SELF ORGANIZED WIRELESS SENSOR NETWORK

EE 662

San Diego State University

Department of Electrical and Computer Engineering

Lana Pantskalashvili

Fall 2022

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Introduction

Data collection is an important application of wireless sensor networks (WSNs) and Internet of Things (IoT). Wireless networks provide a lot of significance to communication systems due to the nodes' ability to choose their location and recover from failures.

The initial version of the data collection tree is organized into clusters in a multi-hop architecture as shown in Figure 1.

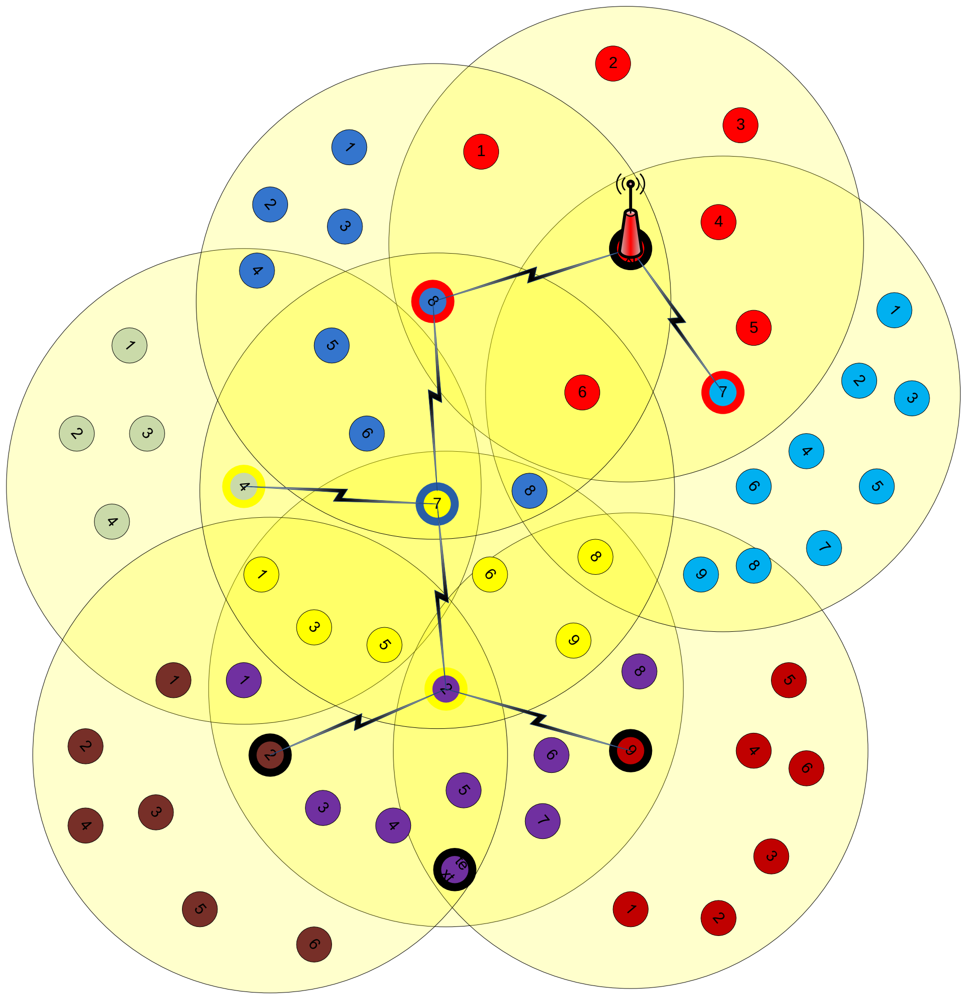


Figure 1: The Initial Data Collection Tree Topology

A self-organizing cluster tree topology is designed and implemented. Among the many functionalities, collision avoidance by reducing the cluster overlapping area and an overlay multicast protocol play key roles.

Network Topology Construction: Initial Stages

When a sensor node is powered on it will start listening to the channel. If there are more than one possible channel it will listen on all available channels to determine whether a network exists or not. Depending on the node and the availability of a network the role and behavior of the network will be determined. The activity of a node arriving to a channel is given in Figure 2.

As shown in the Specification and Description Language diagram in Figure 2, the node arriving to the channel will set a trial counter of N to zero and start a timer (Timer A). It will them start listening to the channel for network activity. There are two potential events that will activate the node. The Timer A may expire indicating no network activity during the listening period. In this case, the number of listening attempts (N) will be compared to a preestablished threshold and if the number of attempts is less than the threshold, Timer A will be restarted, and number of attempts N will be incremented by one. If the number of attempts N is larger than the threshold, the action of the node depends on the node abilities. If this node is rootEligible than the node will claim the role of the root node. If the node is not root eligible than it will set a timer B (which will presumably be longer than Timer A) and will go into a sleep mode. The node will come out of sleep mode when time B expires and will rerun the node search model we explained here.

If a packet arrives during the channel listening, the node will check if this is a heartbeat packet and decide next action. A heartbeat packet will be transmitted by root node and cluster heads to indicate availability of the network and the willingness of accepting new nodes into the network. If the packet is a heartbeat packet, the node will initiate the network join process, otherwise the packet will be ignored as shown in Figure 2.

The node that becomes root will also assume the role of the cluster head for network 1.

The protocol defined in Figure 2, specifies two timers and two packets: Heartbeat message and Probe packet. The probe packet is intended to force response from ClusterHead and the response to the probe will in this case be a heartbeat message. All messages transmitted on this network will be transmitted using a common header:

NextHop:	2 bytes
DestinationAddress:	2 bytes
SourceAddress:	2 bytes
PacketType:	1 byte
Payload:	N bytes

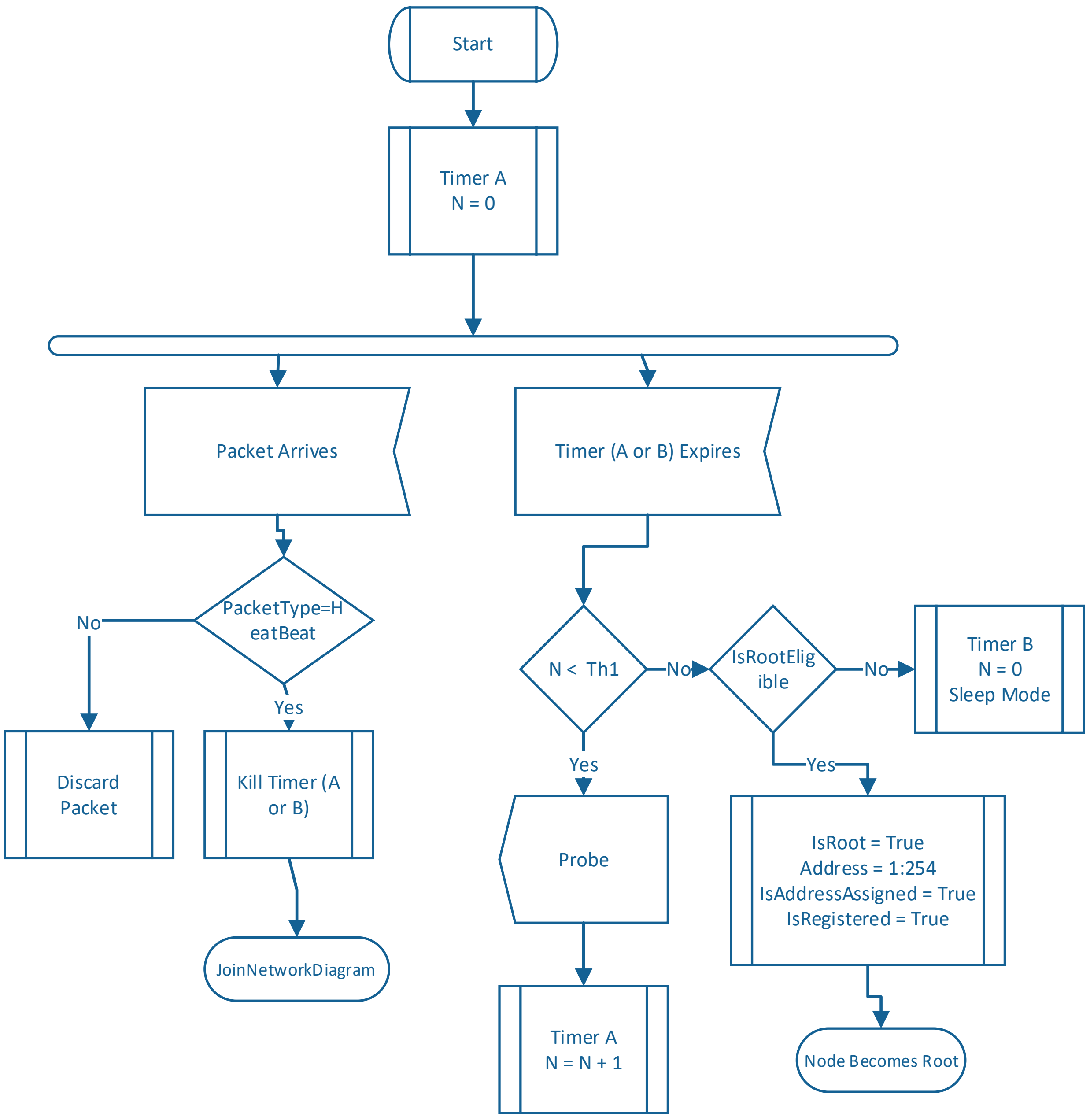


Figure 2: SDL of the Node Arriving to the Channel

Root node serves as the controller for the entire network. The Root node is responsible for the management of the entire network, creation of new clusters and maintaining network topology. It is also responsible for acting as a gateway between the data collection tree and external networks such as IP networks. Root node will have more than one network interface to be able to act as a router between data collection tree and external networks.

The root node is also the ClusterHead for network 1 which is the root network. Each network in this topology is managed by a ClusterHead which serves as the router for the nodes in its cluster. In addition, the ClusterHead is the manager for its own cluster. ClusterHeads form a backbone network such as the one given in Figure 3.

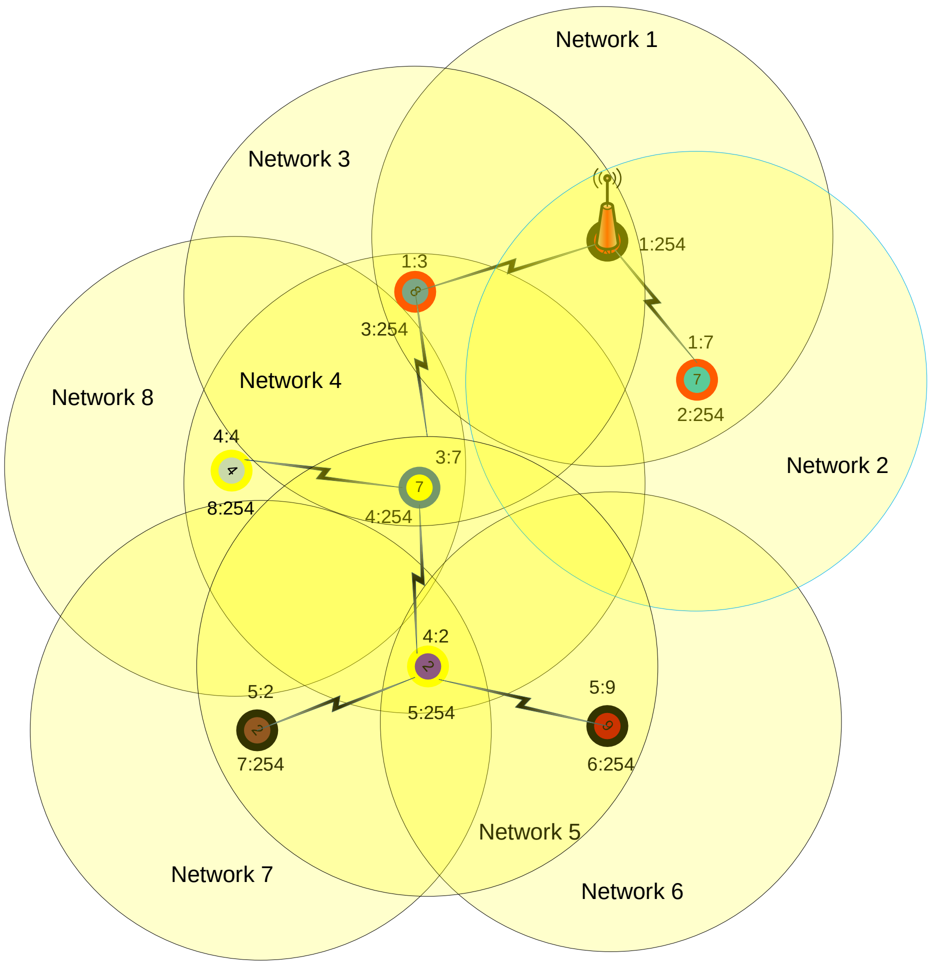


Figure 3: Backbone Network Formed by ClusterHeads in a Data Collection Network

The ClusterHead is responsible for admission of new nodes into the network and assigning local network addresses. ClusterHeads also serve as routers between the nodes in its network and the root as well as other clusters.

Router is a functionality that will be performed by ClusterHeads and other designated roles. Routers repeat the packets they received towards their destination.

Nodes are sensor or actuator nodes that do not have additional network functions.

There are six roles in the network:

1- Root
2- ClusterHead
3- Router
4- Registered Node
5- Unregistered Node
6- Undiscovered Node

Additionally, based on the capabilities of the node (battery, transceiver, processor, or policy related) each node may have several attributes as defined below. Some of these attributes will be assigned at the time the node is produced. The attributes such as "is root eligible" and "is router eligible" will be used during the network setup and runtime to decide the role of a network. These attributes will take a true/false value and will not be changed during the runtime.

Node attributes are the following:

- IsRouter?
- IsRegistered?
- IsRoot?
- IsGateway?
- IsRouterEligible?
- IsRootEligible?
- IsaddressAssigned?

The attributes such as "Is Router", "Is Root" are assigned at run time and will identify the role of the node in the network. Initially all nodes will be assigned to the following attribute values (the run-time initialized attributes).

- IsRouter = False
- IsRegistered = False
- IsRoot = False
- IsGateway = False

- IsaddressAssigned = False

Each node will have a globally unique ID (GUID) assigned to the node at the time of production. The GUID be used for identification of individual nodes.

For routing and communication purposes the network will utilize dynamically assigned addresses. The addresses will be two-byte short addresses and will be composed of two parts (NetworkID, NodeID). Network ID will identify the cluster to which nodes belong to and NodeID will identify nodes within a cluster.

The following addresses are predefined:

- 2 bytes (Network ID + Node ID) (8 bits each)
- 0.X - reserved
- 254.X – multicast
- X.254 - cluster heads
- 255.255 - broadcast
- X.255 - local broadcast
- 255.X – reserved

After becoming a ClusterHead, the node will start transmitting HeartBeat messages in response to received probe messages. The format of the Probe message is shown in Table 1.

2 bytes	**1 byte**	**2 bytes**
Source Address	Packet Type: Probe	Destination Address

Table 1: Probe Message Format

In response to probe messages, the ClusterHead will send heartbeat message which is described in Table 2.

1 byte	**2 bytes**	**1 byte**	**2 bytes**
No. Hops to Gateway	Source Address	Packet Type: Heartbeat	Destination Address

Role (state)	Originator Address (self)		ClusterHead Address
GUID			

Table 2: HeartBeat Message Format

The sequence diagram demonstrating the interaction between the new node and the ClusterHead is shown in Figure 4. As described, an unregistered node will send a Probe message in order to be recognized. The ClusterHead, upon receiving a Probe message will reply with a HeartBeat. Once an unregistered node hears a HeartBeat, it sends to Join Request message to join a cluster. ClusterHead replies by assigning a node ID which means that the unregistered node can become a registered node and join the cluster. Finally, the node sends an Address Acknowledgment message to acknowledge becoming part of the network.

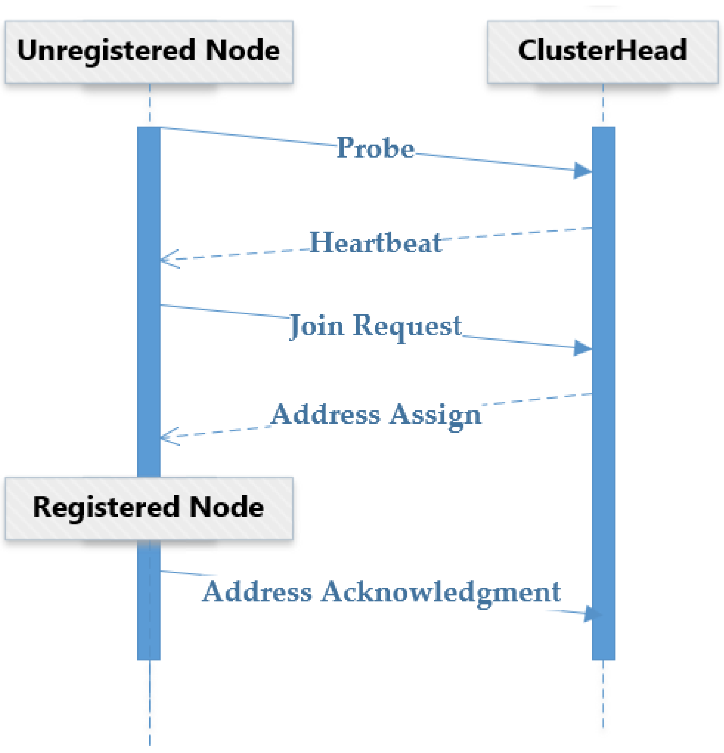


Figure 4: Sequence Diagram for the Node and ClusterHead Interaction

As shown in Figure 4, after receiving the heartbeat message, the node may send a JoinRequest Message. It is possible for the node to collect several heartbeat messages from the ClusterHeads and then choose its network. The new node sends a JoinRequest message, and its format is described in Table 3.

6 bytes	1 byte	2 bytes
GUID	Packet Type: JoinRequest	Destination Address

Table 3: Join Request Message Format

Upon successfully receiving the Join Request Message, the ClusterHead will send an Address Assign Message that acknowledges join request and assigns address for the new node. The Address Assign Message format is shown in Table 4.

6 bytes	2 bytes	1 byte	2 bytes
GUID	Source Address	Packet Type: AddressAssign	Destination Address
Destination GUID	Root Address	No. Hops to Gateway	Address (to be assigned to a registered node)

Table 4: Address Assign Message Format

Only the RootNode and ClusterHead are eligible to assign network addresses. A LeafNode, or a registered node may only broadcast a heartbeat message. Now the registered node will send an Address Acknowledge Message shown in Table 5.

6 bytes	2 bytes	1 byte	2 bytes
GUID	Source Address	Packet Type: AddressAcknowledgment	Destination Address

Table 5: Address Acknowledgment Message Format

The whole process is shown on SDL activity diagrams in Figure 5 for the ClusterHead, and Figure 6 for the new node.

The ClusterHead will start listening to the channel. Depending on the received packets and timer duration, the behavior of the network will be different. The activity of the ClusterHead when a new node is joining its cluster is given in Figure 5.

As shown in the SDL diagram in Figure 5, the ClusterHead may receive several types of packets. If the received packet is Probe, the ClusterHead in return, will send a HeartBeat message. If the received packet is Join Request, the ClusterHead will respond by transmitting a Join Reply message to assign a Node ID so that the node will be able to become to change its state to registered and will become a part of the network. If packet type is Address Acknowledgment from the newly registered node, this node will be added to the member's table. Member's table simply consists of joined, registered nodes.

The protocol defined in Figure 5, specifies four packets: Probe, Heartbeat message and Join Request, Address Acknowledgment packet. All of them were described using Tables above.

According to Figure 6 which demonstrates SDL of the unregisteed node, upon starting the Probe timer, the node might receive several types of packets. The response of the node is different for all of them. The unregistered node may receive a HeartBeat message from the ClusterHead, which means that there is a ClusterHead in this node's transmission range, and it can join its network. In response to the HeartBeat message, this node should send a Join Request message to the HeartBeat packet source. Another type of message that an unregistered node can receive is Join Reply, or Address Assign message, which holds the new Node ID for the potential registered node. If the unregistered node is satisfied with the Join Reply message it will send an Address Acknowledgement packet to finalize the joining process and acknowledge its new network.

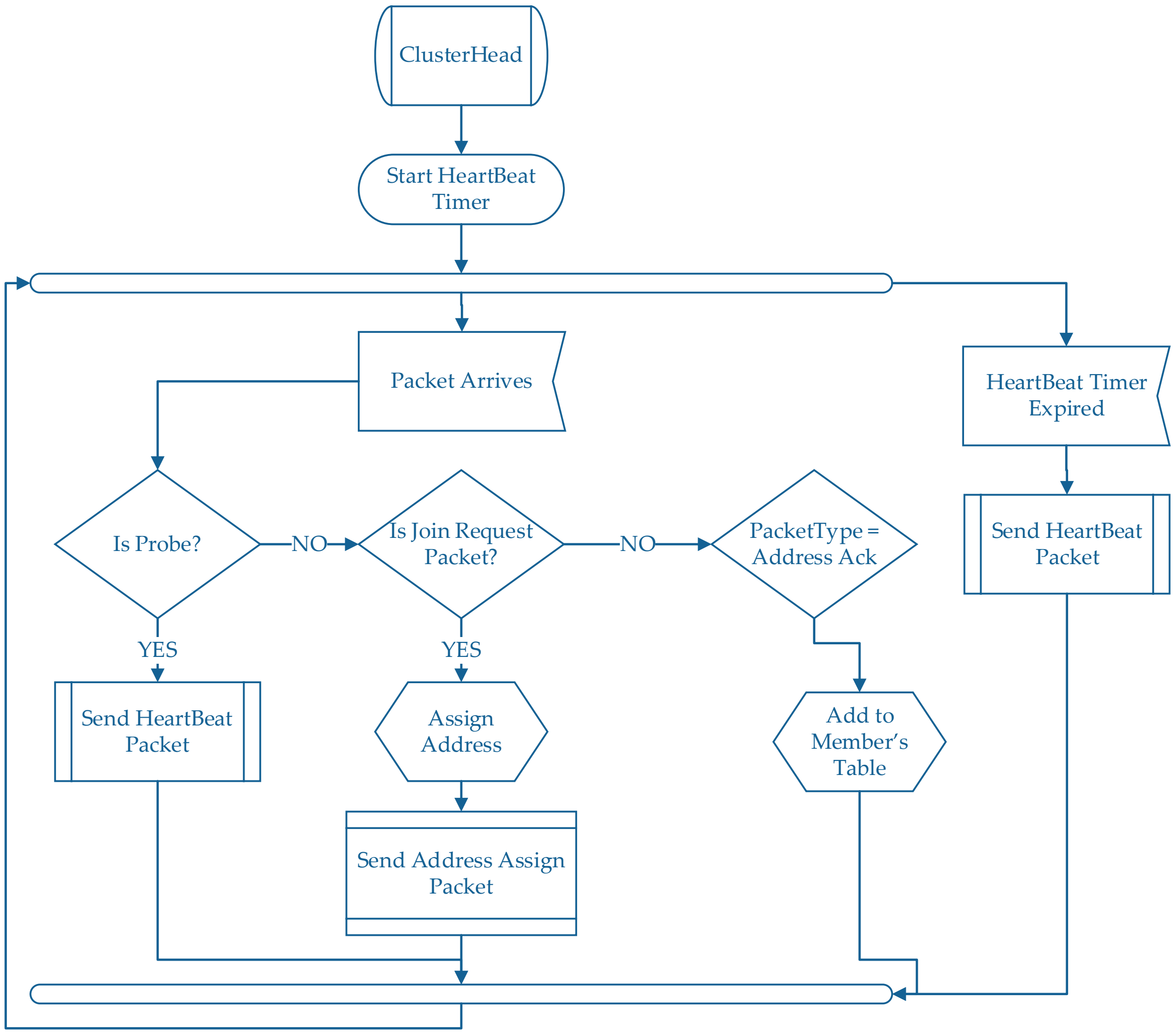


Figure 5: SDL of the ClusterHead, Joining a Cluster

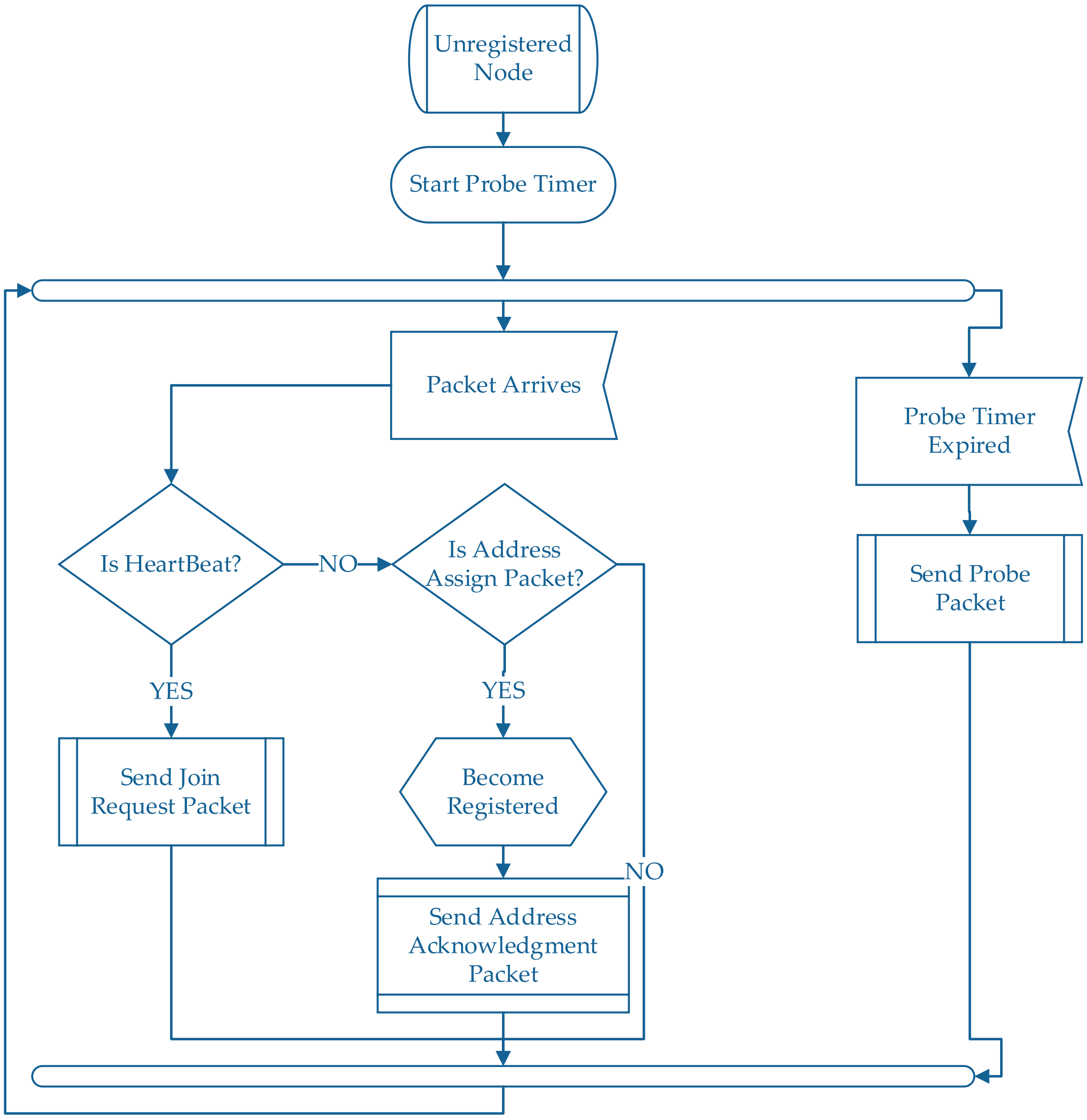


Figure 6: SDL of the Unregistered Node, Joining a Cluster

Routing

Now that we have discussed the initial network setup, we can analyze the routing process for this network since it is the most important part of the network. For this reason, we are using routing tables, such as the one given in Table 6 to shorten processing time and extend the lifetime of the system.

Child Node N	Address	Networks under child

Table 6: Routing Table Format

The first column of the table contains the internal child node ID's, the second column represents addresses of the child nodes. The third column is the array of the networks under a certain child ID. Figure 7 demonstrates the example on how the table gets populated. It is important to note that routing is done using dynamically assigned addresses.

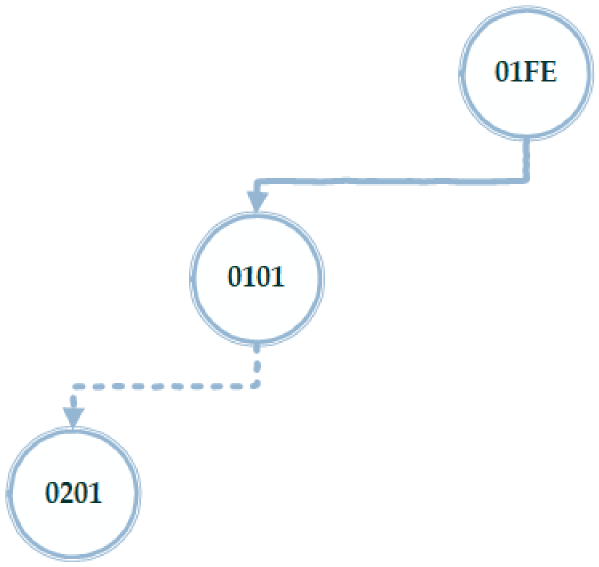


Figure 7: Network Routing Example

Initially, there is only one gateway: 01:254 joined by the first node 0101. The routing table of this node in the beginning is empty. The node tries to join the network and sends a Network Request Message to which the ClusterHead responds with a Network ID 2. At this point, node 0101 will change its state to a router and responds to the requesting node with address: 0201. The routing table is updated with a first entry for 0101:

Child Node N	Address	Networks under child
1	0201	[] – empty array of child networks

Table 7: Routing Table Updated (I)

If the node received another Join Request message, this other node will be given an address of 0202. Routing table will also be updated in a similar fashion.

If 0202 receives a Join Request message from a node, 0202 needs to ask for the address – Network ID from ClusterHead through its parent 0101. The ClusterHead assigns 6 as the Network ID and this ID is sent to the 0202 through 0101. The network routing table for 0101 is updated accordingly. Simultaneously, as node 0202 received the Network ID, it will change its state to router and gives the node address 0601. The updated routing table for 0101:

Child Node N	Address	Networks under child
1	0201	[]
2	0202	[]
3	0203	[6]

Table 8: Routing Table Updated (II)

The complete diagram can be seen in Figure 8 below:

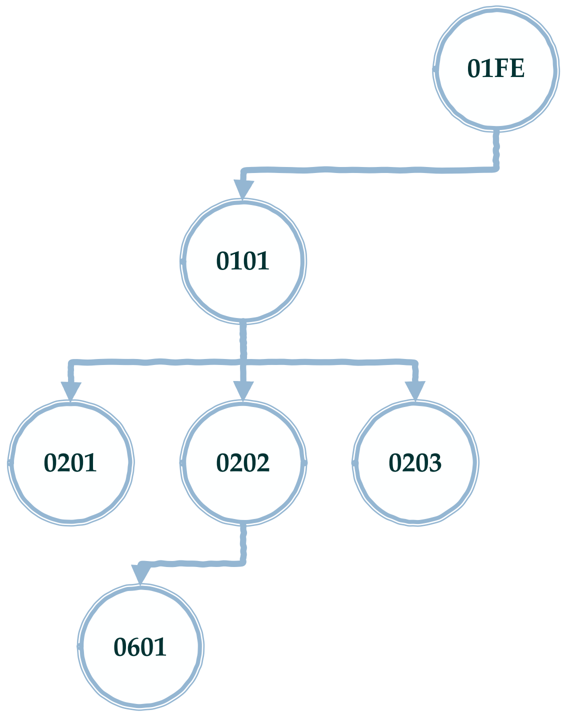


Figure 8: Network Routing Example

To route a packet, the algorithm will check the router and destination addresses. If the router address is the parent, or more specifically, if the parent receives a packet, the network address field is checked for all children from the destination. If found, the packet will be repeated and sent to the next hop. In case of a miss, the packet is dropped.

If the router address is the child, the network address is checked for all children. If it matches, the packet will be sent to the parent and up to the gateway. If the gateway does not see a way to transmit the packet, it will be dropped.

New Cluster: Choosing ClusterHead

The newly registered node receives address after which it can begin to broadcast heartbeat messages. The process is as follows: the registered node received a Join Request Message from an unregistered node that wants to join the network. The ClusterHead node is not eligible to assign addresses and accept new nodes, it can only assign Node IDs. Network ID assignment is done only by the Root Node. Therefore, there are two possible scenarios:

1. The registered will become a ClusterHead by obtaining a Network ID and it will be able to manage its own network and respond to Join Request Messages from unregistered nodes.
2. The unregistered node that sends a Join Request Message will become the ClusterHead after the registered node that became the clusterhead will transfer its role to the new child node.

The second solution is preferred due to its ability to reduce cluster overlap. By reducing cluster overlap, it will reduce traffic and message collision. Before going into the details of a more optimal solution, first, we will describe the initial version of the network setup.

According to the first algorithm, in order to respond to Join Request Messages, a registered node will obtain a Network ID from the root node and will become a ClusterHead. Network IDs are assigned by the RootNodes and the process indicates creating a new cluster. The registered node sends a Network ID Request message to the RootNode. The RootNode will allocate an address and send Network ID Response message to the registered node which becomes a ClusterHead. The new ClusterHead will assign Node ID to the node that was asking to join the network by sending Address Assign message described before. The sequence diagram in Figure 9 illustrates the packet exchange activity.

As we can see, we have two new packets: Network ID Request, and Network ID Response, both of which are shown in Tables 9, and 10.

2 bytes	1 byte	2 bytes
Source Address	Packet Type: RequestNetworkID	Destination Address (RootNode)

Table 9: Network ID Request Message Format

2 bytes	2 bytes	1 byte	2 bytes
Address (CH address of new network)	Source Address	Packet Type: AssignNetworkID	Destination Address

Table 10: Network ID Response Message Format

In total, the packets exchanged if the ClusterHead will be the node existing in the cluster:

1. Probe
2. Heartbeat
3. Join Request
4. *Network ID Request*
5. *Network ID Response*
6. Address Assign
7. Address Acknowledgement

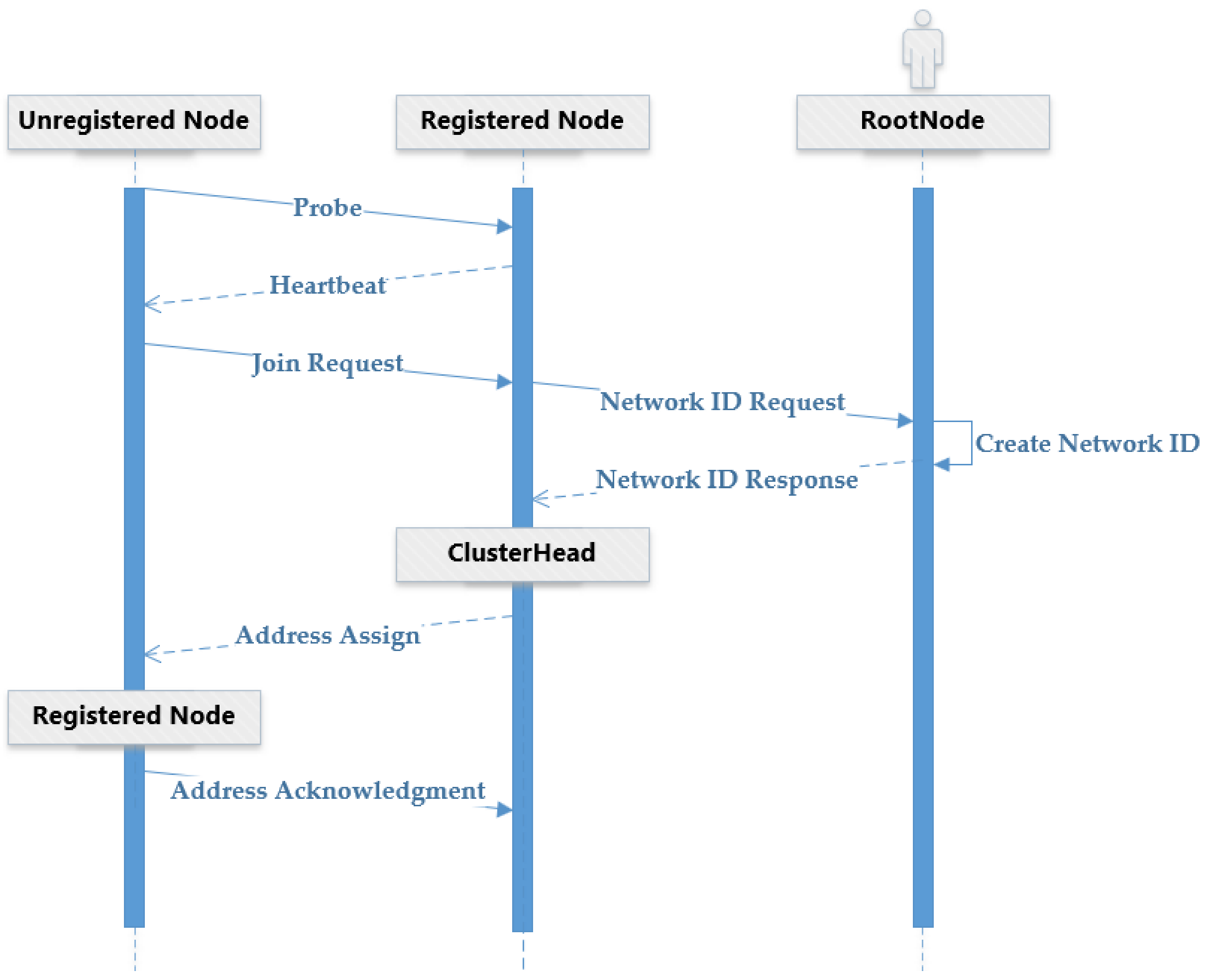


Figure 9: Message Sequence Diagram (I)

The algorithm is shown on SDL activity diagrams in Figure 10 for the undiscovered node, Figure 11 for the unregistered node; similarly Figure 12 demonstrates the SDL diagram for the registered node activity, and Figure 13 illustrates the ClusterHead activity in details.

According to Figure 10, if an undiscovered node receives a HeartBeat packet, it will kill probe timer and become unregistered.

In Figure 11, we see that upon starting the Probe timer, the node might receive several types of packets. The unregistered node may receive a HeartBeat message from the ClusterHead, which means that there is a ClusterHead in this node's transmission range, and it can join its network. In response to the HeartBeat message, this node should send a Join Request message to the HeartBeat packet source. Another type of message that an unregistered node can receive is Join Reply, or Address Assign message, which holds the new Node ID for the potential registered node. If the unregistered node is satisfied with the Join Reply message it will send an Address Acknowledgement packet to finalize the joining process and acknowledge its new network.

Figure 12 demonstrates the activity of the registered node in case it becomes a ClusterHead. If this node receives a Probe packet from an unregistered node, it will simply respond with a HeartBeat. The action is different if the registered node receives a Join Request message. Upon receiving this message, the registered node sends a Network ID Request to the RootNode which should provide a Network ID. Once the registered node successfully is assigned a Network ID, this node will become a ClusterHead of its own network. It will send a Join Reply message to the unregistered node that was asking to join the network which will make the unregistered node registered inside this new network.

In Figure 13, we can observe the ClusterHead’s activity. The ClusterHead will start listening to the channel. Depending on the received packets and timer duration, the behavior of the network will be different.

As shown in the SDL diagram in Figure 13, the ClusterHead may receive several types of packets. If the received packet is Probe, the ClusterHead in return, will send a HeartBeat message. If the received packet is Join Request, respond by transmitting a Join Reply message to assign a Node ID so that the node will be able to become to change its state to registered and will become a part of the network. If the packet received by the ClusterHead is Network ID Request, the ClusterHead will check its state to make sure it is eligible to assign network address. If it is a RootNode, it will reply with a Network ID Response message which contains the Network Address. Otherwise, if packet type is Address Acknowledgment from the newly registered node, this node will be added to the member’s table. Member’s table simply consists of joined, registered nodes.

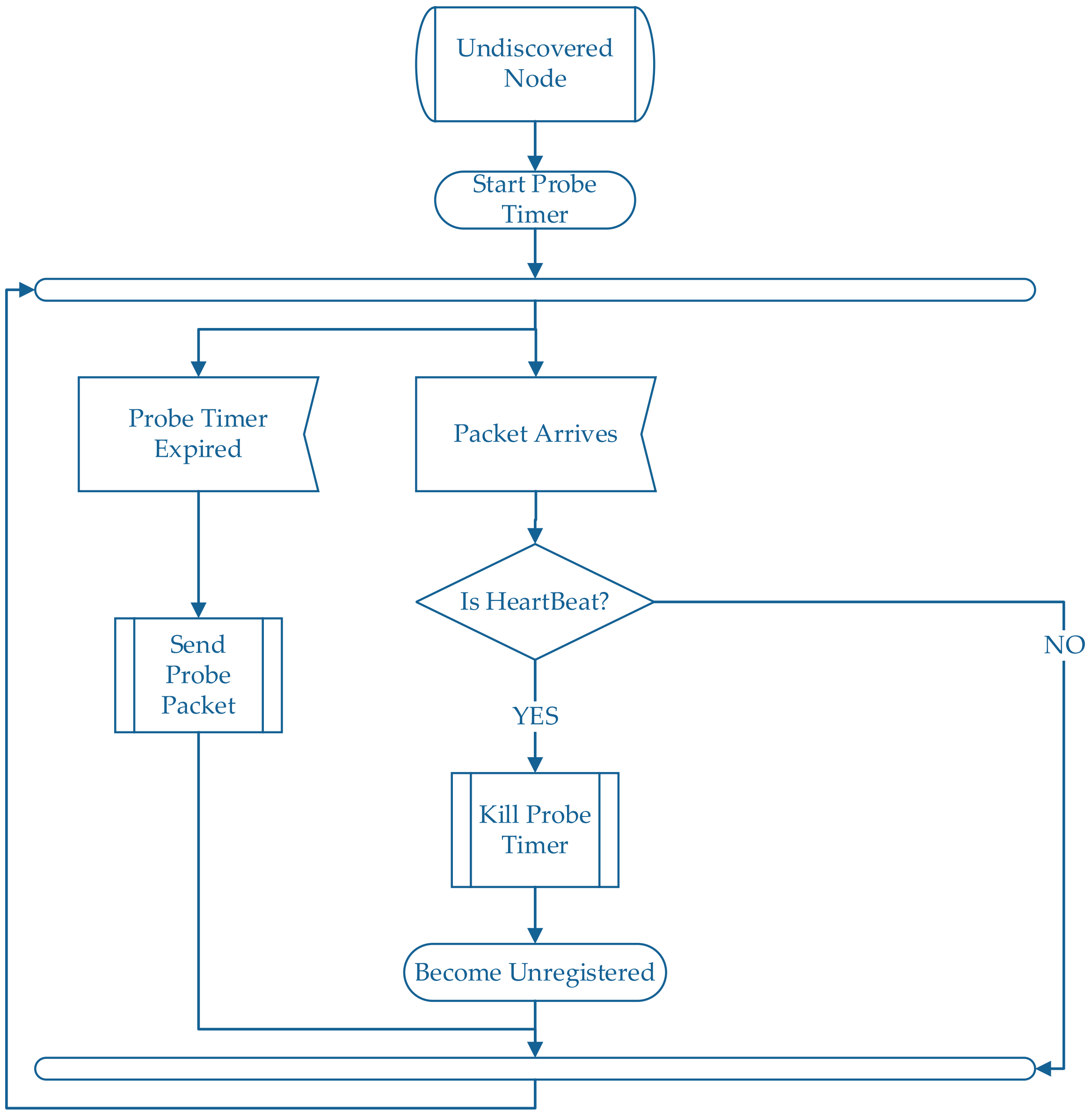


Figure 10: SDL of the Undiscovered Node (I)

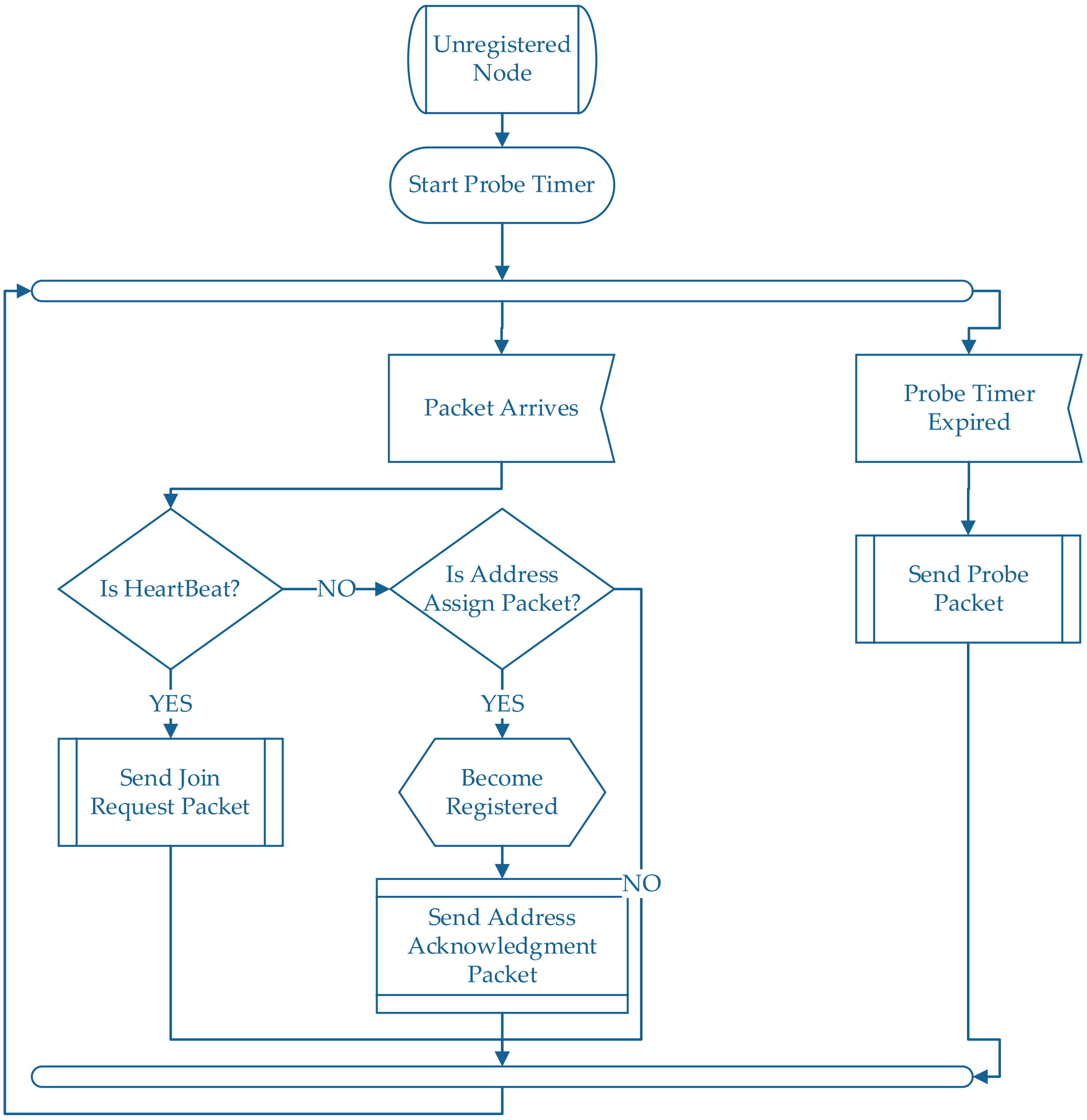


Figure 11: SDL of the Unregistered Node (I)

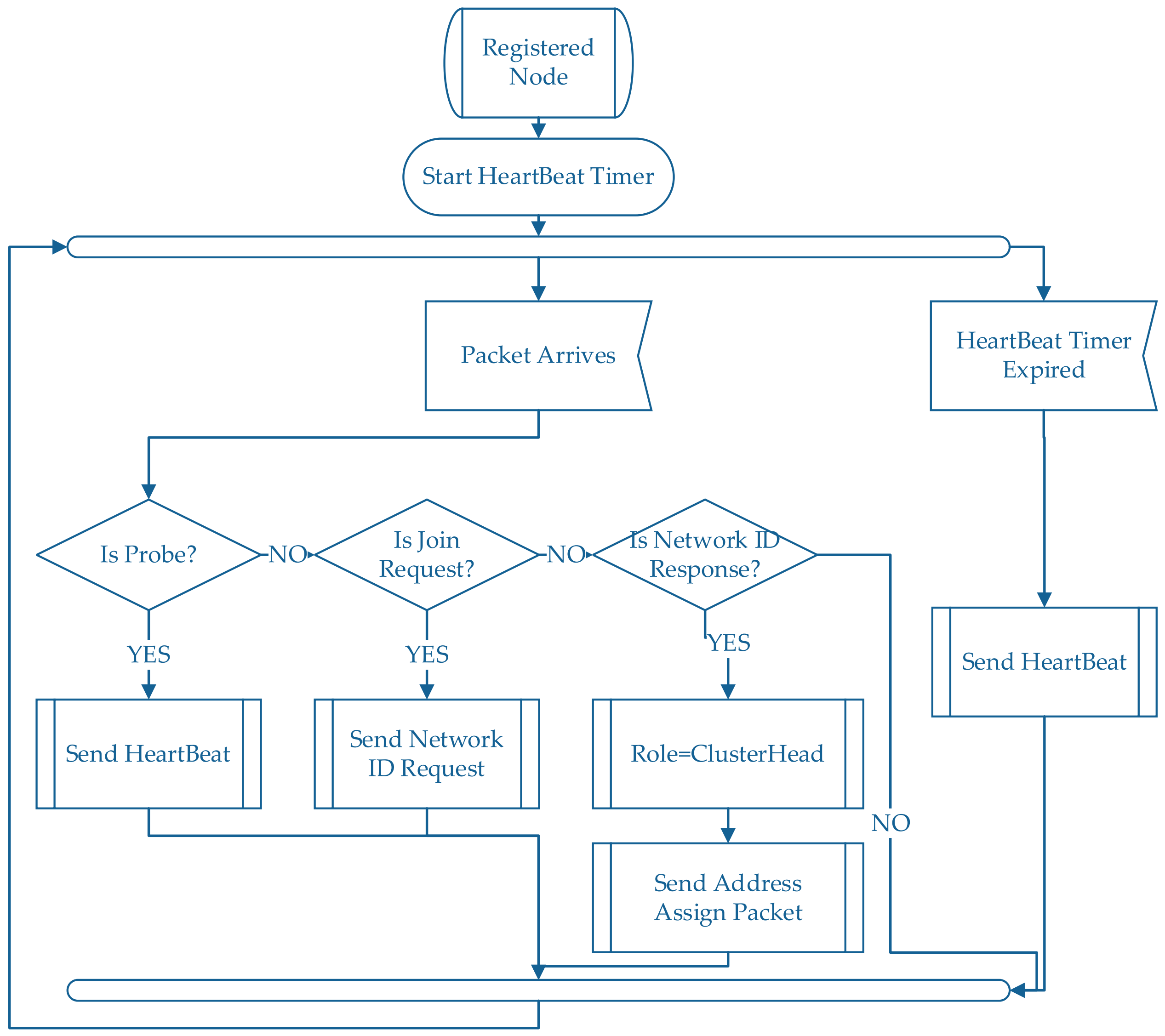


Figure 12: SDL of the Registered Node (I)

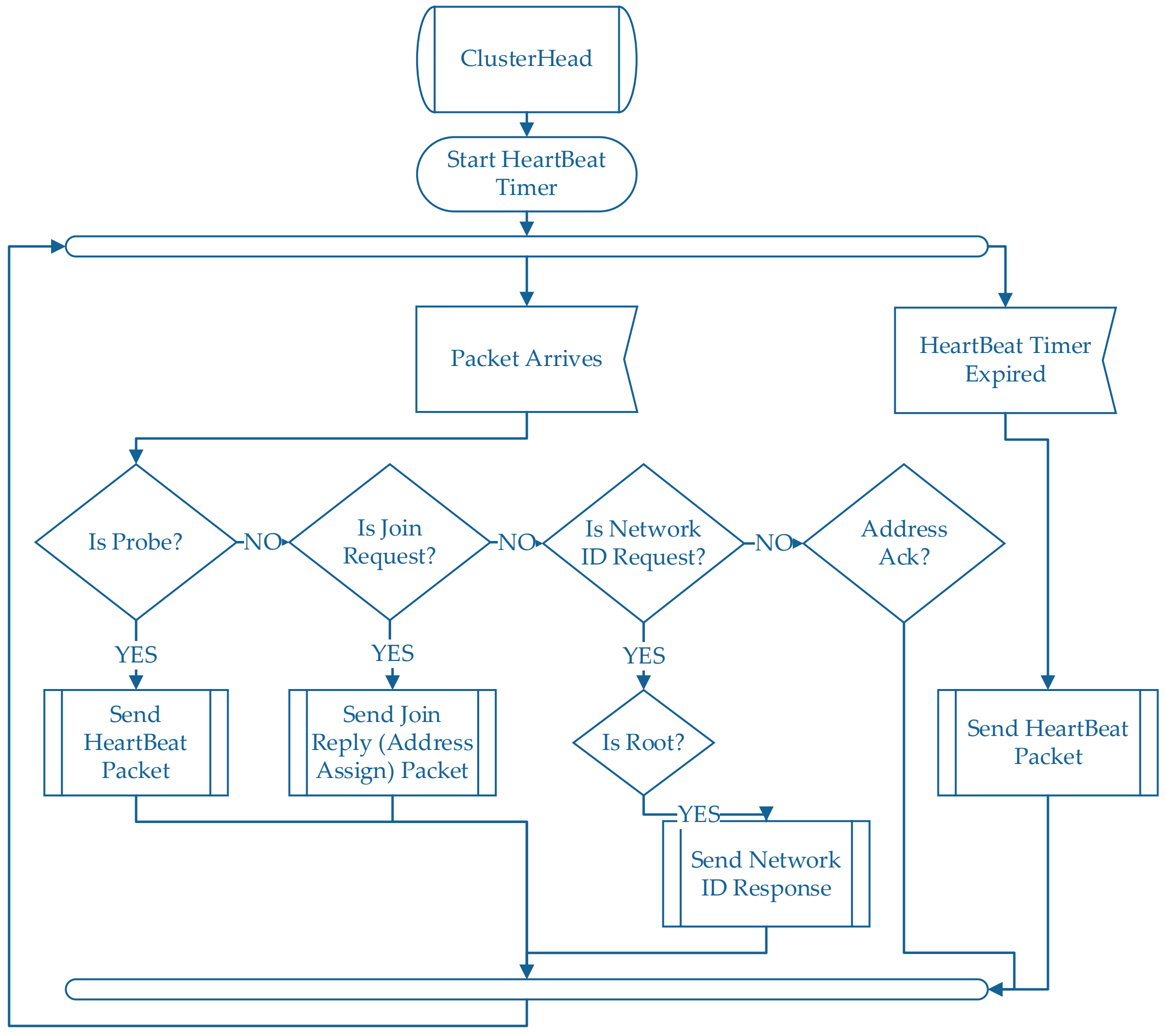


Figure 13: SDL of the ClusterHead (I)

New Cluster: Represent Unregistered Node as the ClusterHead

To minimize the overlapping area between the clusters and solve the hidden terminal problem, the initial algorithm needs to be changed so that the unregistered node that wants to join the network becomes the ClusterHead.

The registered node sends a Network ID Request message to the RootNode. The RootNode will allocate an address and send Network ID Response message to the registered – this is the same process that was described when analyzing the first algorithm. The new ClusterHead will send a Join Reply message to the unregistered node as described previously. The only difference is that after that, a very short timer – ClusterHead Transfer Timer will start to transfer this role to the newly joined node that was asking to join the network. The new node will send ClusterHead Acknowledge Message to the previous ClulsterHead. After receiving the ClusterHead Acknowledgment message, the ClusterHead will become a router, it's child – the node that was unregistered and was asking to join the network, will become a ClusterHead. The sequence diagram in Figure 14 demonstrates the packet exchange activity between nodes with different roles.

As we can see, we have two new packets: ClusterHead Transfer and ClusterHead Acknowledgment, both of which are shown in Tables 11 and 12.

6 bytes	**2 bytes**	**1 byte**	**2 bytes**
GUID	Source Address	Packet Type: ClusterHear Transfer	Destination Address
			Address (Address of the new CH)

Table 11: ClusterHead Transfer Message Format

6 bytes	**2 bytes**	**1 byte**	**2 bytes**
GUID	Source Address	Packet Type: CH Acknowledgment	Destination Address

Table 12: ClusterHead Acknowledgment Message Format

Therefore, in total, the packets exchanged if the ClusterHead will be the new node joining the network:

1. Probe
2. Heartbeat
3. Join Request
4. Network ID Request
5. Network ID Response
6. *CH Transfer*
7. *CH Acknowledgment*

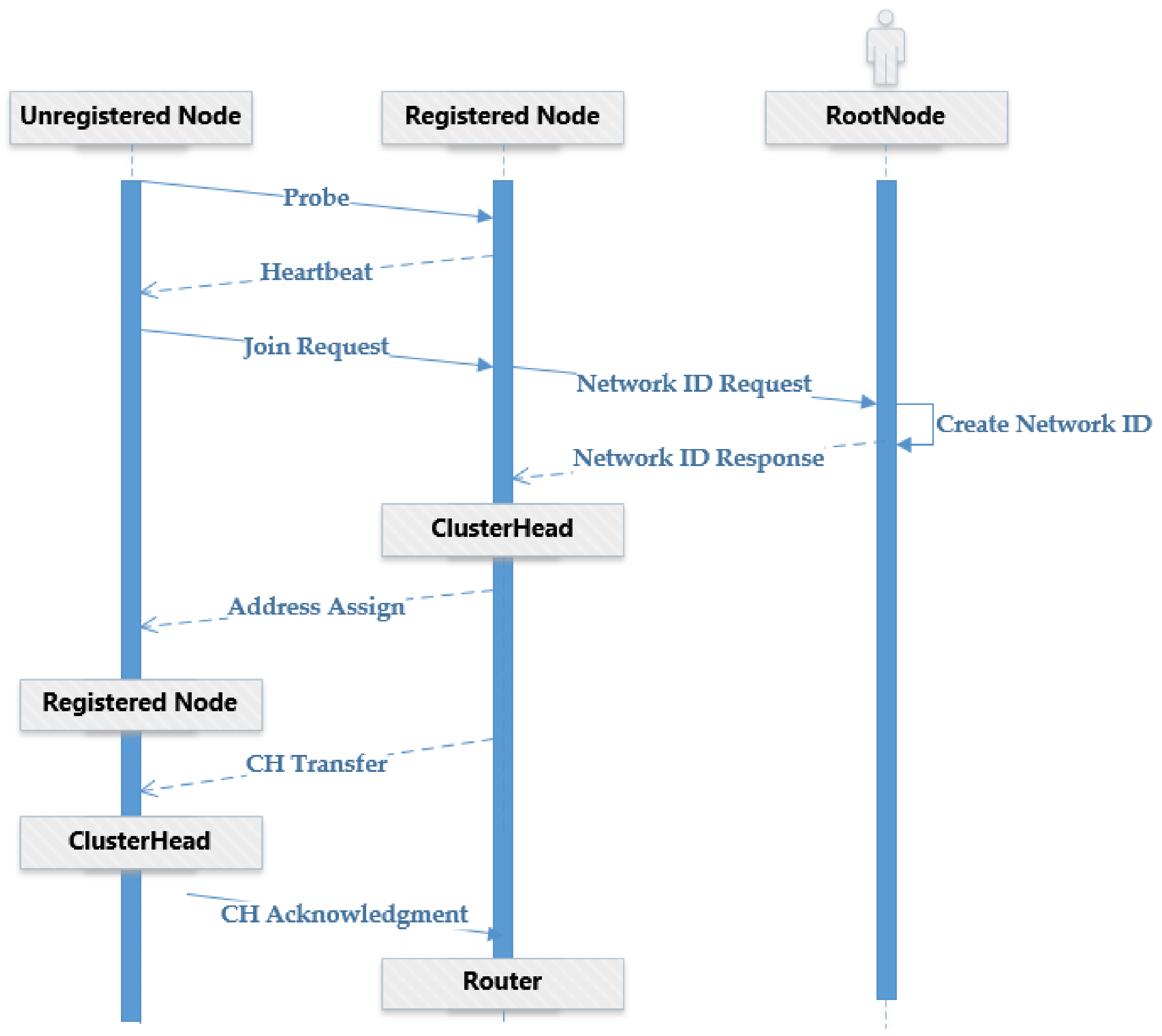


Figure 14: Message Sequence Diagram (II)

Since the only node roles that are affected by the changes are, unregistered and registered nodes, the SDL diagrams are demonstrated only for them. The SDL diagram for the unregistered node that becomes the ClusterHead is demonstrated in Figure 15.

In Figure 15, we see that upon starting the Probe timer, the node might receive several types of packets. The unregistered node may receive a HeartBeat message from the ClusterHead, which means that there is a ClusterHead in this node's transmission range, and it can join its network. In response to the HeartBeat message, this node should send a Join Request message to the HeartBeat packet source. Another type of message that an unregistered node can receive is Join Reply, or Address Assign message, which holds the new Node ID for the potential registered node. The unregistered node, at this point, will become registered. As ClusterHead Transfer Timer starts, the newly registered node will become a ClusterHead and send a ClusterHead Acknowledgment packet to the ClusterHead Transfer message source.

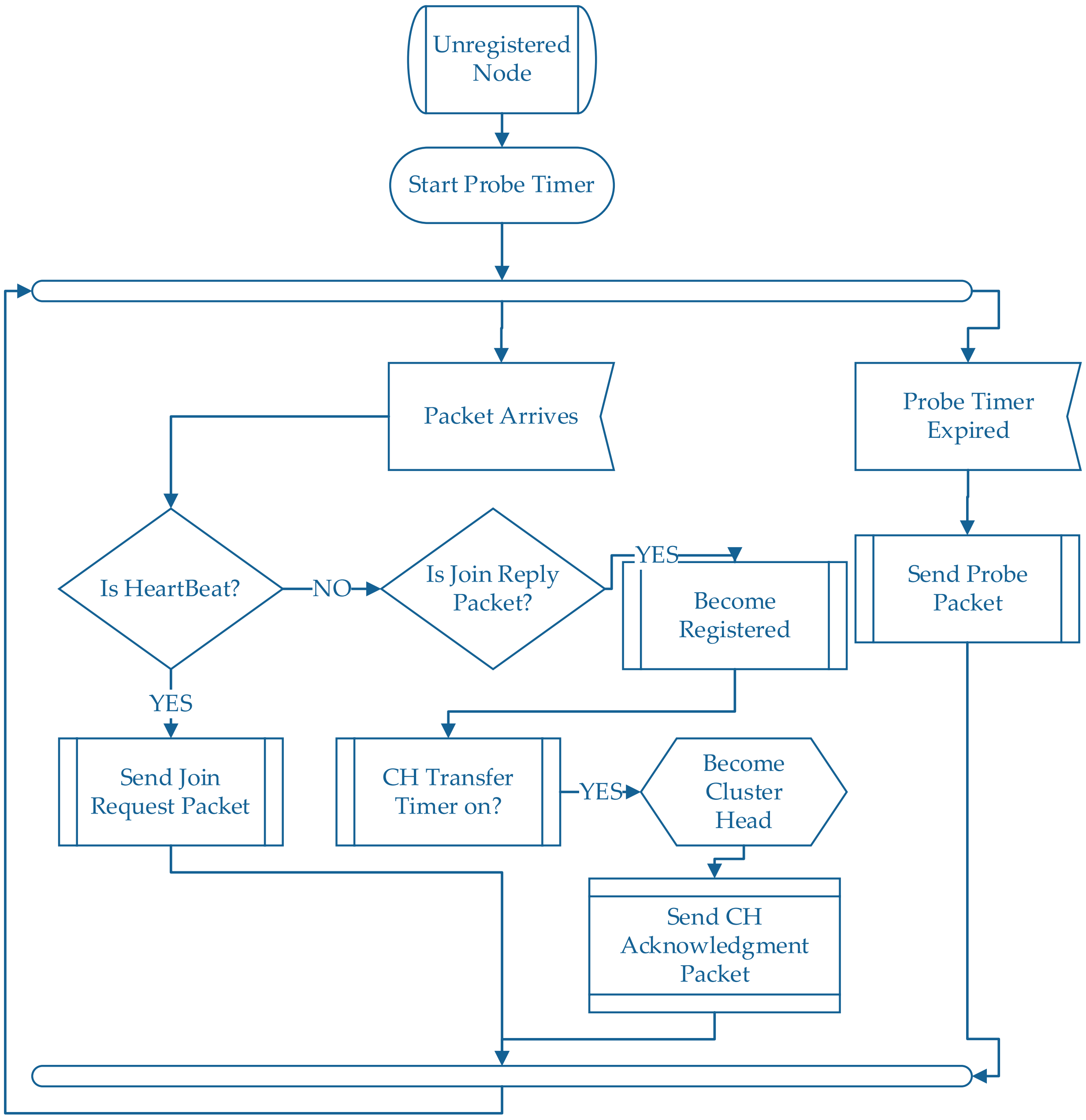


Figure 15: SDL of the Unregistered Node (II)

Figure 16 shows how a registered node that received a probe message, became a ClusterHead, transferred this role to its child node, and became a router for messages to the upper layers.

As shown in the SDL diagram in Figure 16, the registered node may receive several types of packets If this node receives a Probe packet from an unregistered node, it will simply respond with a HeartBeat. The action is different if the registered node receives a Join Request message. Upon receiving this message, the registered node sends a Network ID Request to the RootNode which should assign a Network ID. Once the registered node successfully is assigned a Network ID, this node will become a ClusterHead of its own network. It will send a Join Reply message to the unregistered node that was asking to join the network which will make the unregistered node registered inside this new network. As ClusterHead Transfer Timer starts, the newly registered node will become a ClusterHead and send a ClusterHead Acknowledgment packet to the ClusterHead Transfer message source – ClusterHead. At this point, the ClusterHead will become a router.

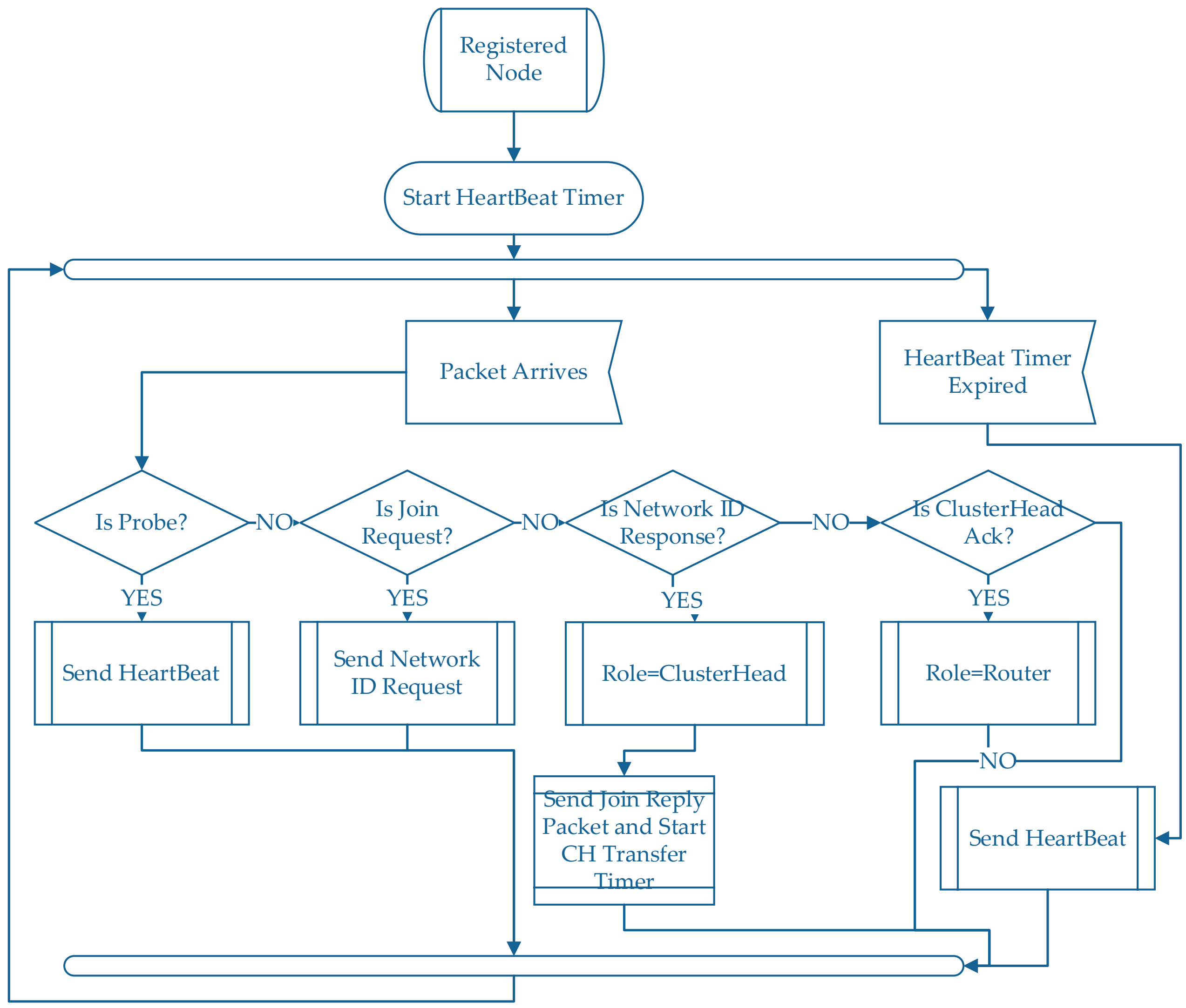


Figure 16: SDL of the Registered Node (II)

ClusterHead Role Mobility

Since packet transmission and reception takes a lot of energy, it is a very desirable that the role of the ClusterHead is movable. ClusterHead mobility is a feature that extends lifetime of a network and solves hidden terminal problems.

To make this role mobile, we need to consider the overlapping area between networks, and we need to make it as small as possible. This will happen if we follow certain rules.

First of all, the most important rule will be that the ClusterHead of the new cluster will be the node joining the network, and the sequence diagram, as well as the SDLs will be the same as discussed above.

If the distance between the center and the circumference of the network circle (essentially, the radius) is r, then the distance between the center this network and another network with which the network has some overlapping area should be less than 2r: $d < 2r$. The ClusterHead will be placed in the center of each network, so my idea is to base the assignment of the ClusterHead role on the location. This should minimize the overlapping area between networks and requires the role to be mobile. Overlapping node will be used as routers and nodes in the network will be able to communicate with their ClusterHeads based on the above-mentioned distance rule. The node inside the overlapping area that is supposed to become a gateway is the node that was supposed to be a ClusterHead but it transferred its role to the new node using the ClusterHead Transfer message.

This protocol will minimize the area that networks are sharing so that the ClusterHead is not part of two networks and more importantly, this method utilizes ClusterHead mobility.

The protocol will transform the network if every node is publishing a HELLO message. HELLO packets exchanged by nodes are placed in tables that shows node ID and RSSI. This information will be reported to the centralized node. Everybody knows who they can hear, entire topology is known; distance is known by looking at RSSI values. We should be able to tell how far nodes are from each other. The central node collects information about every node and decides who cluster center should be and assigning them to their roles.

We should have at least one node between any two clusters to make sure our clusters are connected. As new nodes are added to the cluster, we will move ClusterHead role to another location to make sure new ClusterHead covers larger portion and reduces overlapping area between nodes and that location will be the center of the network and this will follow the distance rule. Overall, the ClusterHead mobility will be utilized, overlapping area will be minimized, and network lifetime will extend.

As soon as the unregistered node joins the network and becomes a registered node, it will start collecting information about its neighboring nodes using RSSI (Received Signal Strength Indicator) values.
The HELLO message is described in Table 13.

2 bytes	1 byte	2 bytes	2 bytes
Source Address	Packet Type: HELLO	Nest Hop Address	Destination Address
Node ID N		RSSI	

Table 13: Hello Message Format

This message includes the Node ID that the node can hear with its RSSI in a connection. This record is very useful to understand the radio circle of the node and the general situation about the node in terms of its location.

Once each node receives the message, HELLO message will be appended, and the repackaged information is forwarded to their ClusterHead nodes.

This is a theoretical assumption which was not used in implementation. The implementation is based on self-optimization by picking the minimum number of hops. If a node sees that another node wants to join, it sends a network request to the RootNode, becomes a ClusterHead, sets a very short timer as the new node joins the network, and makes this new node a ClusterHead. Therefore, when the cycle is completed, a new timer is set up to make changes and optimize the network. This addition to the design made the network better organized in terms of reducing message collision but did not minimize the probability of the disadvantage.

To make the network as optimized as possible, we decided to modify the joining algorithm by selecting a better parent to connect to.

Firstly, the node that is trying to join the parent, will check its candidate parent's table and try to choose a minimum hop parent. If in the neighborhood, the RootNode is the closest, the node will connect to it and stop trying to choose another parent by breaking out of the loop. If ClusterHead is in the neighborhood, and either hop count is minimum, or minimum hop neighbor is registered, this case will be chosen as the minimum hop. If none of these situations happen, then, by default, minimum hop count will be chosen based on the minimum hop count to the RootNode. In that case, if the minimum hop’s role is ClusterHead, this node will be sent Join Request. Otherwise, if minimum hop’s role is Registered, this registered node will receive a Join Request message.

After running multiple simulations, and observing the network behavior, we decided to include a condition when the node’s role is registered and neighbor’s role is ClusterHead, or when the neighbor’s role is registered, they will receive Join Request packets. Including this particular case, we have found that ClusterHead role distribution is more uniform, and we eliminated seeing multiple unnecessary ClusterHeads in close neighborhood.

After accomplishing all the above-mentioned ideas, we decided to check one more time whether there is a better parent option for the unregistered node to select destination address for the Join Request message.

<u>Algorithm for self-optimizing network</u>

```
1: Check candidate parent’s table
2:    Search for min hop nodes in neighbor’s table
3:       Select this address
4:    if neighbors_table[gui][role] == Root
5:       Select this address and break
6:    if neighbors_table[gui][role] == ClusterHead
7:       min hop nodes in neighbor’s table OR if neighbors_table[min_hop_gui] == Registered
8:       Select this address
9:    else
10:      Search for min hop nodes in neighbor’s table
11:         if neighbors_table[min_hop_gui] == ClusterHead
12:            if this is min hop
13:               Select this address
14:         if neighbors_table[min_hop_gui] == Registered
15:            Select this address
16: Check candidate parent’s table
17:    Search for min hop nodes in neighbor’s table
18:       if neighbors_table[gui][role] == ClusterHead AND role == Registered
19:          Select this address
20:       if neighbors_table[gui][role] == Registered
21:          Select this address
22: if there is a min hop that is not already a parent
23:    Select this address
24:    Send Join Request
```

Figure 17 is an illustration of the self-optimized network where registered nodes are represented with green color. Blue nodes 2, 3, 5, 6, 14, 18, 21, 25, 29 are ClusterHeads, the remaining purple nodes: 7, 8, 10, 11, 13, 15, 19, 23 are routers.

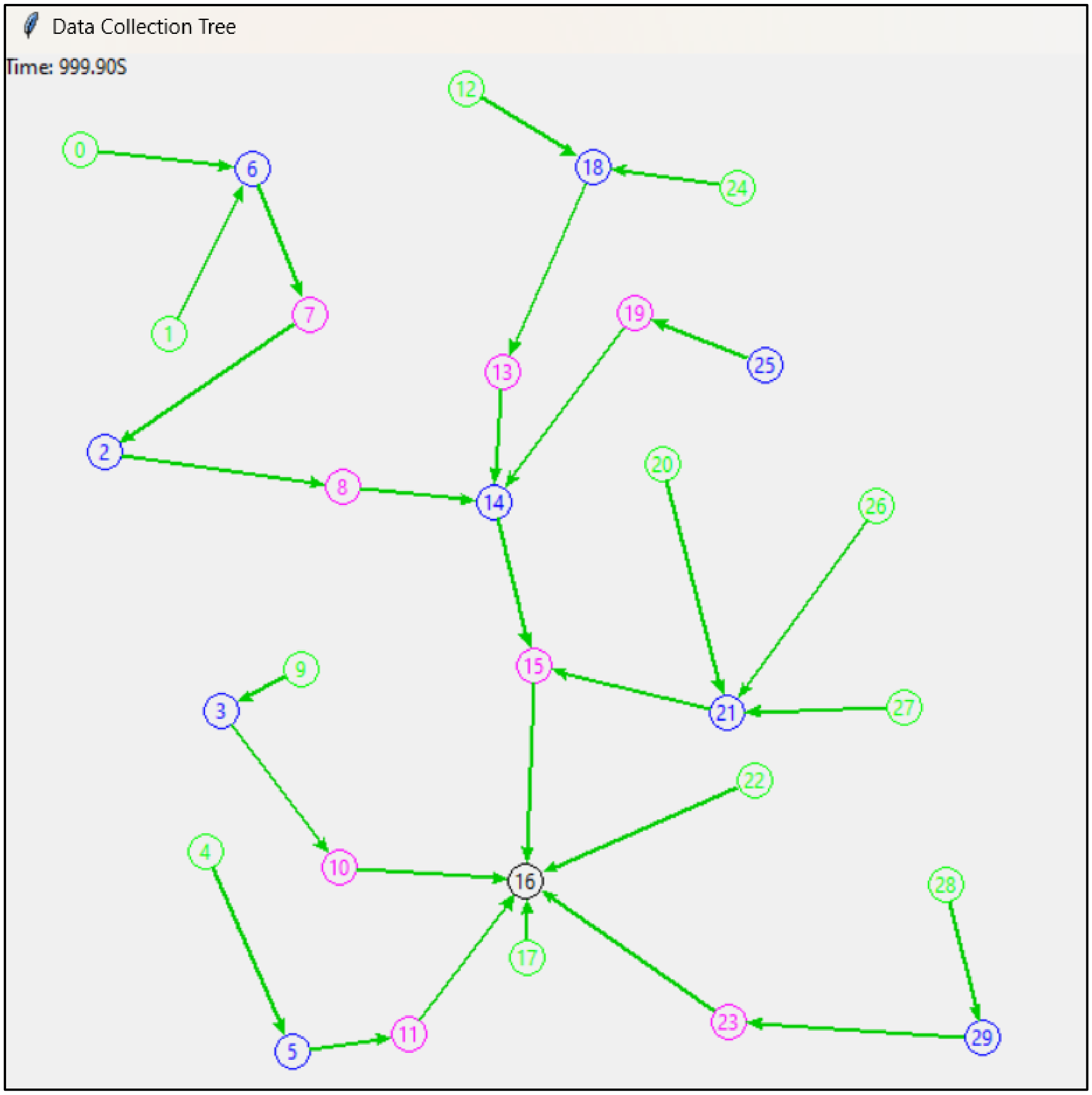


Figure 17: Self-Optimized Network

Sensor Node State Machine

WSN node's main components are controller, power source, transceiver, external memory, and sensors. A typical architecture is shown in Figure 18.

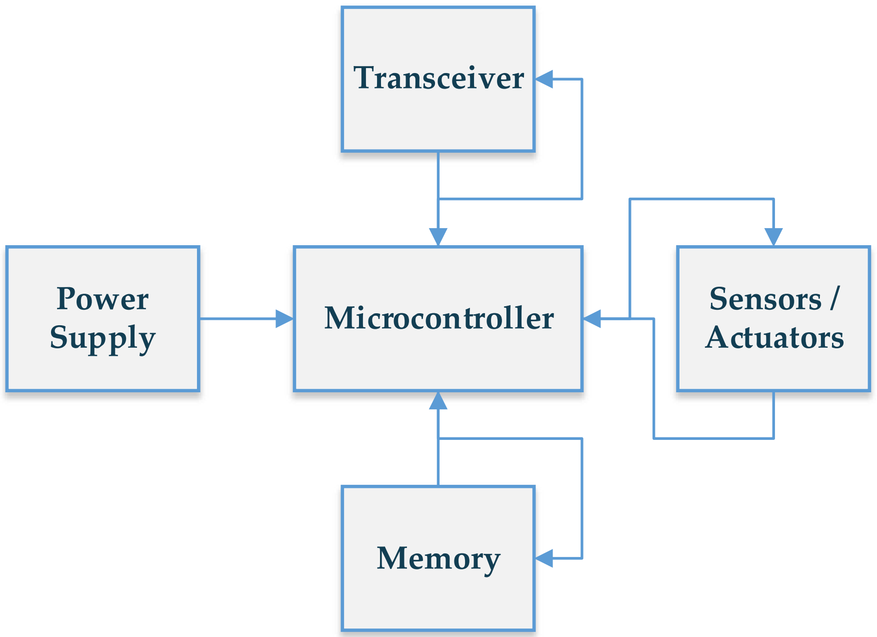


Figure 18: Sensor Node Architecture

Transceiver's state machine consists of the following states:

1. Sleep – unable to receive immediately; requires recovery time and startup energy to leave this state.
2. Idle – ready to receiver but not yet receiving. Idle state is good to switch off some hardware functionalities and reduce energy consumption.
3. Receive
4. Transmit

As we can observe in Figure 19, sensor node is supposed to sample and transmit the data to the root at predefined intervals which in our case if the remainder of each node's GUID divided by 20 – this is the length of the sensor timer. Sensor node records the data in the node's unregistered state when the received packet is Address Assign. The energy is displayed for every sensor node.

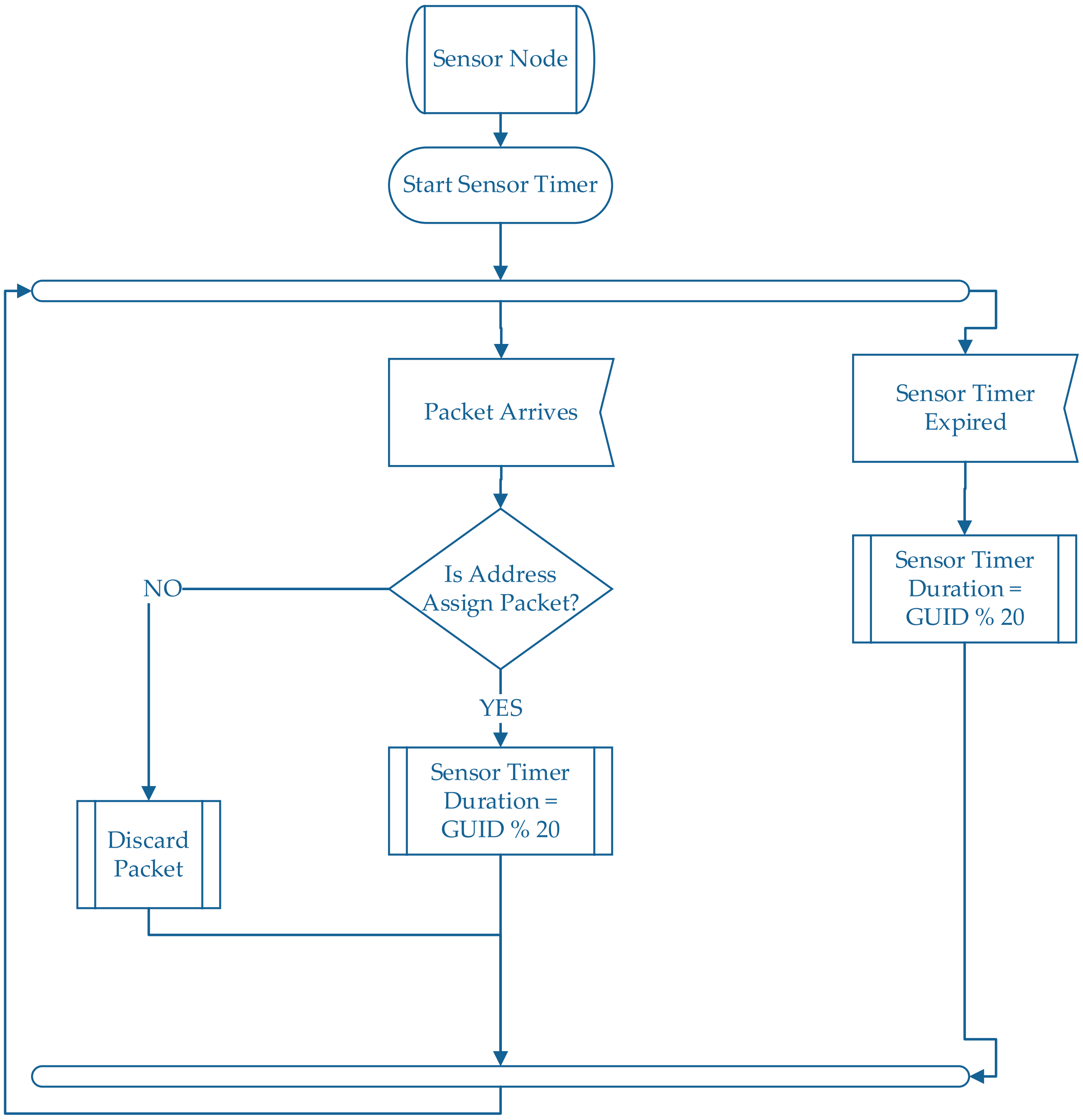
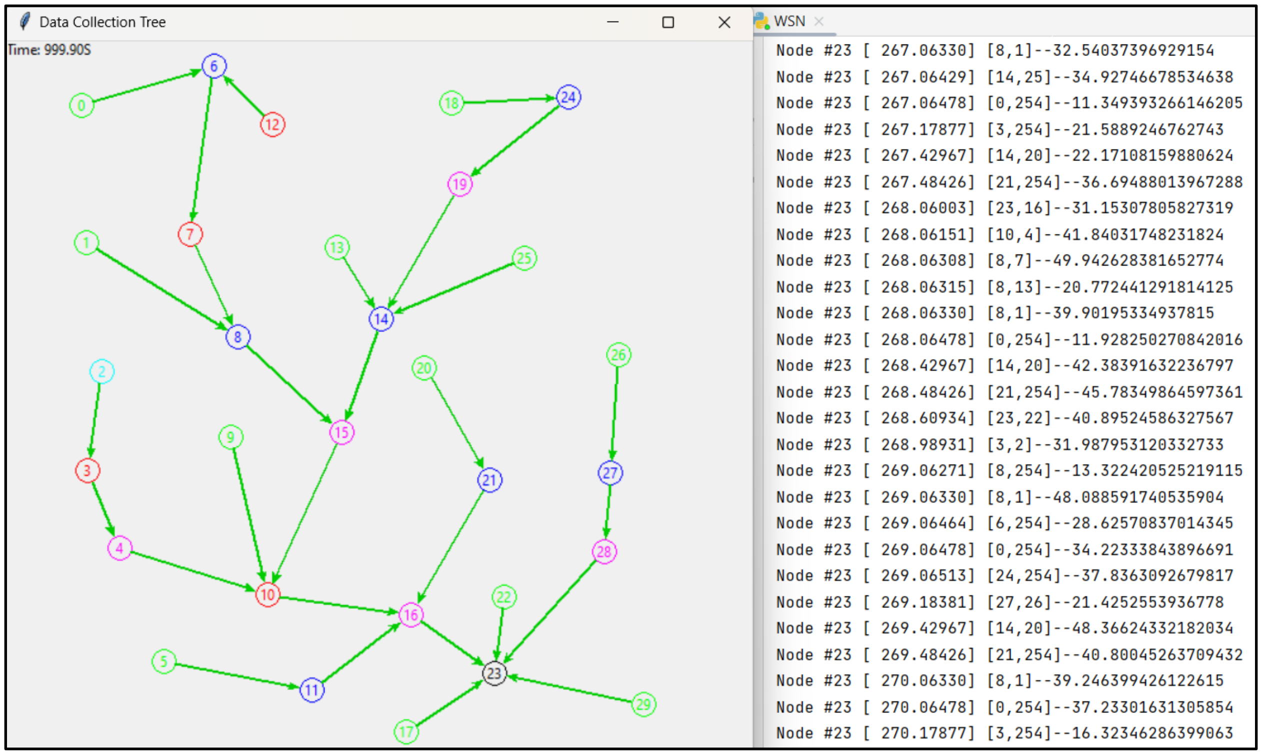

Sensor Node
Start Sensor Timer
Packet Arrives
Sensor Timer Expired
Is Address Assign Packet?
NO
YES
Discard Packet
Sensor Timer Duration = GUID % 20
Sensor Timer Duration = GUID % 20

Figure 19: SDL of the Sensor Node

Figure 20 demonstrates sensor node sampling and transmitting data to the root at predefined intervals integrated into the WSN.



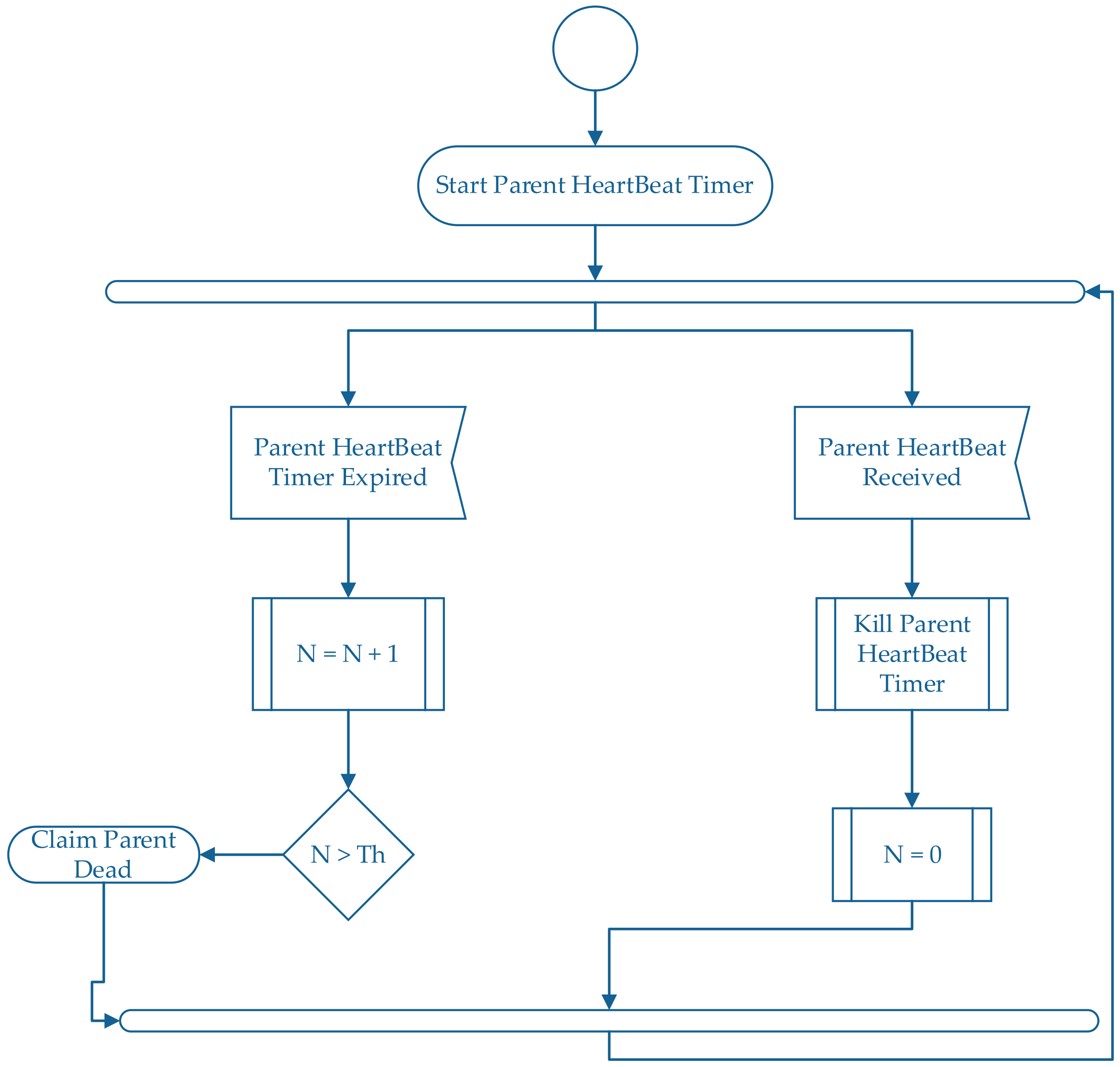
```
Node #23 [ 267.06330] [8,1]--32.5403739692915 4
Node #23 [ 267.06429] [14,25]--34.92746678534638
Node #23 [ 267.06478] [0,254]--11.349393266146205
Node #23 [ 267.17877] [3,254]--21.5889246762743
Node #23 [ 267.42967] [14,20]--22.17108159880624
Node #23 [ 267.48426] [21,254]--36.69488013967288
Node #23 [ 268.06003] [23,16]--31.15307805827319
Node #23 [ 268.06151] [10,4]--41.84031748231824
Node #23 [ 268.06308] [8,7]--49.942628381652774
Node #23 [ 268.06315] [8,13]--20.772441291814125
Node #23 [ 268.06330] [8,1]--39.90195334937815
Node #23 [ 268.06478] [0,254]--11.928250270842016
Node #23 [ 268.42967] [14,20]--42.38391632236797
Node #23 [ 268.48426] [21,254]--45.78349864597361
Node #23 [ 268.60934] [23,22]--40.89524586327567
Node #23 [ 268.98931] [3,2]--31.987953120332733
Node #23 [ 269.06271] [8,254]--13.322420525219115
Node #23 [ 269.06330] [8,1]--48.088591740535904
Node #23 [ 269.06464] [6,254]--28.62570837014345
Node #23 [ 269.06478] [0,254]--34.22333843896691
Node #23 [ 269.06513] [24,254]--37.8363092679817
Node #23 [ 269.18381] [27,26]--21.4252553936778
Node #23 [ 269.42967] [14,20]--48.36624332182034
Node #23 [ 269.48426] [21,254]--40.80045263709432
Node #23 [ 270.06330] [8,1]--39.246399426122615
Node #23 [ 270.06478] [0,254]--37.23301631305854
Node #23 [ 270.17877] [3,254]--16.32346286399063
```

Figure 20: Sensor Node Implementation

Recovery from Node and Link Failures

Nodes can fail anytime so we should develop a protocol that takes into account node failure and is able to recover nodes from failure. HeartBeat messages are used to detect the existence of the nodes. As shown in Figure 21, we should wait for several HeartBeats to make sure the node is dead and there is no signal coming from it.

The node that fails is removed from the ClusterHead's address table. The ClusterHead waits for some time, waits for several missed heartbeats to retrieve the address.



Start Parent HeartBeat Timer
Parent HeartBeat Timer Expired
Parent HeartBeat Received
N = N + 1
Kill Parent HeartBeat Timer
Claim Parent Dead
N > Th
N = 0

Figure 21: SDL to Detect Node Failure

Once a node realizes that a parent is dead, there are two options:

1. Reset downstream network - send a Reset Network Packet downstream so that the nodes will go to the unregistered state.

2. "Take Your Children with You" - after a node discovers that it is orphan, it does not share the information with the other nodes. It will send a probe to obtain address. Some nodes will respond with a HeartBeat message. That node will collect heartbeats. Once it successfully receives these heartbeats, it will select a candidate parent, send a Join Request and receive address. Additionally, what it needs to do is to create a Network Update packet and send upstream to inform the upstream routers about the child networks. The process is discussed below.

The ClusterHead that fails or changes location, will be removed from the network. Its child (or children) will connect to the network and choose the parents in a similar way that was described above, and they will be assigned new Node IDs and the parents will "take all its children with it" and no additional changes will occur.

The figures 22 and 23 demonstrate how a ClusterHead for network 2, 0101 fails. 0201, 0202, 0203 will become orphans and will need to rejoin the network as they did initially. All the nodes in network 3 will be taken by the parent 0201, the ClusterHead.

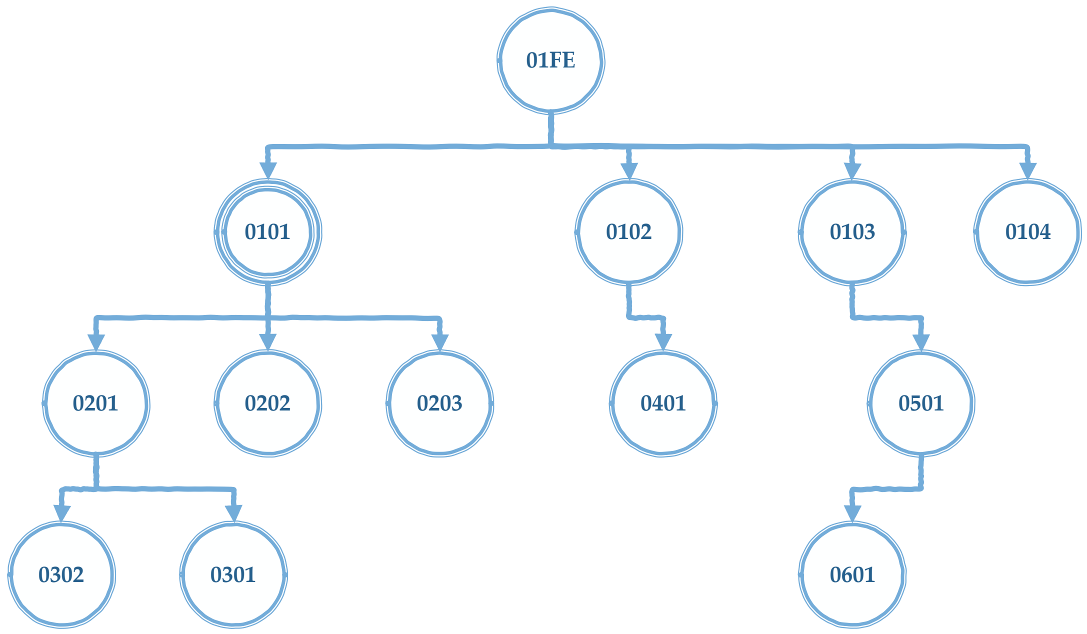


Figure 22: Network Before Node Failure

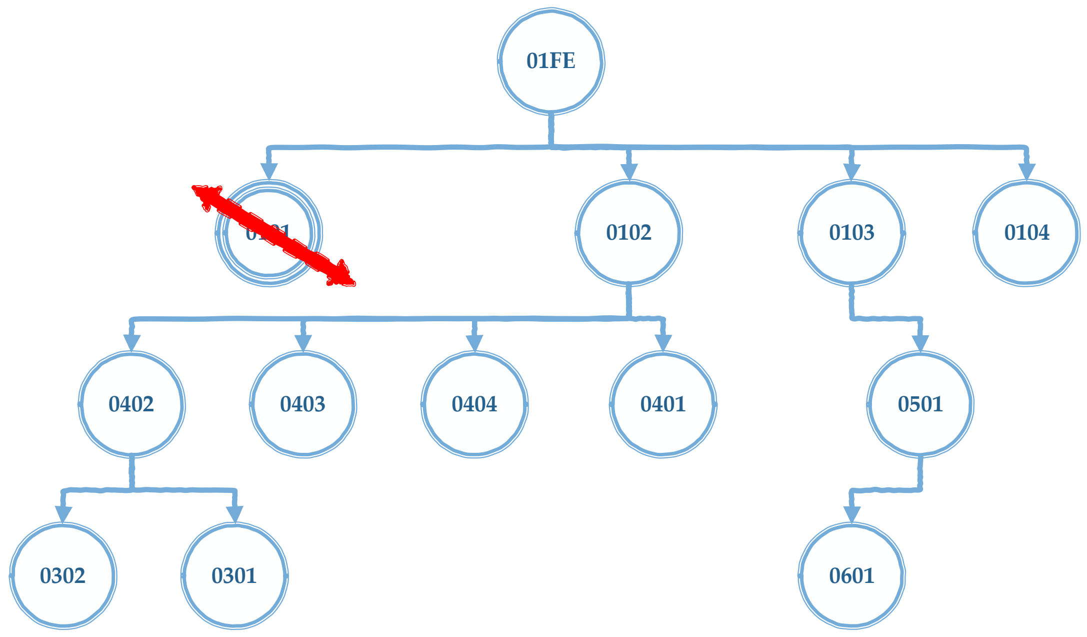


Figure 23: Network After Node Failure

Figure 24 shows the sequence diagram after the parent node fails. As we can see, the child nodes will send probe messages to which they will receive HeartBeats. After receiving HeartBeat messages, Join Request is sent to join another parent and its network. As soon as the parent received Network ID Response to its Network ID Request to the RootNode, it will become a ClusterHead and will let the orphans join it by giving them Node IDs. If the network is self-optimizing, the rules for self-optimization will be followed here, as well.

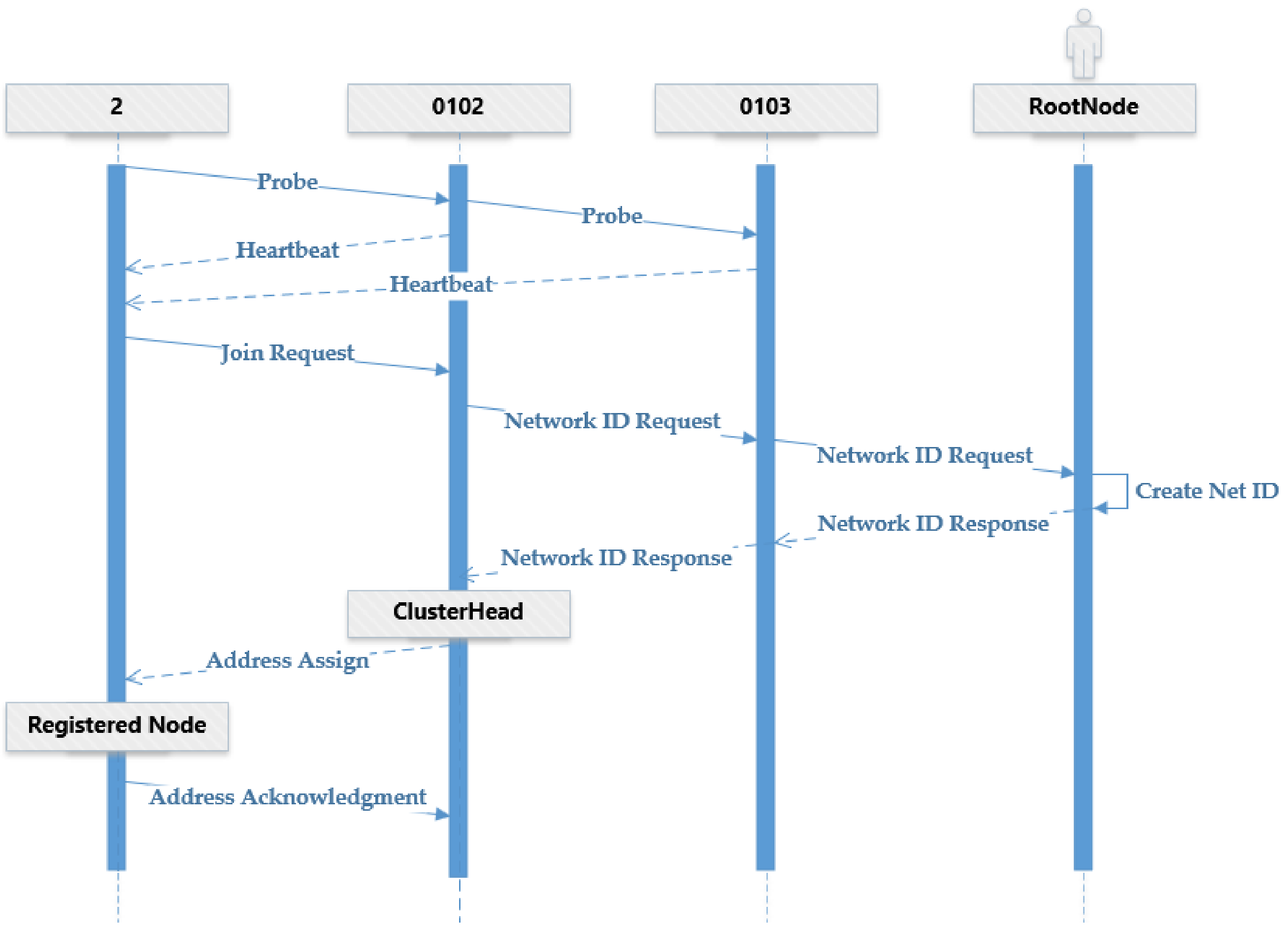


Figure 24: Sequence Diagram for Node Failure

As it can be seen in Figure 23, Nodes 0201, 0202, 0203 registered under 0102 as 0402, 0403, 0404. Nodes 0301, 0302 maintained their IDs as no changes were necessary, the node failure did not have any impact on them. Message exchange process during the failure of network 2 is described in Figure 24.

"Take Your Children with You" is an efficient way to deal with network failure, because when a ClusterHead is disconnected, it will move on and reconnect to another branch of the tree with its children. This reduces network control traffic and saves time as it involves just one node that needs to join the network.

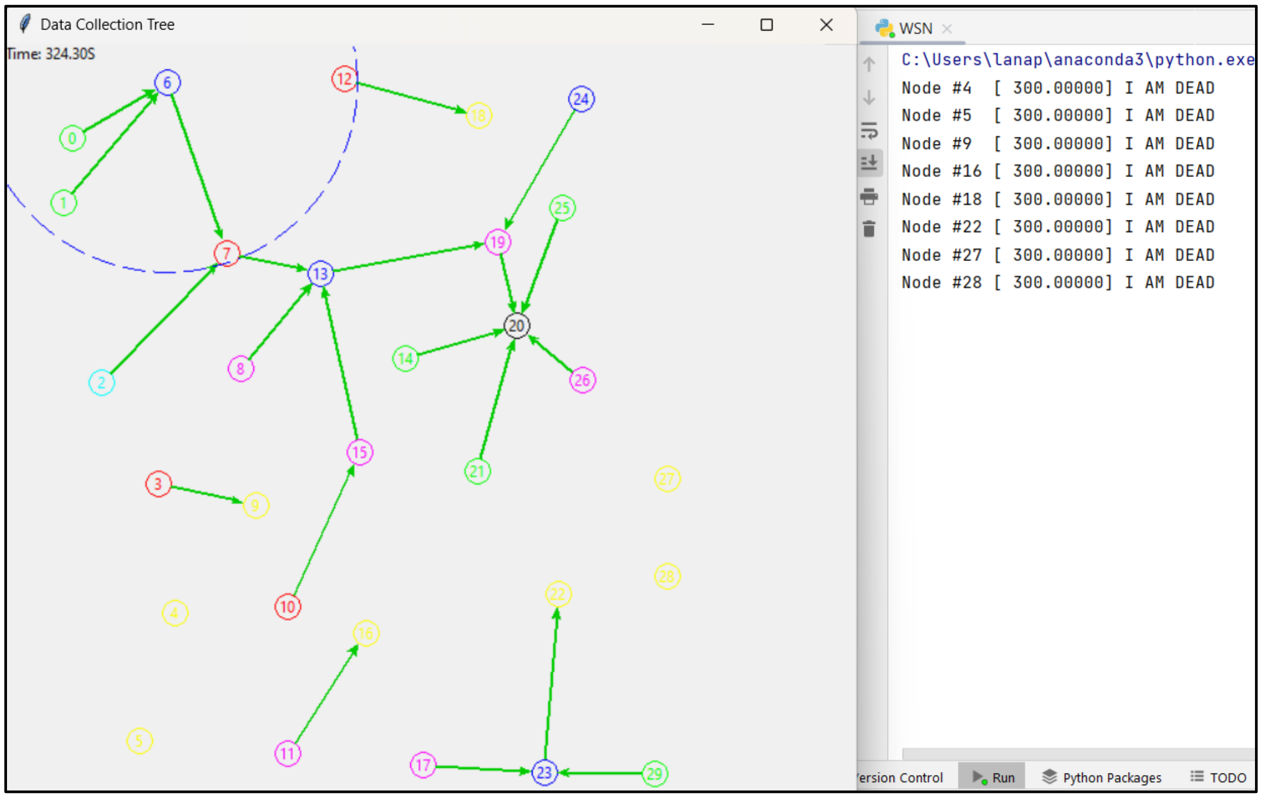
Table 14 shows a message that Node 0402 will send to its new parent 0102 to inform it about the child networks.

6 bytes	2 bytes	1 byte	2 bytes
GUID	Source Address	Packet Type: Network Update	Destination Address (neighbors table [parent])
	Child Network ID		

Table 14: Network Update Message Format

For the situation when the failed node comes back to the network, it will join the network from unregistered (initial) state like any other new node would join the network.

In Figure 25, we can see how some of the nodes failed.



```
C:\Users\lanap\anaconda3\python.exe
Node #4  [ 300.00000] I AM DEAD
Node #5  [ 300.00000] I AM DEAD
Node #9  [ 300.00000] I AM DEAD
Node #16 [ 300.00000] I AM DEAD
Node #18 [ 300.00000] I AM DEAD
Node #22 [ 300.00000] I AM DEAD
Node #27 [ 300.00000] I AM DEAD
Node #28 [ 300.00000] I AM DEAD
```

Figure 25: Node Failure Before Recovery

Figure 26 shows how a network self-recovered after multiple node failure.

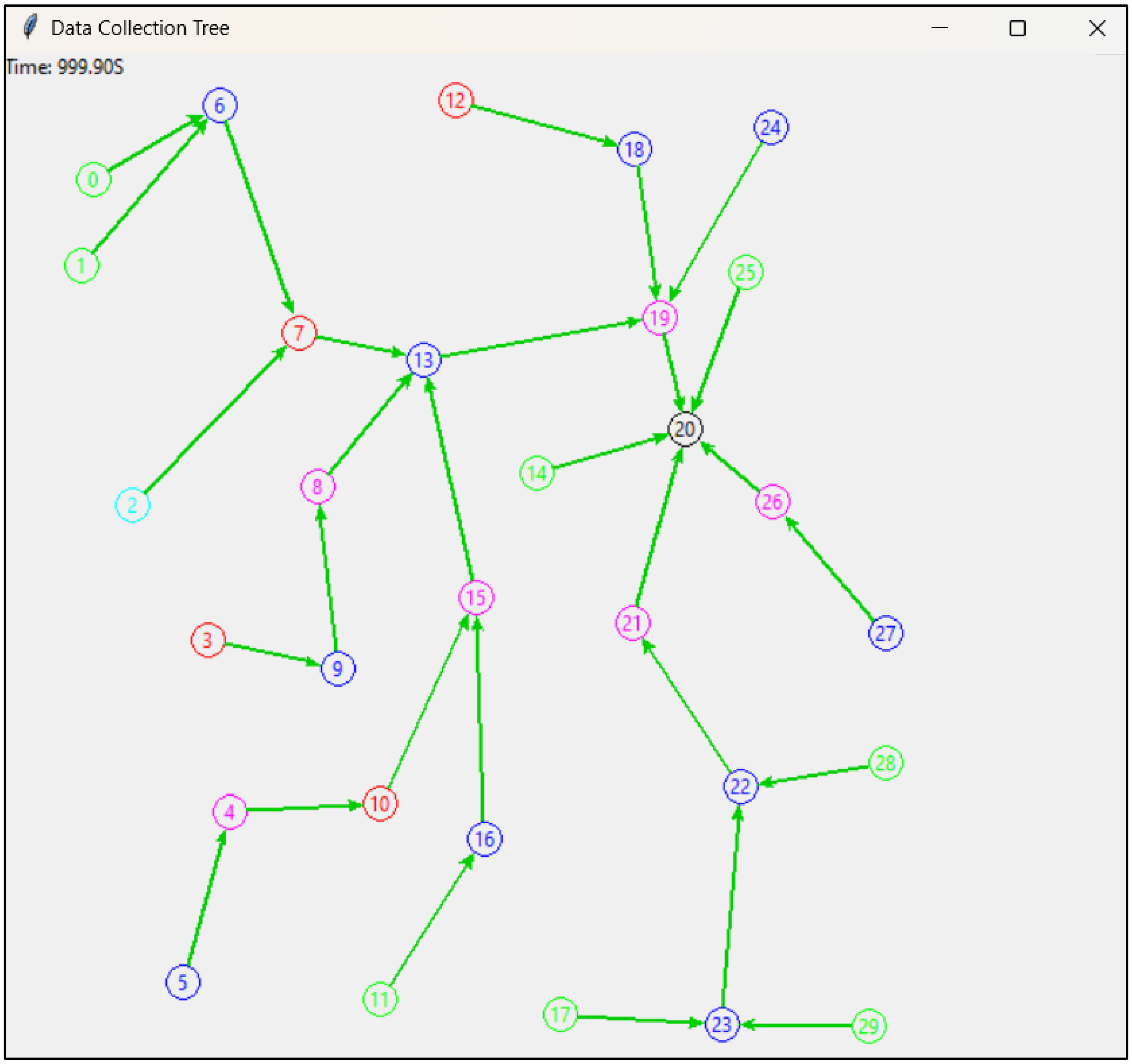


Figure 26: Repaired Network After Recovery

Name Resolution Protocol

Packet routing is done using dynamically assigned addresses, not GUIDs. GUIDs are not dynamic and are never used in sending/receiving packets but GUIDs are critical to make sure data is delivered to a correct node. Therefore, there is a need to map GUIDs to dynamic addresses to ensure packet delivery to the destination.

For example, if we have a dynamic address: 3:12, and GUID: A, and a packet is transmitted to A, A is translated to 3:12. If a packet is transmitted to 3:12, we need to make sure that 3:12 is still A because as node moves from one cluster to another, address will change. We need to make sure that when the dynamically assigned address, that is used for routing, changes, the translation from GUID to dynamically assigned address is maintained.

This issue and its solution can be associated with the Dynamic Name Server. Proper domain names registered in DNS server are translated to the IP addresses. We need a tool similar to the dynamic DNS to keep records of dynamically assigned addresses for each GUID.

One solution can be for the RootNode to keep records of changed GUID's to their associated addresses:

GUID	Dynamic Address
A	3:12, 4.2, …
B	5.11, 3.1, …

Table 15: Root's GUID Table

However, as we can see, this table can increase drastically. Thus, we need a more efficient solution.

A more rational solution can be cache. The root has a cache, a very short memory approximately of one minute. It sends a Who Is message to every ClusterHead but only the ClusterHead who has the desired GUID in its members table will respond to this message. The current dynamic address will be kept in cache no longer than one minute. After that time, the cache should be cleared because changes might happen during this time. The disrupted nods will also send a message to the root about disruption so that the cache can be cleared promptly. Therefore, this system is very similar to DNS. The RootNode will act as a name server that translates GUID to a dynamic ID.

Table 16 shows the content of the Who Is packet that is trying to search for a certain GUID by sending this message to the ClusterHeads.

6 bytes	2 bytes	1 byte	2 bytes
GUID (that it is looking for)	Source Address	Packet Type: WHO IS	Destination Address (CH)

Table 16: Who Is Packet Format

Table 17 shows the content of the HERE packet that is the response to Who Is packet. HERE contains the GUID that the RootNode is looking for, its dynamic address, packet type and Root Address as its destination.

6 bytes	2 bytes	1 byte	2 bytes
GUID (of the desired node)	Dynamic Address	Packet Type: HERE	Destination Address (Root)

Table 17: Here Packet Format

Figure 27 demonstrates the sequence diagram for exchanged messages between the RootNode and several ClusterHeads.

Similarly, in Figure 28, we can observe the ClusterHead's activity as it receives Who Is message.

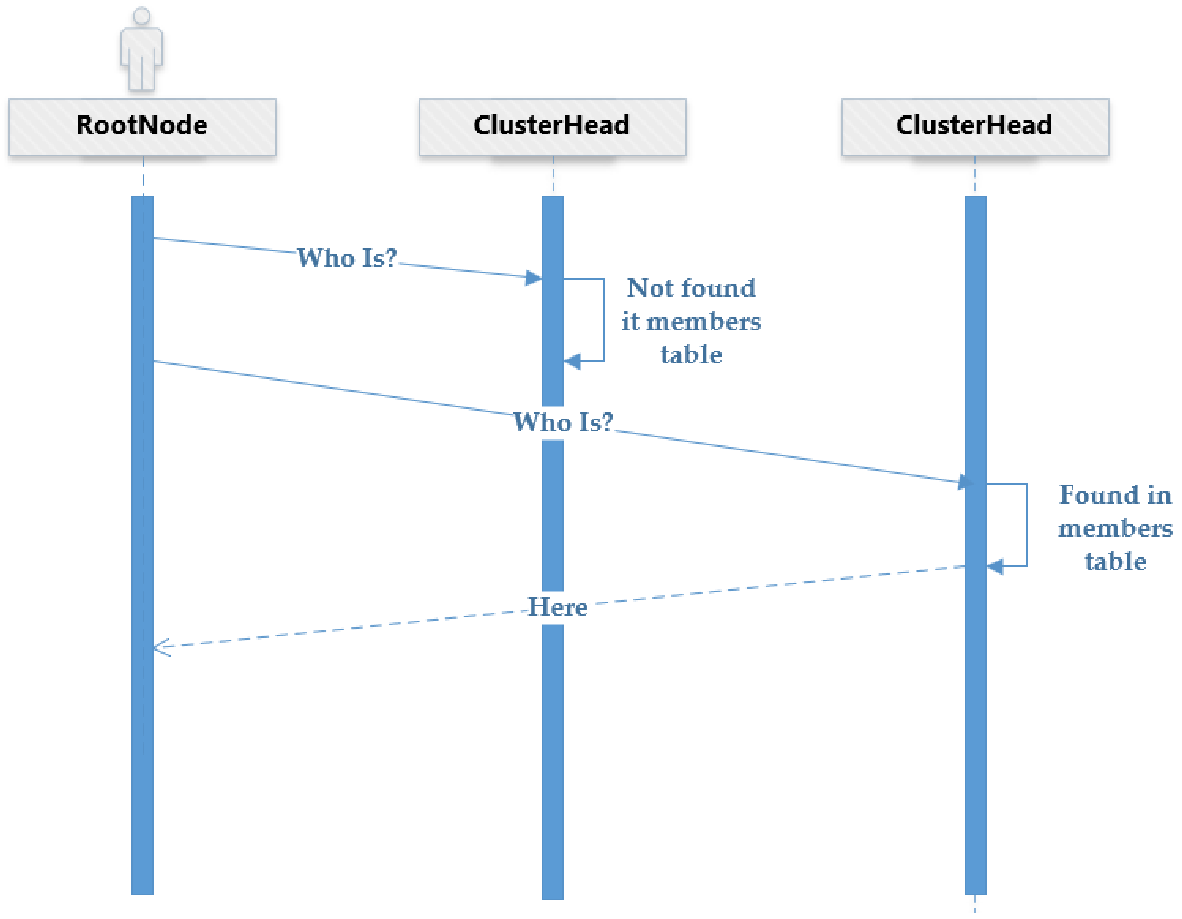


Figure 27: Sequence Diagram for Identifying GUID

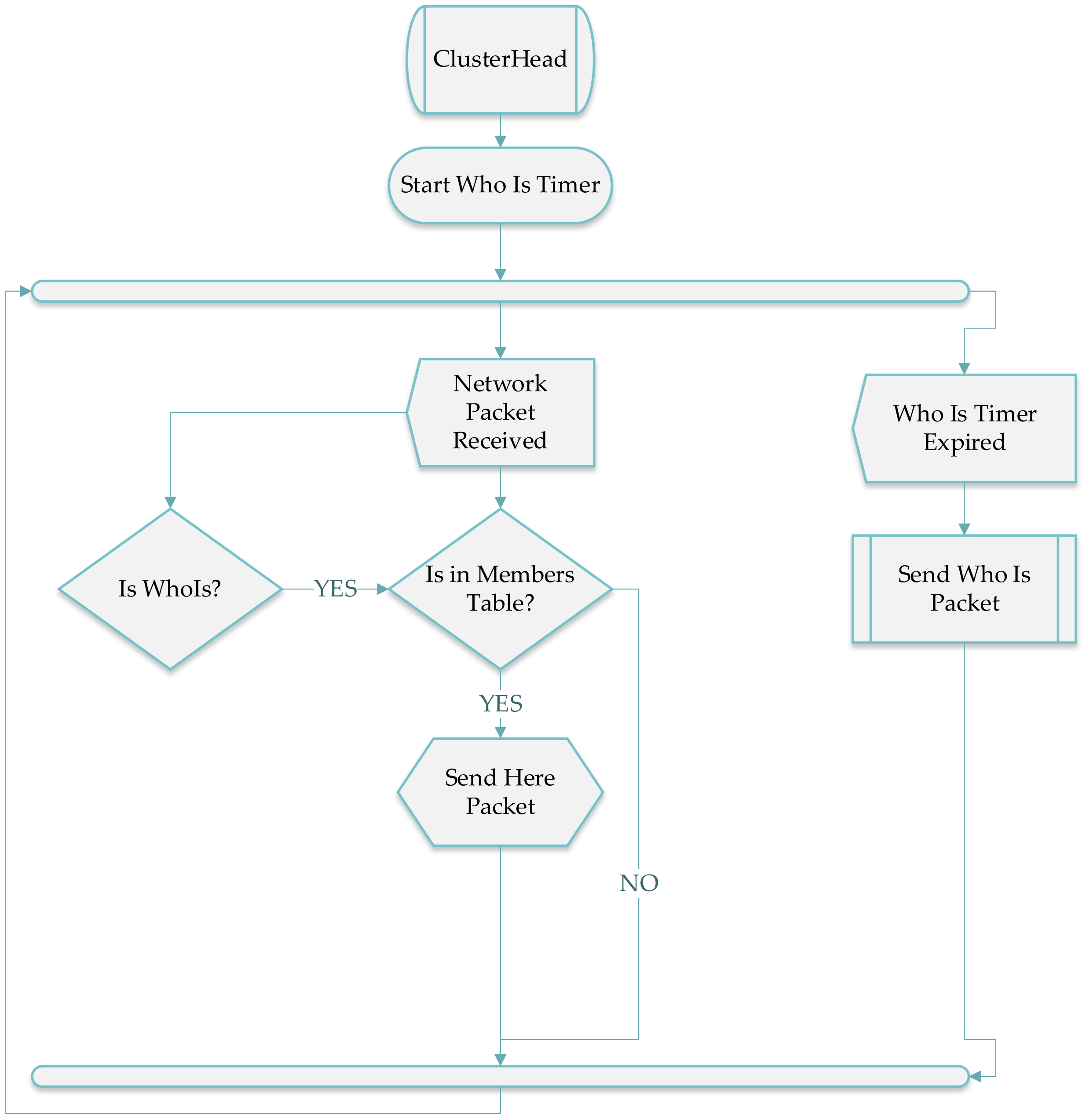


Figure 28: SDL for ClusterHead for Mapping GUID

Multicast Protocol

Multicast protocol is a receiver-oriented protocol so that the destination chooses the path from the destination to the source. This is a one-to-many communication originated from the receiver. Figure 29 illustrates multicast protocol.

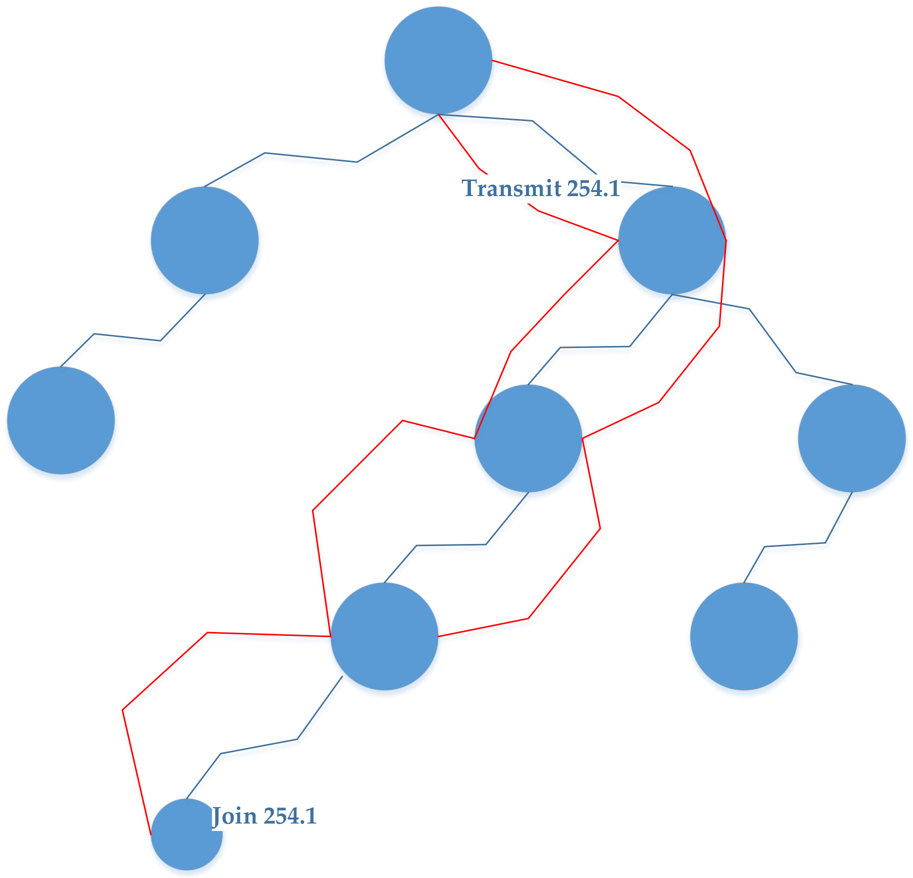


Figure 29: Multicast protocol example

The node issues a *listen* command to the parent which will have a table with multicast channel IDs that is shown in Table 18:

Multicast Channel IDs
254 : 1

Table 18: Multicast Subscribe Table Format

The RootNode has a table with 254:1; it will check if it has a customer for this address, more specifically, it will check its member's table. If it finds this address in the table, it will broadcast. The process is similar for all ClusterHeads who have a customer for a multicast packet.

Therefore, for multicast routing multicast address array will be used. Nodes will register for Channel IDs by sending requests to parent nodes. These requests will be transmitted to the gateways and all the nodes in the path will store these Channel IDs. Multicast packet on the channel will be sent from the gateways to the subscribed nodes through routers.

Table 18 represents Multicast Channel IDs that are subscribed to by the child node or the nodes that are connected to the child node.

When a multicast transmission arrives to the root, the root will repeat (broadcast) this message and this message will reach everybody so that each node will look at its Multicast table. If no such ID is found, the node will ignore the message. The node that recognizes the ID, will repeat the multicast message to its children; these nodes on their own, will also refer to their multicast lists and this process will repeat until the packet reaches the destination.

Therefore, we will have two overlapping networks:

1. Original network following unicasting protocol whose behavior is decided by source-destination pair; the message transmitted to particular nodes will reach the nodes regardless of their desire.
2. Network overlayed on top of the original network that uses additional table to decide whether the message should be delivered or not.

The protocol will require one additional packet multicast_subscribe that is given in Table 19.

6 bytes	**2 bytes**	**1 byte**	**2 bytes**
GUID	Source Address: multicast listener	Packet Type: MULTICAST SUBSCRIBE	Destination Address

Table 19: Multicast Subscribe Message Format

The Sequence diagram for the protocol is demonstrated in Figure 30.

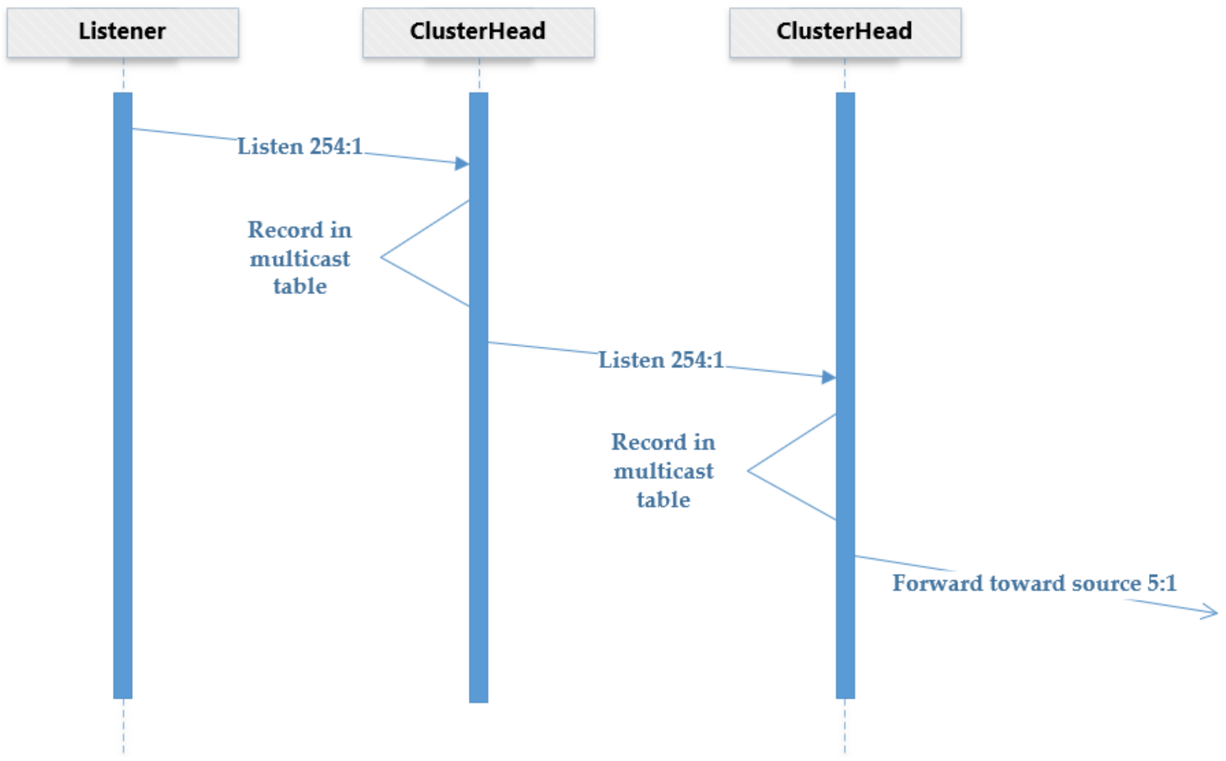
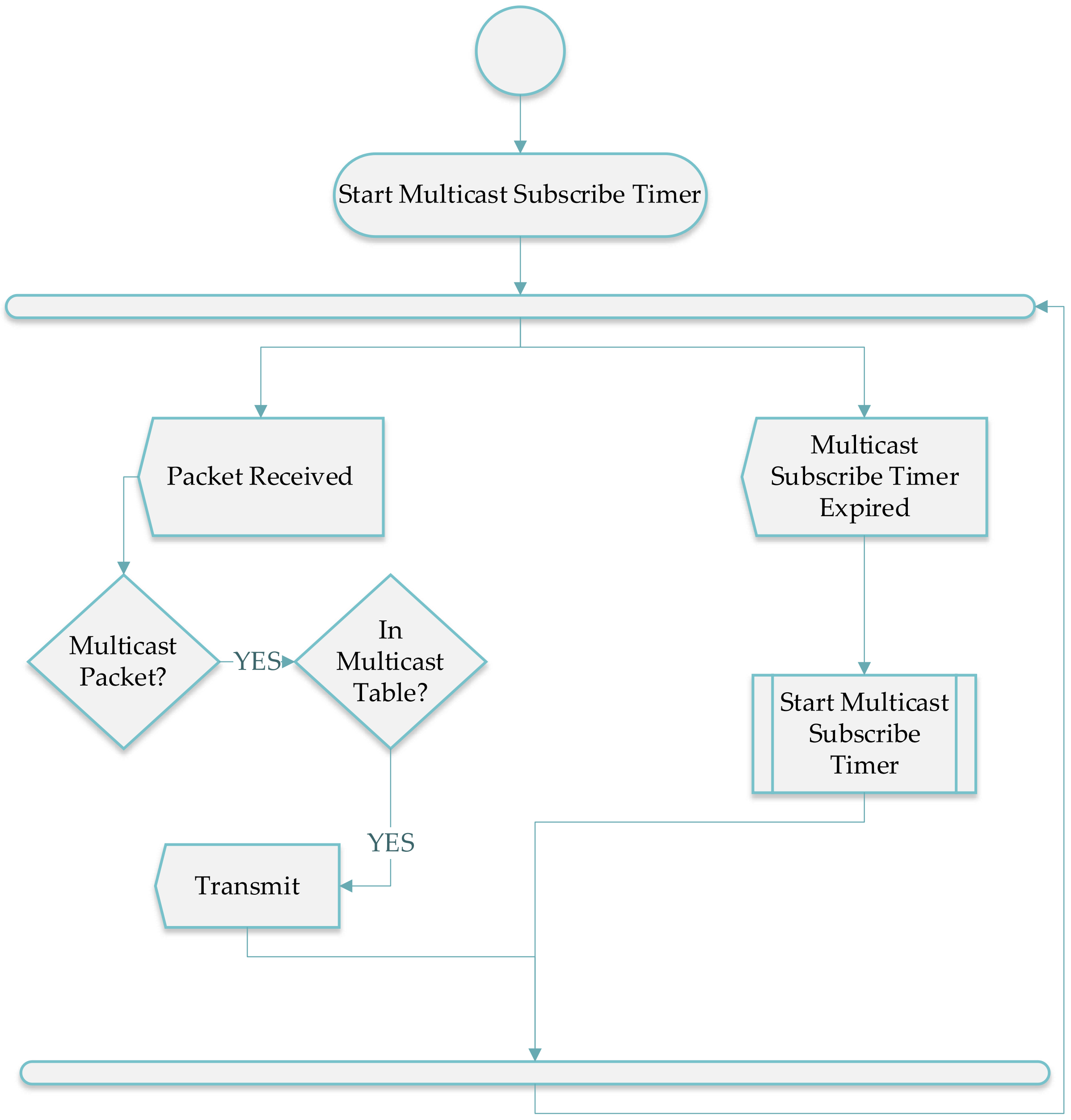

Listener
ClusterHead
ClusterHead
Listen 254:1
Record in multicast table
Listen 254:1
Record in multicast table
Forward toward source 5:1

Figure 30: Multicast Protocol Sequence Diagram

The SDL of the multicast protocol is given in Figure 31.



Start Multicast Subscribe Timer
Packet Received
Multicast Subscribe Timer Expired
Multicast Packet?
YES
In Multicast Table?
Start Multicast Subscribe Timer
YES
Transmit

Figure 31: SDL Diagram for Multicast Protocol

Figure 32 demonstrates the complete self-optimized network with multicast protocol, where node 2 is a publisher and nodes: 3, 7, 10, 12 are listeners.

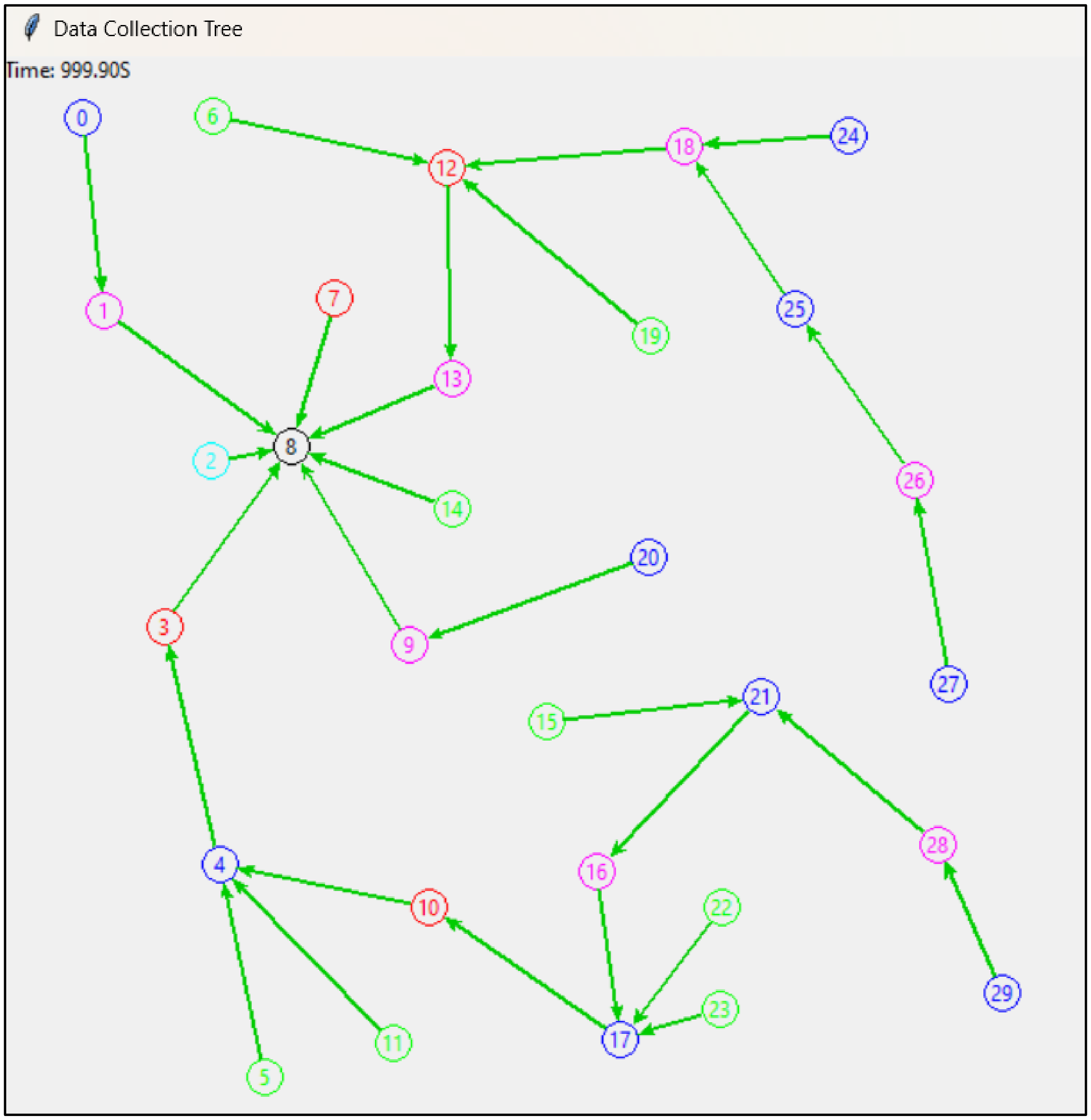
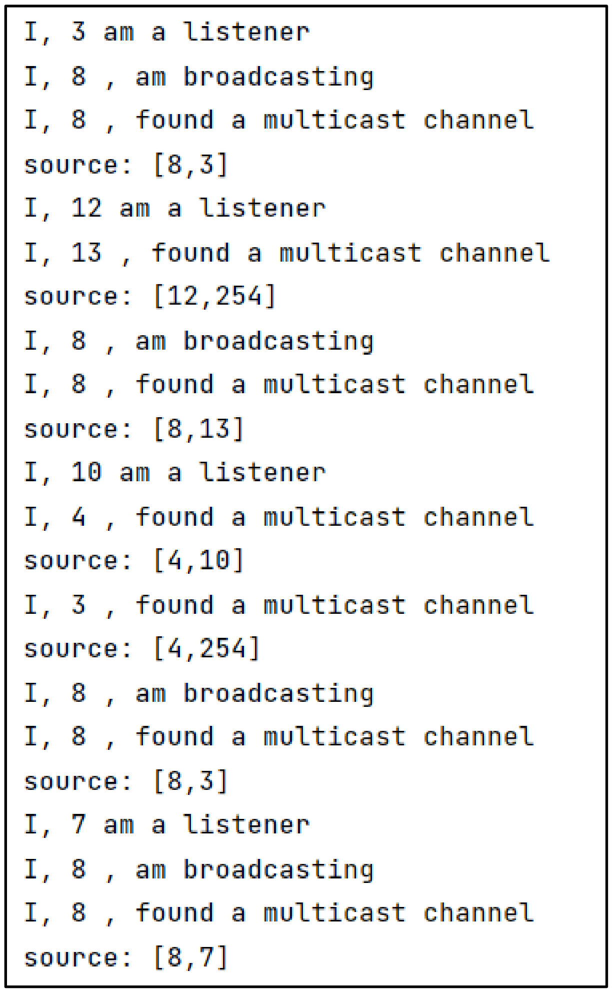


Figure 32: Complete Network

Figure 33 shows a path that a multicast packet goes through to be delivered to every destination listener.



```
I, 3 am a listener
I, 8 , am broadcasting
I, 8 , found a multicast channel
source: [8,3]
I, 12 am a listener
I, 13 , found a multicast channel
source: [12,254]
I, 8 , am broadcasting
I, 8 , found a multicast channel
source: [8,13]
I, 10 am a listener
I, 4 , found a multicast channel
source: [4,10]
I, 3 , found a multicast channel
source: [4,254]
I, 8 , am broadcasting
I, 8 , found a multicast channel
source: [8,3]
I, 7 am a listener
I, 8 , am broadcasting
I, 8 , found a multicast channel
source: [8,7]
```

Figure 33: Complete Multicast Path

Simulation using Different Parameters

At first, our renewed data collection tree was unable to function properly for increased number of nodes. The reason was the function in one of the source files: *wsnlab.py* file. We modified this function so that it would return false if network address is not found at all and returns true only when two network addresses and their node addresses are identical. This change enabled us to test the functionality of our design in various scenarios. Figures below demonstrate network after recovery.

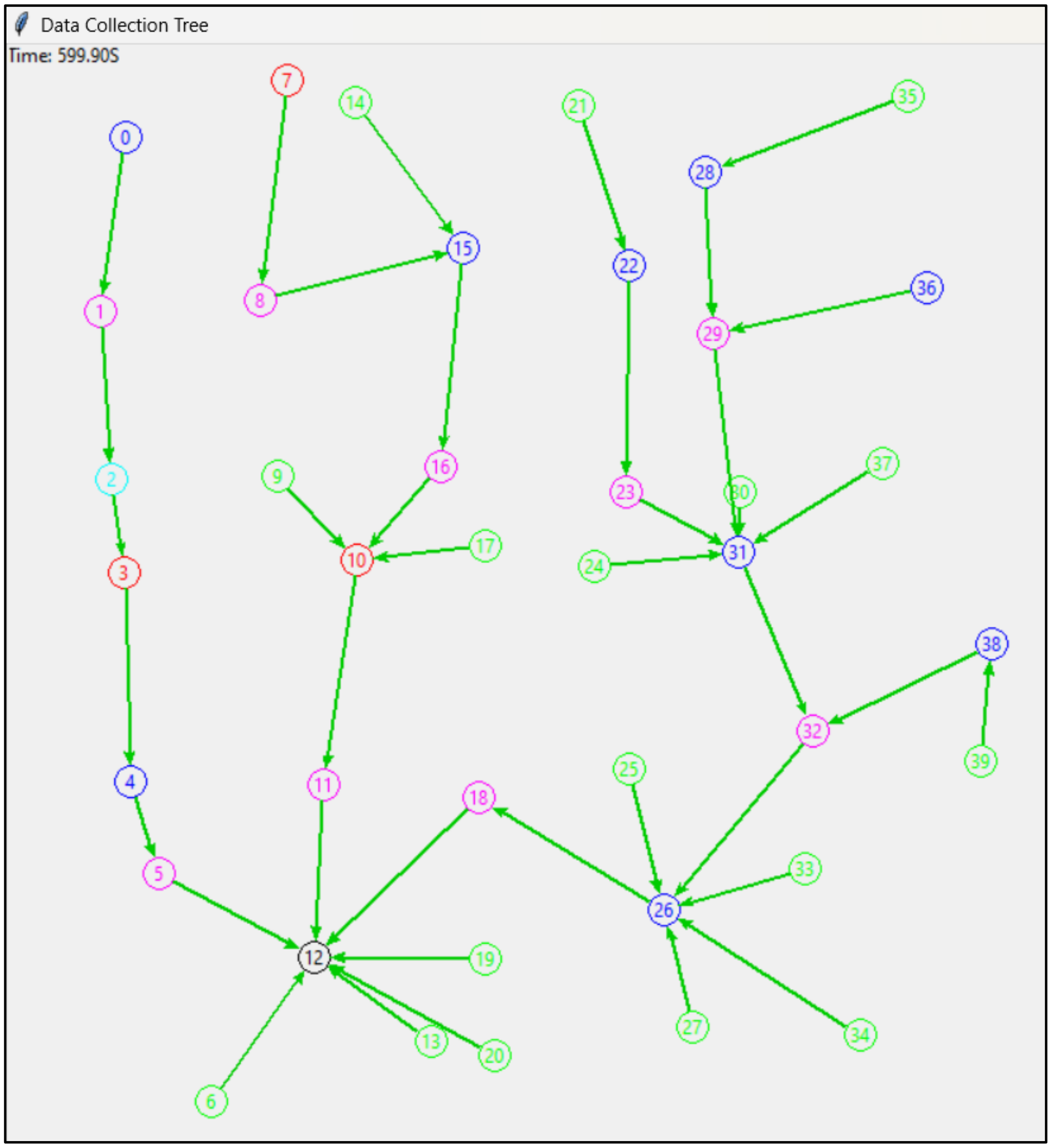


Figure 34: Simulation, *node_count* = 40

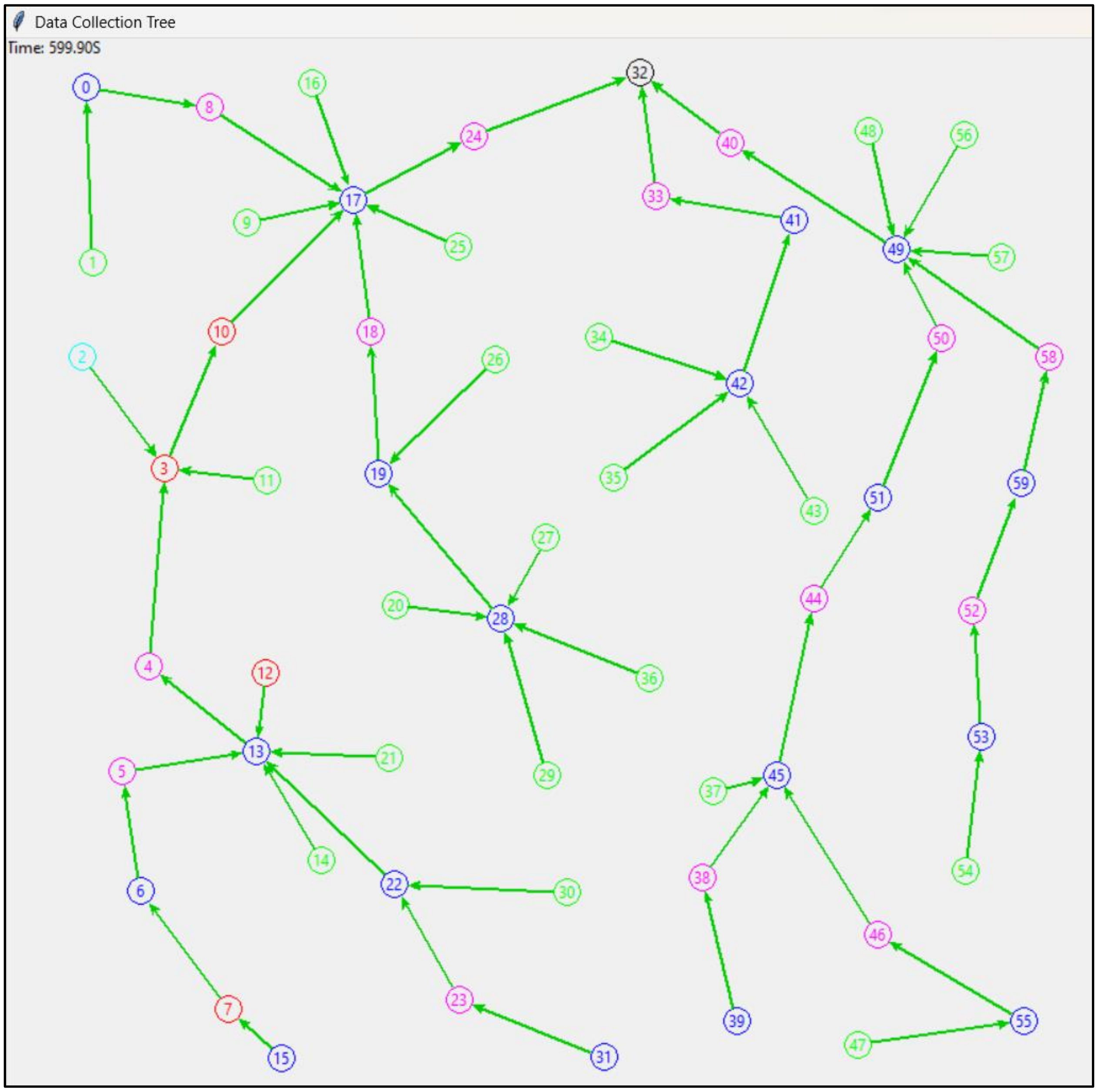


Figure 35: Simulation, *node_count* = 60

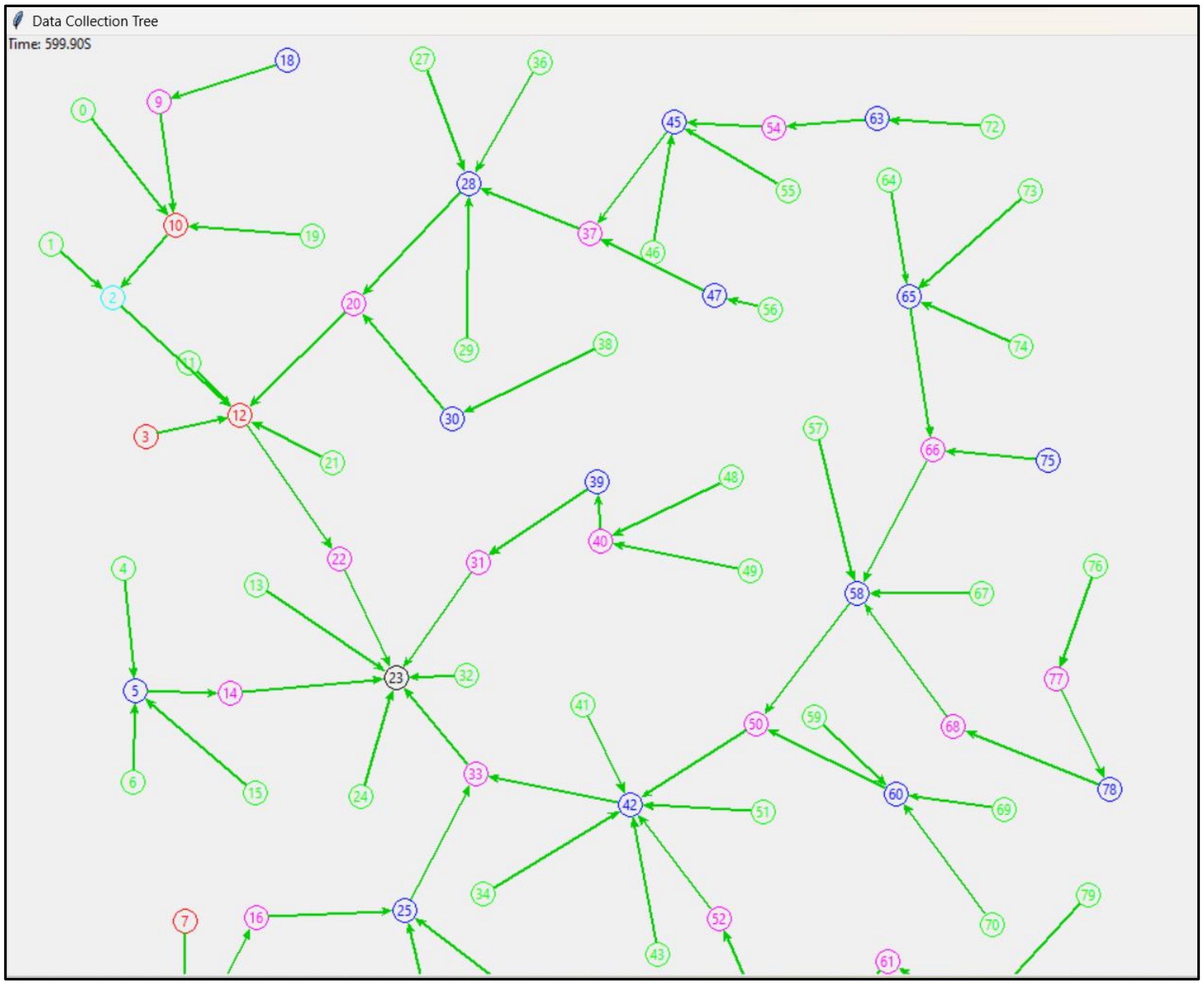


Figure 36: Simulation, *node_count* = 80

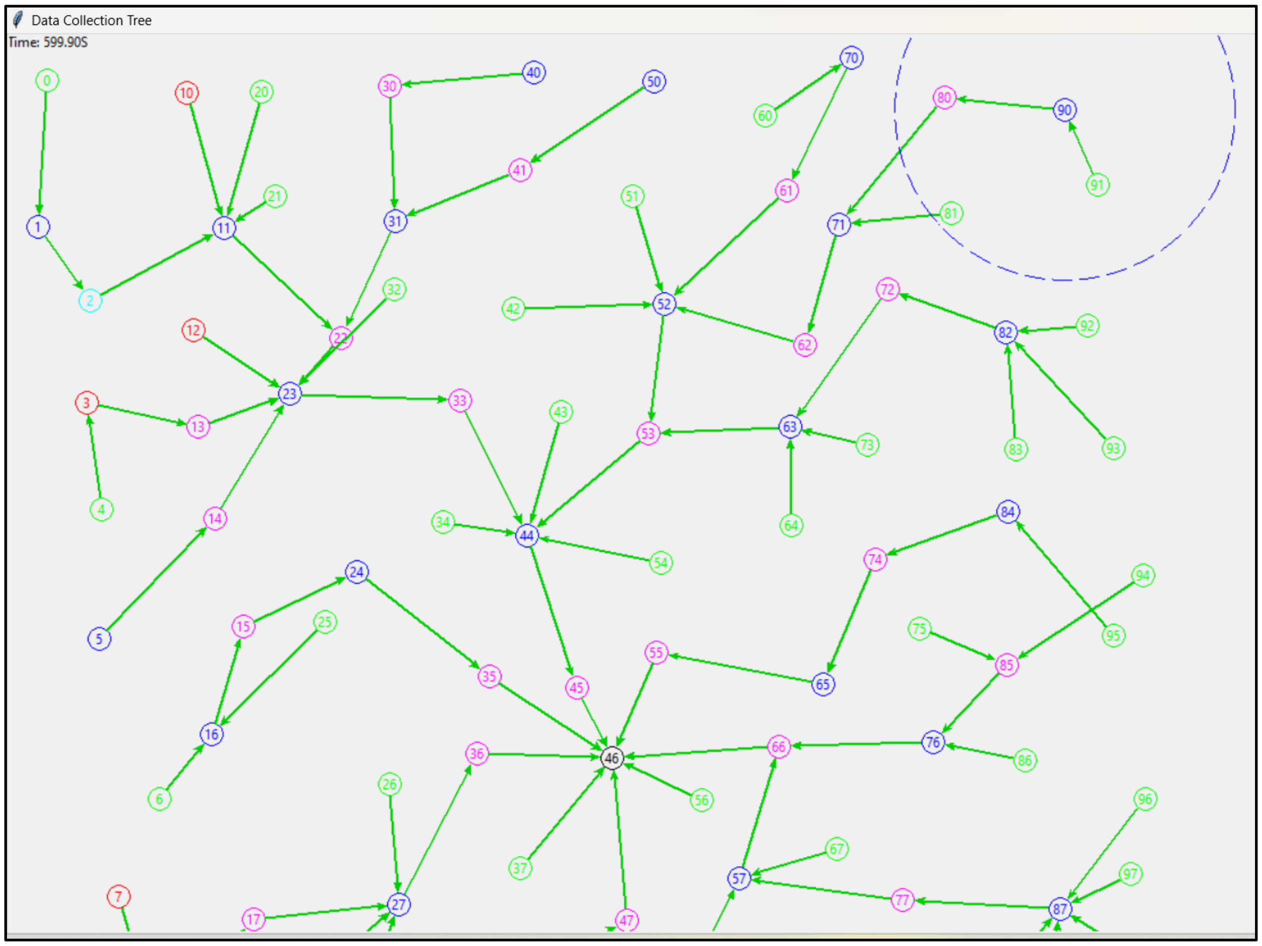


Figure 37: Simulation, *node_count* = 100

Simulation Analysis

We did an experiment to measure the exact time needed to construct the network for different number of nodes. The result of the experiment is shown in Figure 38 below.

It should be noted that nodes fail after 300-time intervals of starting the simulation. Random number of nodes may fail at random time. Therefore, the time needed to fully reconstruct the network may not be predictable.

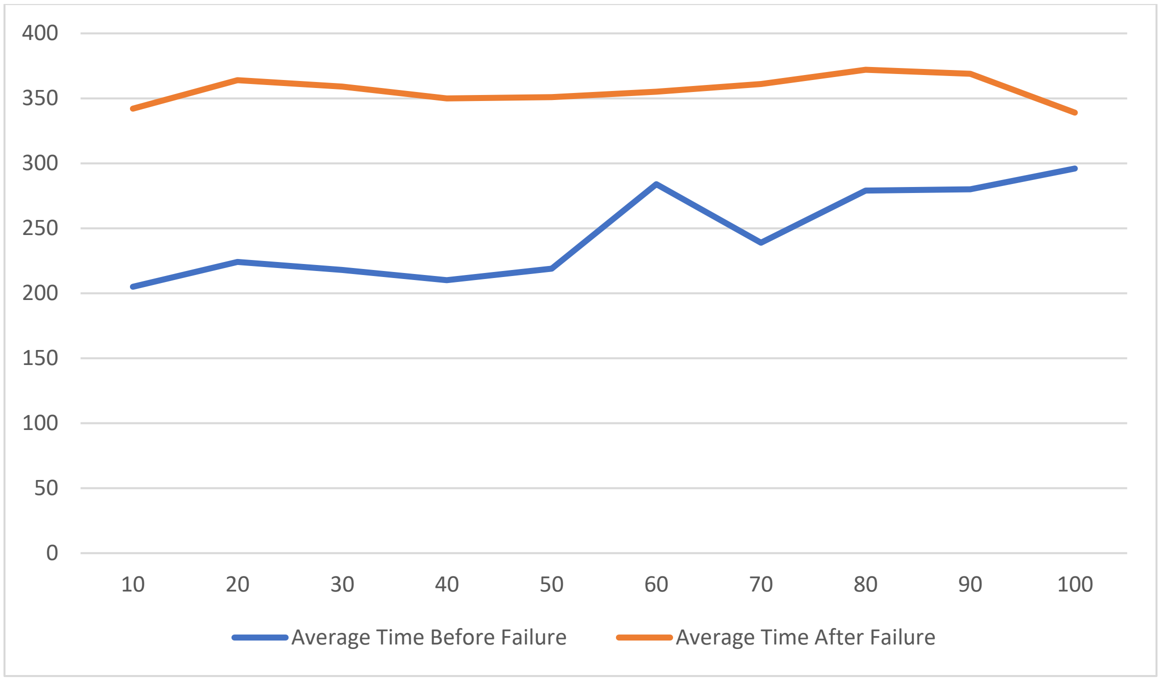


Figure 38: Time to Construct the Network

We can calculate the average time needed to join the network for each node by noting the time they joined a cluster. This measurement tool should be more precise since it averages the random results.

The time when the node first appeared is be subtracted to the previously noted time. This information is collected for all the nodes and the average is calculated using the formula:

$$\frac{Time\ when\ joined\ the\ cluster - Time\ when\ first\ appeared}{Total\ number\ of\ nodes}$$

As it can be observed, the average time is between 21 and 23 seconds. Based on the diagram, we believe that the design is efficient for the average time is approximately the same.

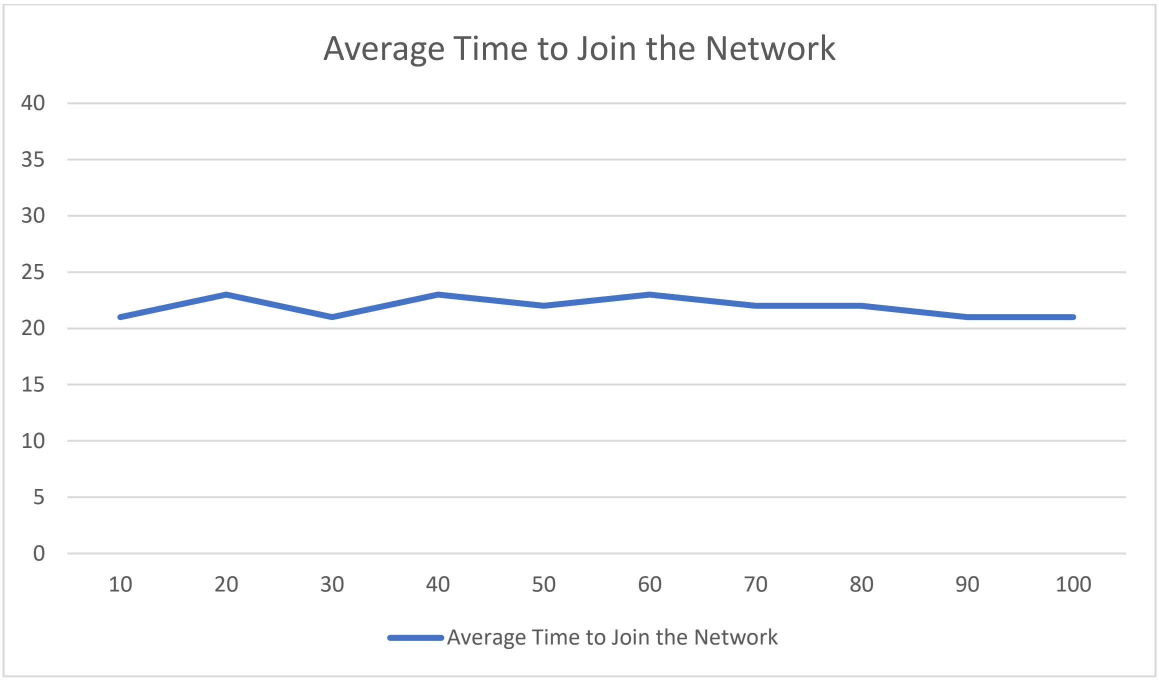


Figure 39: Average Time to Join the Network

Another tool to analyze the level of optimization of the design is to measure the average number of hops to the Root by calculating the number of hops for different simulations with number of nodes and then dividing the total number by the number of nodes:

$$\frac{\Sigma \text{ Number of hops to the root for each node}}{Total\ number\ of\ nodes}$$

As Figure 40 shows, the average number of hops increases slightly as the number of nodes increase in the simulation. We believe that this result again demonstrates correctness and effectiveness of our algorithm.

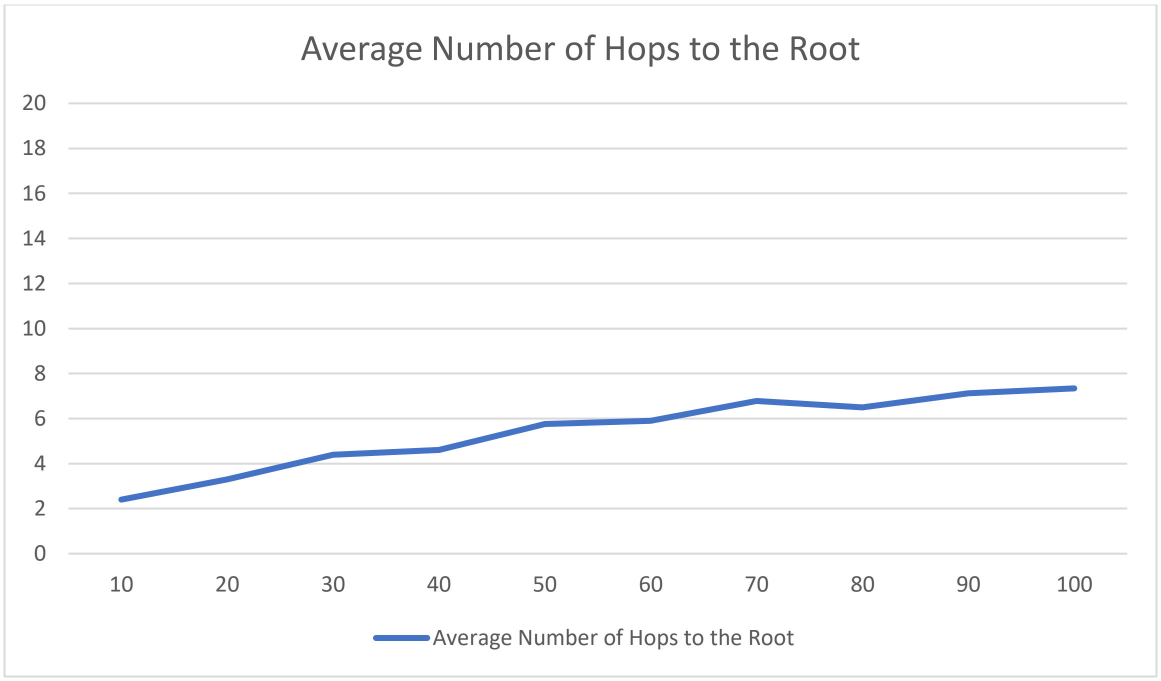


Figure 40: Average Number of Hops to the Root

Conclusions

Data collection tree constructed in this project is a very important communication topology that is used to gather multiple sensor node data. It is a hierarchical tree structure with a set of interconnected nodes. In this work, we designed and developed an efficient algorithm for the data collection tree by modifying the hierarchy of the tree, specifically, the ClusterHead role mobility. We also developed another overlay protocol on top of the tree structure, which is the multicast protocol. We suggested several interesting design principles for mapping globally unique IDs with dynamic addresses. Moreover, we evaluated the efficiency of our design and implementation by observing the key parameters, such as time and number of links, and proved that our design can be used in typical wireless sensor network applications.

Proactive system, or the table-based routing is expensive, but it reacts faster because the tables are set up ahead of time. Our suggestion for the future is to develop a hybrid design that will combine proactive and reactive protocols. This network will consider both mechanisms:

1. For immediate neighborhood (one or two-hop), it will use proactive, table-driven system.
2. Beyond certain distance, it will use the tree topology, meaning data will be transmitted level-by-level.

To identify neighbors, a neighbor discovery algorithm will be used. A good example of this can be Adaptive On-demand Distance Vector (AODV) protocol. We believe this approach will speed up routing and we will be able to combine mesh-type of network on top of the tree topology.

References

[1] V. B Nagarnaik, “Mobile Node Tethering Using Self Organized Ad-Hoc Wireless Network”, 2009.

[2] V. Richert, B. Issac, N. Israr, “Implementation of a Modified Wireless Sensor Network MAC Protocol for Critical Environments”, 2017.

[3] G. Samara, M. Aljaidi, “Aware-Routing Protocol using Best First Search Algorithm in Wireless Sensor”, 2018.