; Z(NodeE) = self; Z(NodeE)=NodeD; Z(NodeE)=NodeF  
I(NodeF) = {self, NodeE, NodeG}; Z(NodeF) = self; Z(NodeF)=NodeE; Z(NodeF)=NodeG  
I(NodeG) = {self, NodeF, NodeH}; Z(NodeG) = self; Z(NodeG)=NodeF; Z(NodeG)=NodeH  
I(NodeH) = {self, NodeG, NodeA}; Z(NodeH) = self; Z(NodeH)=NodeG; Z(NodeH)=NodeA

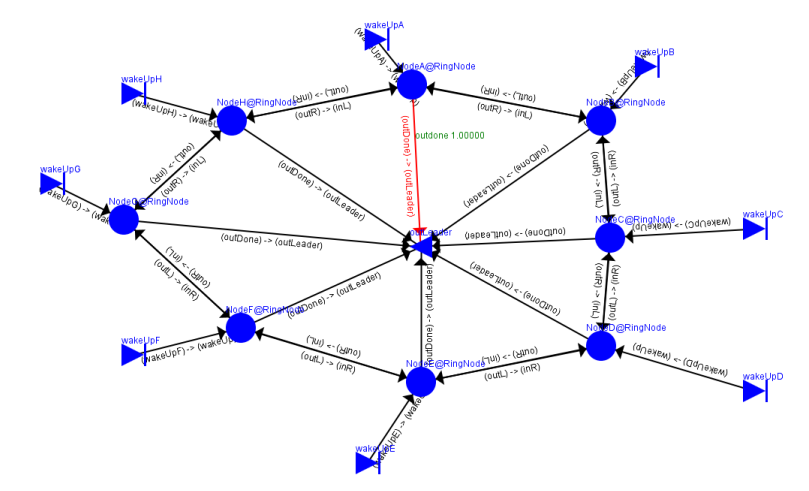
SELECT: ({NodeA, NodeB})=NodeA; ({NodeB, NodeC})=NodeB; ({NodeC, NodeD})=NodeC;  
 ({NodeD, NodeE})=NodeD; ({NodeE, NodeF})=NodeE; ({NodeF, NodeG})=NodeF;  
 ({NodeG, NodeH})=NodeG; ({NodeH, NodeA})=NodeH;

**4. Testing**

The atomic model RingNode can be tested with any event sequence in the RingNode.EV file. The key ideas are:

1. Consistently output decreasing (i.e. better) or equal values. Never output a higher value as time goes by.
2. The node can be awakened by any input port. The *wakeUp* port is ignored once the node is no longer ASLEEP.
3. Input from the inL results in output to outR (and vice versa); if the value is better than ‘min’.
4. Once received a lower value than your ‘id’, become a FOLLOWER
5. Once received your own ‘id’ and haven’t seen something smaller, then: become a LEADER, send ‘id’ to outPort, and ignore all future messages.

The network can be tested by “waking up” various number of nodes at various times in RingNetwork.EV file.

1. If no node is awakened, nothing happens.
2. A node can be awoken by any value passed through one of the eight *wakeUp* ports.
3. If any node is awaken, we are guaranteed an output through *outLeader* port, regardless of how many nodes are defined in RingNetwork.ma
4. id values can be adjusted in RingNetwork.ma to test various configurations.

The animation can neatly show the progress of the algorithm, and how it’s guaranteed to reach the *outLeader* port every time. Use a timestep of 500 to best see the results of various EV file configurations and id’s:

**References**

[1] Nicola Santoro, “*Design and Analysis of Distributed Algorithms*”. Wiley-Interscience 2006.