LEACH Protocol for Energy Conservation in Wireless Sensor Networks.

Connor Kirby
CPE 400
11/20/2015

Introduction

Wireless sensor networks (WSNs) are a composition of individual sensor nodes delivering information about their environment to a common base station. WSNs come in a variety of sizes and functions; however, they all share the desired interest of efficient energy consumption at the link layer. Every time a frame is sent from a sensor, energy is lost. The amount of energy lost per transmission is determined by the required distance the transmission has to travel. Unfortunately, having each node directly transmit to the base station could quickly deprive a sensor node of available power if it is located a considerable distance away from a sensor.

There exists many different protocols for creating much more efficient energy consumption within a network. This project specifically deals with the Low Energy Adaptive Clustering Hierarchy (LEACH) protocol. The protocol is simulated using python with the matplotlib libraries. Improvements will also be made on the simulated LEACH protocol to explore if beneficial optimizations are possible.

I currently work in the IOT industry. I find the ability to provide real time data over a network fascinating, more so when I can see the data happening from a web browser. Wireless sensor networks provide the functionality to do that, thus sparking my interest in doing the LEACH protocol simulation.

Contribution

LEACH is a MAC based routing protocol which aims at improving how sensor nodes utilize their energy. LEACH protocol divides data transmission into rounds. Each round is composed of 4 stages:

1. Self Election - Each sensor in the network uses a stochastic algorithm to determine whether or not it will be a sensor head for the round.
2. Advertisement - Each of the elected sensor heads broadcast a signal to all nodes that were not elected sensor heads.
3. Schedule - Each non-sensor node chooses a sensor head to transmit data to. The head is chosen based on the strongest signal received by the node from the broadcasts sent in the previous phase. The node then responds that it will be sending its data to the chosen sensor head. The sensor head is responsible for creating TDMA schedule so that each sensor will have no conflicts over the medium while transmitting its' data to the head.
4. Steady State - Based on the schedule, sensors send their data to the predetermined head. The sensor head compresses all received data from the sensors and then finally transmits the data to the base station.

This simulation also hopes to improve on LEACH by offering optimization within the self election phase. Nodes will be aware of their relative energy versus all the other nodes. A threshold will be set so that a node below the relative average will not be chosen as a sensor head.

Instructions/Code Explanation

The code should be able to run on the CSE Ubuntu machines as is. I tested it at *ubuntu.cse.unr.edu*.

To run with improvements:

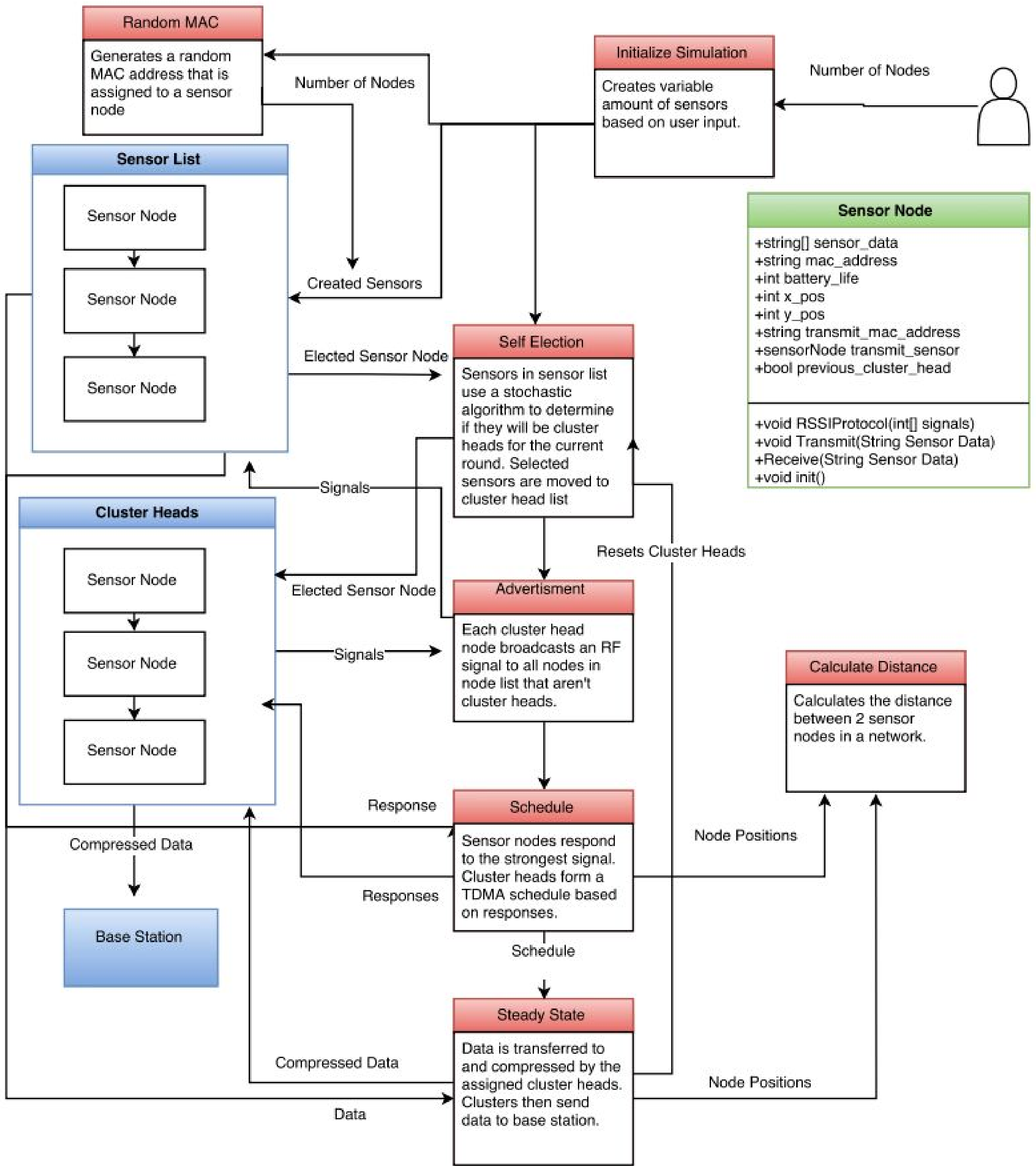
```
python LEACH.py -I
```

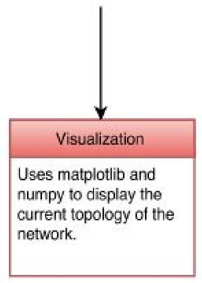
To run normally:

```
python LEACH.py -N
```

Flags must be **SPECIFIED**

Upon seeing the display window for the given round, close the window to continue the simulation.


Random MAC
Generates a random MAC address that is assigned to a sensor node
Initialize Simulation
Creates variable amount of sensors based on user input.
Number of Nodes
Number of Nodes
Sensor List
Sensor Node
Sensor Node
Sensor Node
Created Sensors
Sensor Node
+string[] sensor_data
+string mac_address
+int battery_life
+int x_pos
+int y_pos
+string transmit_mac_address
+sensorNode transmit_sensor
+bool previous_cluster_head
+void RSSIProtocol(int[] signals)
+void Transmit(String Sensor Data)
+Receive(String Sensor Data)
+void init()
Elected Sensor Node
Self Election
Sensors in sensor list use a stochastic algorithm to determine if they will be cluster heads for the current round. Selected sensors are moved to cluster head list
Signals
Cluster Heads
Sensor Node
Sensor Node
Sensor Node
Resets Cluster Heads
Elected Sensor Node
Advertisment
Each cluster head node broadcasts an RF signal to all nodes in node list that aren't cluster heads.
Signals
Calculate Distance
Calculates the distance between 2 sensor nodes in a network.
Response
Schedule
Sensor nodes respond to the strongest signal. Cluster heads form a TDMA schedule based on responses.
Node Positions
Compressed Data
Responses
Base Station
Schedule
Steady State
Data is transferred to and compressed by the assigned cluster heads. Clusters then send data to base station.
Compressed Data
Node Positions
Data


Visualization
Uses matplotlib and numpy to display the current topology of the network.

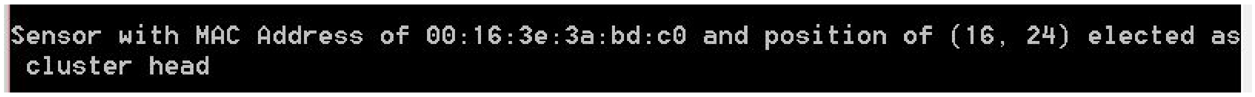
This simulation seeds the random generator with the integer 8. Doing so allows the same random integers to be used again and again within the simulation.

The simulation begins by prompting a user to enter a set amount of nodes for the network.

Sensor nodes are then randomly generated across a 100x100 meter grid. The position, along with a randomly generated MAC address, are stored within the Sensor Node class. The Sensor Nodes are then stored into the global data structure sensor list. Sensor List is repeatedly used by the main functions of the simulation.

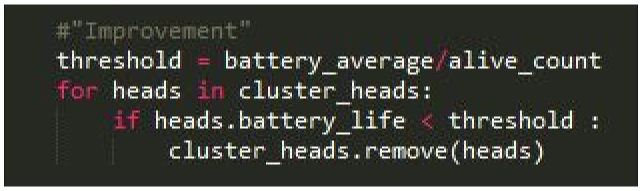
After initializing the simulation, we enter the main loop of the program. The main loop is composed of four main functions that mimic the four stages of the leach protocol and one function that displays the current configuration of the network.

Self Election - Nodes use a stochastic algorithm to determine if they will be a cluster head for the current round. To simulate this, the simulation generates a random number one to 100 and compares it to a preset probability. If the generated number is within the probability, the node becomes a cluster head for the round. Nodes also check to make sure that they were not previously a cluster head in the previous rounds. Upon election, a message is broadcasted in the terminal.



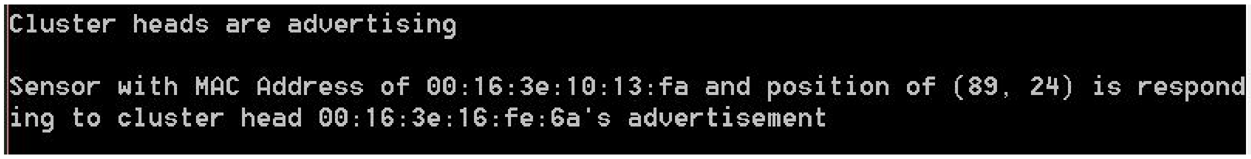
```
Sensor with MAC Address of 00:16:3e:3a:bd:c0 and position of (16, 24) elected as
 cluster head
```

Self Election Improved- Same as self election noted above, except that elected cluster nodes will compare their energy to the average energy of the network. If it is below the average minimum, it will be excluded as a cluster head.



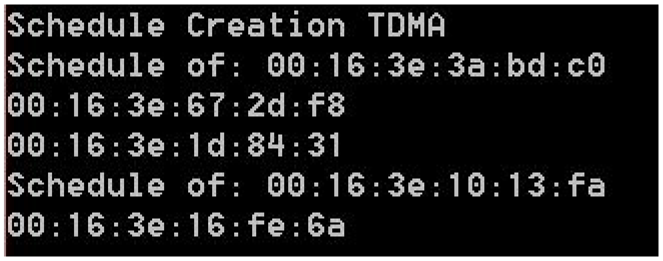
```
#"Improvement"
threshold = battery_average/alive_count
for heads in cluster_heads:
    if heads.battery_life < threshold :
        cluster_heads.remove(heads)
```

Advertisement- Cluster heads broadcast signals to all sensor nodes. Sensor Nodes determine which of the signals is strongest. To simulate this, we apply Calculate Distance between the cluster head and the sensor node. The signals are actually just the non-cluster head nodes comparing distances between each cluster head. In our simulation, the closest node would have the strongest signal. After finding the closest cluster head, a response is then sent from the sensor node to said cluster head. The cluster head MAC address is also stored in the sensor node for the round. We are assuming that all nodes are reachable from any one node to another within our network.



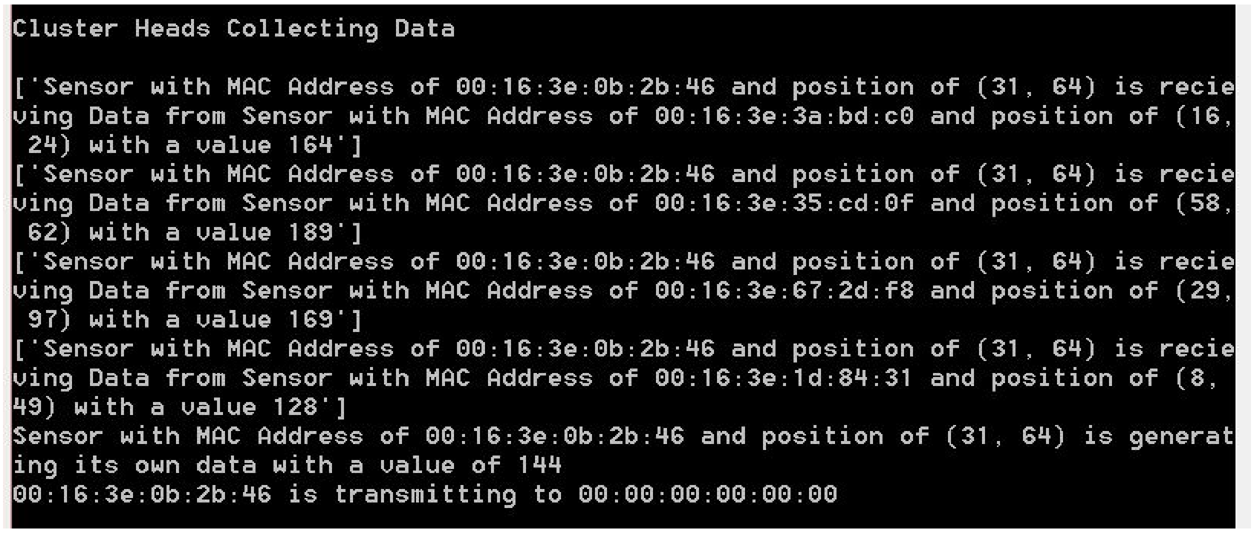
```
Cluster heads are advertising

Sensor with MAC Address of 00:16:3e:10:13:fa and position of (89, 24) is respond
ing to cluster head 00:16:3e:16:fe:6a's advertisement
```

Schedule - Cluster heads receive responses and create a TDMA schedule of when sensors will broadcast to the cluster head based on the stored MAC address in each sensor.



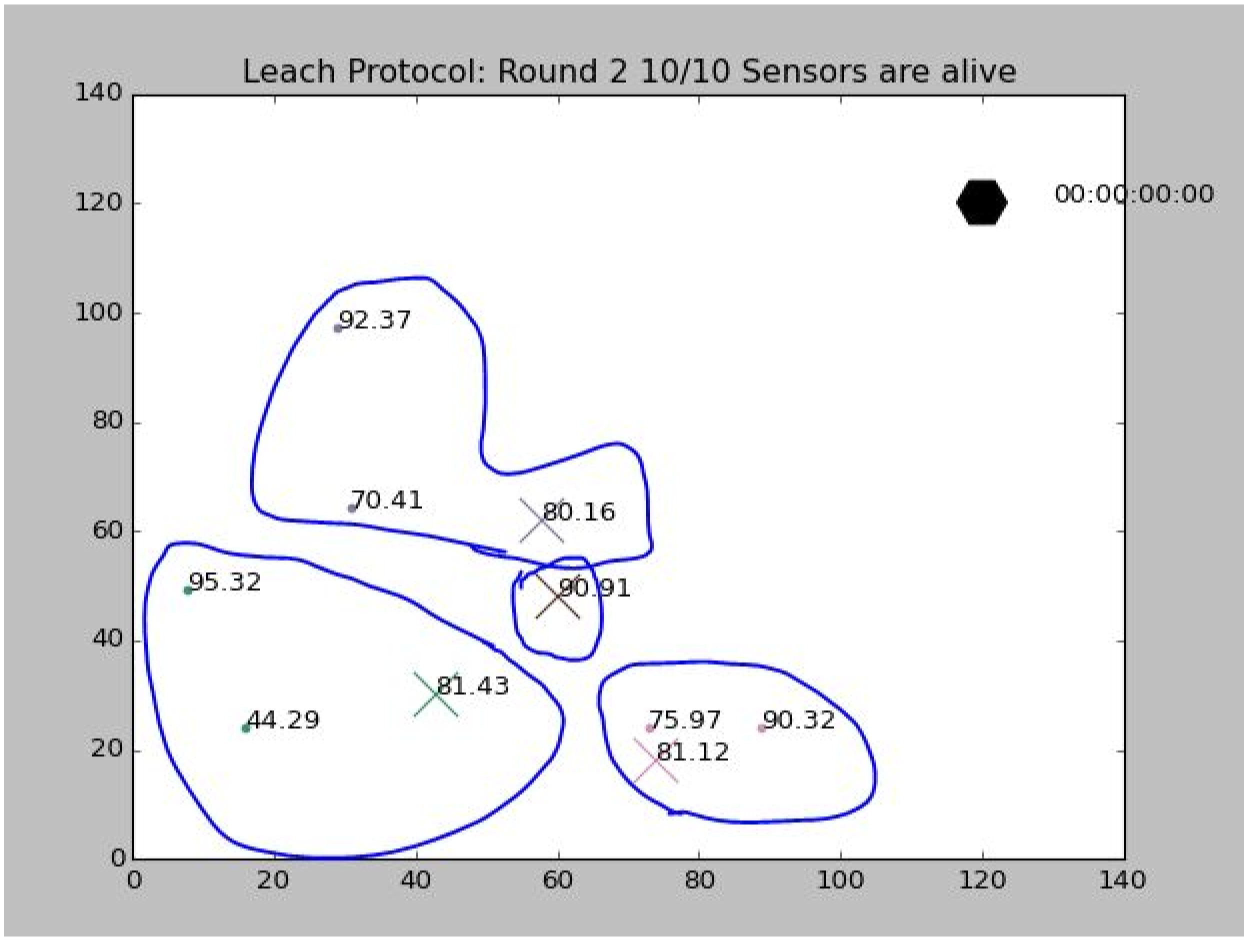
```
Schedule Creation TDMA
Schedule of: 00:16:3e:3a:bd:c0
00:16:3e:67:2d:f8
00:16:3e:1d:84:31
Schedule of: 00:16:3e:10:13:fa
00:16:3e:16:fe:6a
```

Steady State - Data is sent from the sensors to the cluster heads. Cluster heads compress that data and then send it to the base station. In our simulation, the base station has a location of (130, 120). Data is a randomly generated number that is appended to a cluster head's data list. Cluster heads also generate their own random data. Anytime data is transferred in the network, the sensor node will have its energy decremented. We are assuming that energy is decremented by *distance to send * data size * arbitrary scalar to speed up or slow down the simulation*. Once a sensor's energy is below 0, it is disregarded from the continuing calculations.



```
Cluster Heads Collecting Data

['Sensor with MAC Address of 00:16:3e:0b:2b:46 and position of (31, 64) is recie
ving Data from Sensor with MAC Address of 00:16:3e:3a:bd:c0 and position of (16,
 24) with a value 164']
['Sensor with MAC Address of 00:16:3e:0b:2b:46 and position of (31, 64) is recie
ving Data from Sensor with MAC Address of 00:16:3e:35:cd:0f and position of (58,
 62) with a value 189']
['Sensor with MAC Address of 00:16:3e:0b:2b:46 and position of (31, 64) is recie
ving Data from Sensor with MAC Address of 00:16:3e:67:2d:f8 and position of (29,
 97) with a value 169']
['Sensor with MAC Address of 00:16:3e:0b:2b:46 and position of (31, 64) is recie
ving Data from Sensor with MAC Address of 00:16:3e:1d:84:31 and position of (8,
49) with a value 128']
Sensor with MAC Address of 00:16:3e:0b:2b:46 and position of (31, 64) is generat
ing its own data with a value of 144
00:16:3e:0b:2b:46 is transmitting to 00:00:00:00:00:00
```

Visualization- Using matplotlib, a user can see the network configuration before each new round. X's represent the cluster heads, the dots represent sensor nodes, and the black pentagon is the base station. Clusters are color coded as to see what nodes transmit to what base station. Current battery life is also displayed next to each node. *Note: When less nodes appear on screen, the graph has a tendency to change its perspective. All positions stay the same, the graphs perspective may just be shifted since it does not need to display as far left as before.*



The simulation will continue to execute the four main loops until all the sensors in the network are dead.

Results

# Nodes	Self Election	Self Election Improved
10	23	19
20	27	23
30	30	26
40	38	35

Unfortunately, my improvement was not much of an improvement and caused the network to last fewer rounds. My results lean towards the fact that it is better to allow the stochastic algorithm to determine cluster heads each round and only that algorithm. I believe this is because at first, many nodes do not become cluster heads because of the energy requirement threshold. This creates less cluster heads which increases the burden on the current heads causing them to lose energy faster.

Conclusion

I was unable to find any optimizations by using average network powers as a variable when selecting cluster heads. To continue, I would begin looking at the cases where a cluster head has no give nodes that send data to it. This can occur because of bad positioning. It would be optimal to have cluster heads that only have their data to send, send it to the next closest sensor head.