CNG 491, Graduation Project

Simulation and comparison of clustering algorithms for WSNs

Literature Survey and GUI Presentations Report

Instructor: Prof. Enver Ever

Introduction

A wireless sensor network (WSN) is composed of spatially distributed sensor nodes and a Base Station (BS). These nodes take measurements such as humidity, temperature, pressure, light etc. They process these measurements and transmit them to the base station where all the data is collected and all the decisions are made [4].

Given the difficulty associated with the deployment of the nodes, and the fact that the nodes use non-renewable energy source, prolonging the life of the WSN is one of the main concerns that had been studied extensively. Many protocols have been proposed in order to prolong the network lifetime, one of these protocols is clustering which one of the most efficient protocols. The network is divided into clusters, each cluster has a cluster head (CH) and a number of nodes that only communicate with their CH (intra-cluster communication). Cluster heads communicate with each other and/or with the BS (inter-cluster communication) [5].

There are two types of clustering, equal size clustering and unequal size clustering. When equal size clustering is used, a problem known as the hot spot problem arises, where the nodes closer to the BS dies earlier due to the heavy inter-cluster communication [2]. HEED and LEACH are examples of such protocols. Unequal clustering protocols, on the other hand, try to solve this problem by reducing the size of the clusters which are close to the BS, in order to reduce the intra-cluster communication and reserve energy resources to handle the inter-cluster communication, UHEED and RUHEED are examples of such protocols.

In this project, we will try to optimize the UHEED algorithm in order to further prolong the lifetime of the network then we will compare it with the other previously mentioned protocols using simulation to validate that it actually outperforms them. Therefore, the main aim is to find the optimal cluster size in order to solve the hot spot problem and prolong the network lifetime.

Literature Review

Many equal and unequal size clustering protocols had been proposed in order to prolong the life time of wireless sensor networks, in this section we describe some work on both types of clustering protocols. We will describe briefly how each protocol works and address its advantages and drawbacks.

Low Energy Adaptive Clustering Hierarchy (LEACH) is an application specific hierarchical clustering and cluster head selection protocol, aiming for randomized but efficient election and rotation of cluster heads (CH), aggregation/compression and localized coordination and control for cluster set-ups in order to reduce overall energy dissipation in Wireless Sensor Networks (WSNs)[6].

The actual design of LEACH introduces a two layered hierarchy, nodes communicating with CHs and CHs communicating with base station. There is also a suggestion of a multi layered hierarchy, a model where CHs behave similar to nodes and cluster between themselves to reduce network traffic and energy dissipation [6][8].

Original LEACH is consisted of two phases. In the first phase, "set-up phase", CHs are selected and clusters are formed. For CH selection, LEACH uses a stochastic method, calculating different thresholds for each node for each round and promoting a node if its generated random number is smaller than this threshold. Exclusion of candidacy of the recent CHs is achieved using:

$$(1) \qquad T(n)=\frac{P}{\left(1-P*\left(r \bmod \frac{1}{P}\right)\right)} \qquad if\ n \in G$$

$$T(n)=0 \qquad if\ n \notin G$$

where P is pre-calculated CH selection probability, G is the set of nodes that was not selected as CH for last 1/P rounds and T(n) is the threshold. Each selected CH has the responsibility to advertise his role to the nodes using CSMA[8]. After $\frac{1}{P} - 1$ rounds, each node becomes eligible to be selected as CH again. This rotation ensures a fair depletion of residual energy for each node [8].

In the second and last phase, "steady state phase", nodes forward their data to Base Station (BS) over their determined CHs using TDMA protocol. Cluster heads are responsible for collecting, aggregating, compressing the data from nodes within the cluster and sending new packets to BS [6].

LEACH had achieved superior overall network life-time and energy efficiency with random CH rotation within constraints:

- Every node has capability to transmit to BS
- Every node has enough computational power to run simple instructions and signal processing
- Every node is capable of using MAC protocols
- Each node has low level information for end user, but while there is a high correlation between data from neighbor nodes, redundancy is high and data can be approximated, aggregated, compressed or reduced to achieve "effective data"[6]

Due to randomness of CH selection method, there can be more or less number of clusters than desired amount [8]. Based on this problem, an improvement of LEACH, LEACH Deterministic Cluster Head Selection (LEACH-DCHS) was proposed by the Handy et al. in 2002. Randomness was reduced by using the residual energy as parameter in selection method and guarantees a fixed count of CHs and clusters among the lifetime of the network.

Besides the parameter inclusion, LEACH- DCHS uses real time advertisements for CH competition, increasing the probability of selecting nodes with higher potential as CH. As can be seen in two different versions of the improved T(n) calculation:

$$(2) \qquad T(n)=\frac{P}{\left(1-P\times\left(r \bmod \frac{1}{P}\right)\right)}\times\frac{E_{n\_current}}{E_{n\_max}}$$

$$(3) \qquad T(n)=\frac{P}{\left(1-P\times\left(r \bmod \frac{1}{P}\right)\right)}\times\left[\frac{E_{n\_current}}{E_{n\_max}}+\left(r_s \operatorname{div} \frac{1}{P}\right)\left(1-\frac{E_{n\_current}}{E_{n\_max}}\right)\right]$$

current residual energy of the node and initial energy proportion is included as multiplicand. LEACH- DCHS also pointed to another flaw of LEACH, quality of transmission and advised usage of direct-sequence spread spectrum, which has a disadvantage in means of time wasted in set-up phase [7].

Another approach to similar problems was introduced by the original researchers of the LEACH, also in 2002. The paper proposed calculating probabilistic value p each round, individually as following equation:

$$(4) \qquad P_i(t) = min\left\{\frac{E_i(t)}{E_{total}(t)}k, 1\right\}$$

$$(5) \qquad E[\#CH] = \sum_{i=1}^{N} P_i(t) * 1 = \left(\frac{E_i(t)}{E_{total}} + \ldots + \frac{E_N(t)}{E_{total}}\right)k = k$$

where N is total number of nodes, Pi(t) is new individual probability for node i, E[#CH] is the estimated count of CHs in the network and it is denoted as k in equations. These N and k values have to be programmed in each node a priori. To avoid that pre-programming, an approximation to (4) can be used, if initial energy levels are equal for all nodes and average energy of the system assumed to be average residual energy of a node:

$$(6) \qquad P(n) = \frac{k}{N - k * \left(r \bmod \frac{N}{k}\right)} \quad : \quad C_i(t) = 0$$

$$P(n) = 0 \quad : \quad C_i(t) = 0$$

With (6), local decision making mechanism can be achieved, if pre-programming or continuous advertising is not possible [8].

For further improvements, [8] employs a new version of LEACH, named LEACH centralized (LEACH-C), presents new control mechanism to ensure precise number of CHs and better node/cluster distribution, using extra positioning sensors, preferably GPS, and BS as controller. Since this problem is NP-hard, usage of simulated annealing algorithm on BS is advised [8].

Another novel approach to overcome same randomness of LEACH algorithm was proposed by Tong et al. in 2010. LEACH-Balanced (LEACH-B) presents a new, two layered CH selection algorithm, built on the equation which is developed by Heinzelman et al.[8] and tries to fix CH/node ratio to 3 – 5%, which interval is claimed to be optimal for these applications.

At the first phase, every candidate announces themselves as prospective cluster heads. After this advertisement period, system tries to match CH/node ratio to desired value. If there are excessive prospective CHs, first $^{node * ratio}$ heads are assigned as heads. Else, if there are less than expected candidates, BS advertises every node to start their timers. These timers are inverse proportional to their residual energy, thus higher the residual energy node has, shorter its timer is. Each node advertises when their timers finish and BS assigns these first required amount of nodes as new CHs [9].

A different study held by Xu et al. in 2012, argues about problems in fixed round times and comes with completely different equations than the previous researches mentioned above. This perspective assumes thresholds for becoming CH have to be calculated dynamically and introduce their equations, which is calculated every round, locally:

$$P_{head} = \sqrt{\frac{N}{2\pi}}^{*} \sqrt{\frac{E_{fs}}{E_{mp}}}^{*} \frac{M}{d^{2}_{\text{to BS}^{*}} N} \quad (7)$$

This Phead is advised to be used in (2) to obtain more energy concerned results. Also, the round time calculation model is shown in closed form:

$$T_{curr} = NF_{avg}\left(M_{min^{*}} \sigma + \lambda\right) \quad (8)$$

"Where NFavg is the average number of frames for a cluster with size 1/Phead. Mmin*σ+λ is the frame time of a cluster which has the minimum size"[10].

Hybrid Energy Efficient Distributed (HEED) is a distributed, energy efficient clustering approach for ad hoc sensor networks. Its main goal is to prolong network lifetime. It uses two clustering parameters; the first one is the residual energy of each node as primary clustering parameter, in order to select a cluster head. This parameter is estimated rather than calculated. The second parameter is intra-cluster "communication cost", which can be a function of neighbour proximity or cluster density.

There are three main stages of the algorithm, initializing, iterative and final. In the initializing stage, each node is assigned a probability of becoming a tentative cluster head (*CHprob*). In the iterative stage, neighbor tentative cluster heads compete in order to become cluster heads, where the node that has the best of residual energy, node degree and node proximity is selected. In the final stage, each node either joins the nearest cluster head or announces itself to be cluster head when there is no cluster head in its range.

HEED produces equal clusters since the size is independent of the distance between the cluster head and the base station. The problem which arises from using HEED is the hot spot problem. As the sensor nodes closer to the base station deplete their energy faster than distant sensor nodes because of the high inter-cluster traffic, which in turn reduces the overall network lifetime [1].

Unequal Hybrid Energy Efficient Distributed (UHEED), is an unequal clustering protocol that uses HEED to select cluster heads and calculate the competition radius, to form unequal clusters, using the formula defined by EEUC (An Energy Efficient Unequal Clustering).

$$R_{comp} = (1 - c(\frac{(dmax\ -\ d(S_i, BS)}{dmax\ -\ dmin}))R^{\circ}_{comp}$$

The parameter $R^{\circ}_{comp}$is a fixed value equals themaximum transmission radius of a sensor node. C is a constant between 0 and 1.$d_{max}$and$d_{min}$are the maximum and minimum distances of sensor nodes from the BS, d(S_i, BS) is the distance of the sensorS_i from the base station.

UHEED produces unequal clusters, the further the cluster is from the base station, the greater the radius is. This way, the inter-cluster communication can be reduced for the clusters closer to the base station. The simulation results show that UHEEDoutperforms HEED in terms of the network lifetimewhether we consider half nodes dead or the first node dead [2].

RUHEED (Rotated Unequal Hybrid Energy Efficient Distributed) protocol is an improvement overUHEED. At the first round, it works like UHEED, but after the cluster formation, the rotation stage starts where the protocol selects one of the cluster nodes as a cluster head based on it residual energy. This stage lasts until one of the WSN nodes completely depletes its energy. After this, this base station will inform all the nodes and the protocol will start from the beginning and elect new cluster heads using HEED and form the clusters using UHEED.

According to the simulation results, RUHEED outperforms both HEED and UHEEDwhether we consider half nodes dead or the first node dead. This comes from the fact that RUHEED reduces the number of control messages coming from cluster head election and cluster formation [3].

Problem Area, Rationale & Objectives

This research is based on [2], hence shares the same concerns and attempts to present a new set of parameters to define optimum cluster sizes related to their distances to BS.

To produce optimal cluster sizes in means of radius, this study focuses on optimizing the parameters of uHEED clustering algorithm with mathematical optimization methods in order to introduce a closed form model and proving superiority of uHEED in given scenarios by simulation. Considering the physical constraints of wireless sensors, this paper aims to increase overall lifetime of the network, in means of FND, HNA, LND, quality of network and adaptivity to different scenarios.

Research Method

Closed form mathematical methods will be used to optimize the UHEED algorithm. Then simulation will be used to validate whether that the new optimized algorithm actually outperforms the HEED, LEACH algorithms

Scrum Details

The duration of each sprint throughout the entire project will be 4 weeks, however the planning day and sprint review days will not be taken into account when planning the sprint. Therefore, each sprint will have 18 development days.

During each sprint, there will be biweekly meetings where the group members will meet and discuss the developments so far and how much has been accomplished of each task.

The product backlog:

Story ID	Story name	Status	Size	Sprint	Comments
1	Study the related publications and write the literature review.	Ongoing	10	1	Study the publications, summarize them and write literature review. Also, formulate the problem statement, set the goals of the study and indicate the methodology to be used.
2	Simulate the LEACH, HEED and UHEED algorithms using the simulation program.	Ongoing	60	1	
3	Implement the LEACH, HEED and UHEED algorithms.	Planned	20	2	Implement each of the previously mentioned algorithms using the studied publications description.
4	Optimize the UHEED algorithm.	Planned	20	3	
5	Compare the network-lifetime of all the previous algorithms.	Planned	10	3	

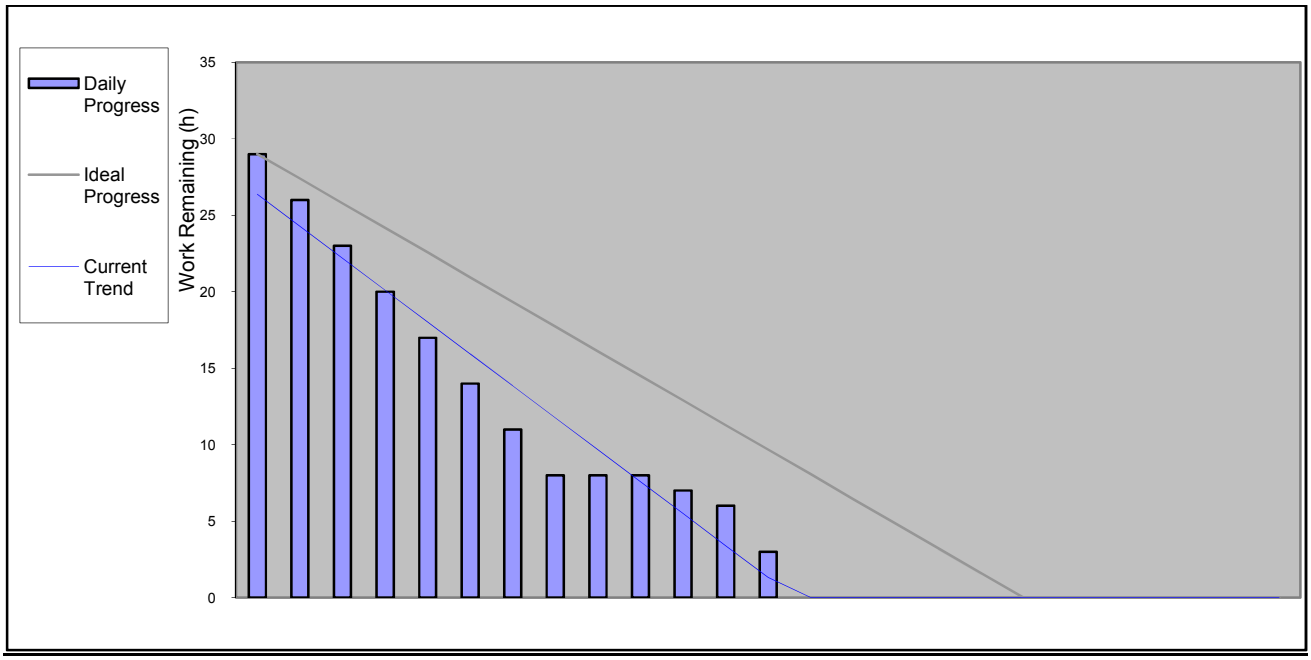
The sprint backlog:

Task name	Task ID
Search for publications related to the project, then sort and classify them based on their relevance to the project.	1
Get familiar with the different WSN simulator and choose one of them to use for the reminder of the project.	2
Practice using the chosen simulator.	3
Study the publication related to the LEACH algorithm.	4
Study the publication related to the HEED and UHEED algorithms.	5

Sprint implementation days	18		
Trend calculated based on last	18	Days	Totals
Task ID	Story ID	Responsible	Status
1	1	Ates,Yassin,Abdullah	Done
2	2	Abdullah	Done
3	2	Abdullah	Done
4	1	Ates	Done
5	1	Yassin	Done

		Effort	Remaining on implementation day…																	
Days	Totals	29	29	26	23	20	17	14	11	8	8	8	7	6	3					
Task ID	Status	Est.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	Done	9	9	6	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Done	3	3	3	3	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	Done	7	7	7	7	7	7	7	6	4	4	4	3	3	1	0	0	0	0	0
4	Done	5	5	5	5	5	4	3	2	2	2	2	2	2	1	0	0	0	0	0
5	Done	5	5	5	5	5	4	3	3	2	2	2	2	1	1	0	0	0	0	0

The sprint burndown chart:



The sprint details:

The sprint started off by working on task 1 which is searching through the internet looking for publications related to the project. Then, the papers found were then classified based on their topic and level of relevance to the project. All the team members were involved in this task which was completed by the next meeting.

After that, Yassin was assigned to study the papers related to HEED and UHEED algorithms and to prepare the related literature study. On the other hand, Ates was assigned the papers related to LEACH algorithm and all of its different variations. While Abdullah was studying surveys related to Wireless Sensor Networks Simulators.

Later on, OMNET++ was chosen to be the simulator which will be used in this project, so Abdullah started working on task 3 by trying to implement simple wireless sensor networks using the previously mentioned simulator.

Estimation

Function Point Estimation

Inputs: There will two inputs, the first is the number of wireless sensor nodes, while the second is a list of entire set of wireless sensor nodes with all of their relevant information like; the location of the node, the residual energy of the node, node ID, etc.

Outputs: the system will generate three outputs, two in the form of text files; the first files will contain all of the nodes with their status at the end of the simulation run. The second file will contain a detailed description of the entire set of packets that were generated by the network of nodes, the file will contain information regarding which node generated that packet, which path did the packet follow until it reached the base-station. As for the third output, it will be in the form of a graph diagram comparing the network-lifetime of all three different algorithms used.

Files: There will be three different files; the first will contain the initial set of nodes before the simulation starts. The second file will contain the nodes and their status after the simulation have ended, as for the third file it contains the list of all the packets that were generated and routed through the network.

To estimate the function points and the adjusted total function points the following website was used:
http://groups.engin.umd.umich.edu/CIS/course.des/cis525/js/f00/harvey/FP_Calc.html

Screenshots of the results are below:

Domain Characteristic Table

MEASUREMENT PARAMETER	COUNT (value >= 0)	WEIGHTING FACTOR Simple	Average	Complex
Number of User Input	2	○	○	◉
Number of User Outputs	3	○	○	◉
Number of User Inquiries	0	◉	○	○
Number of Files	3	○	○	◉
Number of External Interfaces	0	◉	○	○

Complexity Adjustment Table | FP Calculation

Complexity Adjustment Table

ITEM	COMPLEXITY ADJUSTMENT QUESTIONS	SCALE No Influence 0	1	2	3	4	Essential 5
1	Does the system require reliable backup and recovery?	○	◉	○	○	○	○
2	Are data communications required?	○	○	○	○	◉	○
3	Are there distributed processing functions?	○	◉	○	○	○	○
4	Is performance critical?	○	○	○	○	○	◉
5	Will the system run in an existing, heavily utilized operational environment?	○	○	○	○	○	◉
6	Does the system require on-line data entry?	○	◉	○	○	○	○
7	Does the on-line data entry require the input transaction to be built over multiple screens or operations?	○	◉	○	○	○	○
8	Are the master files updated on-line?	○	◉	○	○	○	○
9	Are the inputs, outputs, files or inquiries complex?	○	○	○	○	○	◉
10	Is the internal processing complex?	○	○	○	○	○	◉
11	Is the code to be designed reusable?	○	○	○	○	○	◉
12	Are conversion and installation included in the design?	○	◉	○	○	○	○
13	Is the system designed for multiple installations in different organizations?	○	○	○	○	○	◉
14	Is the application designed to facilitate change and ease of use by the user?	○	◉	○	○	○	○

Domain Characteristic Table | FP Calculation

RESULT	
PROJECT FUNCTION POINTS	82.68

Top of Page | Domain Characteristic Table | Complexity Adjustment Table

LOC SIZE METRIC:

The language to be used during development is C++.

LOC= ATFP * Language Unit Size

Since the language to be used is C++ , therefore language mode unit size is 55.

Mode: LOC = 82.68 * 55 =4547.4

And language minimum unit size is 40.

Minimum: LOC = 82.68 * 40=3307.2

Also, language maximum unit size is 140.

Maximum: LOC = 82.68 * 140=11575.2

Constructive Cost Model COCOMO Basic:

Development mode: organic.

a=2.4, b=1.05, c=0.38.

Number of thousand delivered source instructions= (KDSI) = ATFP* Language unit size/1000

= 4547.4/1000=4.5474.

Effort in staff months =effort in man-moths=MM=$a*KDSI^b$

$= 2.4 * (4.5474^{1.05}) = 11.7723335 \approx 12$

Since team-size is three: 12/3 = 4 Months.

The development time=TDEV=$2.5*MM^{c}$

$=2.5* 4^{0.38} =4.23372 \approx 5$ Months.

Simulation Demonstration

The simulator that will be used to validate the findings of our project is OMNET++ v4.6. Therefore, in order to get familiar with the simulation IDE, we implemented three small projects that should help get us more acquainted with the environment, its capabilities and how to use these capabilities efficiently.

Project 1:

In the first project, we simply built a network of two nodes that will send the same message between each other back-and-forth. First, the node module is built which is simply made out of 2 channels; one for input and one for output as follows:

```
package lab2;

//
// TODO auto-generated module
//
simple Node
{

    gates:
        input in;
        output out;
}
```

After that, the network module is built by placing two nodes next to each other and connecting them together, the code to implement that is:

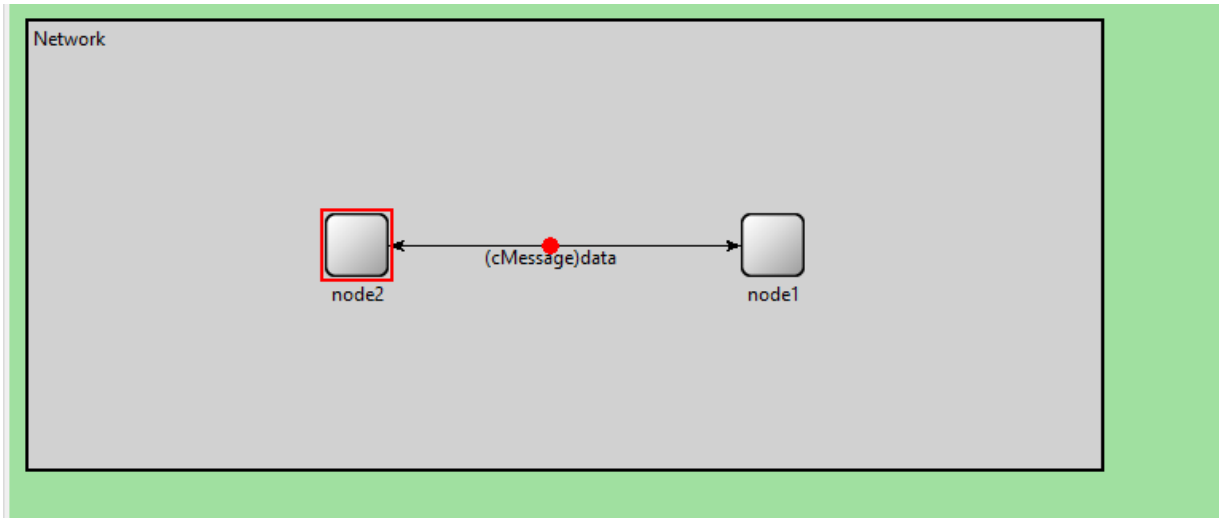
```
package lab2;

network Network
{
    submodules:
        node2: Node {
            @display("p=206,140");
        }
        node1: Node {
            @display("p=467,140;is=vl");
        }
    connections:
        node2.out --> node1.in;
        node1.out --> node2.in;
}
```

Then, the behavior of the node module should be implemented in order to determine how it will handle received messages and created messages.

```
#include "Node.h"

Define_Module(Node);

void Node::initialize()
{
    // TODO - Generated method body
    cMessage * reminder = new cMessage("Interval");
    double tx_interval=0.1;
    scheduleAt(simTime() + tx_interval,reminder);

}

void Node::handleMessage(cMessage *msg)
{
    // TODO - Generated method body
    if(msg->isSelfMessage())
    {

        cMessage * data= new cMessage("data");
        send(data,"out");


        double tx_interval=0.1;
        cMessage * reminder = new cMessage("Interval");

        scheduleAt(simTime() + tx_interval,reminder);


    }



}
```

The method handleMessage will determine the behaviour of the node when it receives a message. The method will check whether this is an internal message i.e. a self-generated timer. Then, it will create a packet called "data" and send it to the other node and set the internal timer again after a certain amount of time determined by the parameter tx_interval. The scheduleAt function will be used to set the internal timer at a given time both when the node is initially created and every time the handleMessage method is called.

After implementing all of the previously mentioned modules and methods, the project is ready to run; simulating the project will give the following results:



Event#	Time	Src/Dest	Name	Info
#1	0.1	node2 --> node1	data	id=2 kind=0
#2	0.1	node1 --> node2	data	id=5 kind=0
#5	0.2	node2 --> node1	data	id=8 kind=0
#6	0.2	node1 --> node2	data	id=11 kind=0
#9	0.3	node2 --> node1	data	id=14 kind=0
#10	0.3	node1 --> node2	data	id=17 kind=0
#13	0.4	node2 --> node1	data	id=20 kind=0
#14	0.4	node1 --> node2	data	id=23 kind=0
#17	0.5	node2 --> node1	data	id=26 kind=0
#18	0.5	node1 --> node2	data	id=29 kind=0
#21	0.6	node2 --> node1	data	id=32 kind=0
#22	0.6	node1 --> node2	data	id=35 kind=0

Project 2:

In this project a group of sensing nodes is implemented where a packet of information is created and sent from a predetermined node to a predetermined sensing node, however the path which the packet will take to get to the target node is random where at each node, the next node is randomly selected until it reaches the target node.

First, the node module should be implemented where in this project each node will have multiple incoming channels and multiple outgoing channels. The code to implement this node will be as follows:

```
simple Node
{
    @display("i=device/pc");

    gates:
    input in[];
    output out[];

}
```

The next step is to build a network out of a set of these nodes. The connections are predetermined between these nodes as follows:

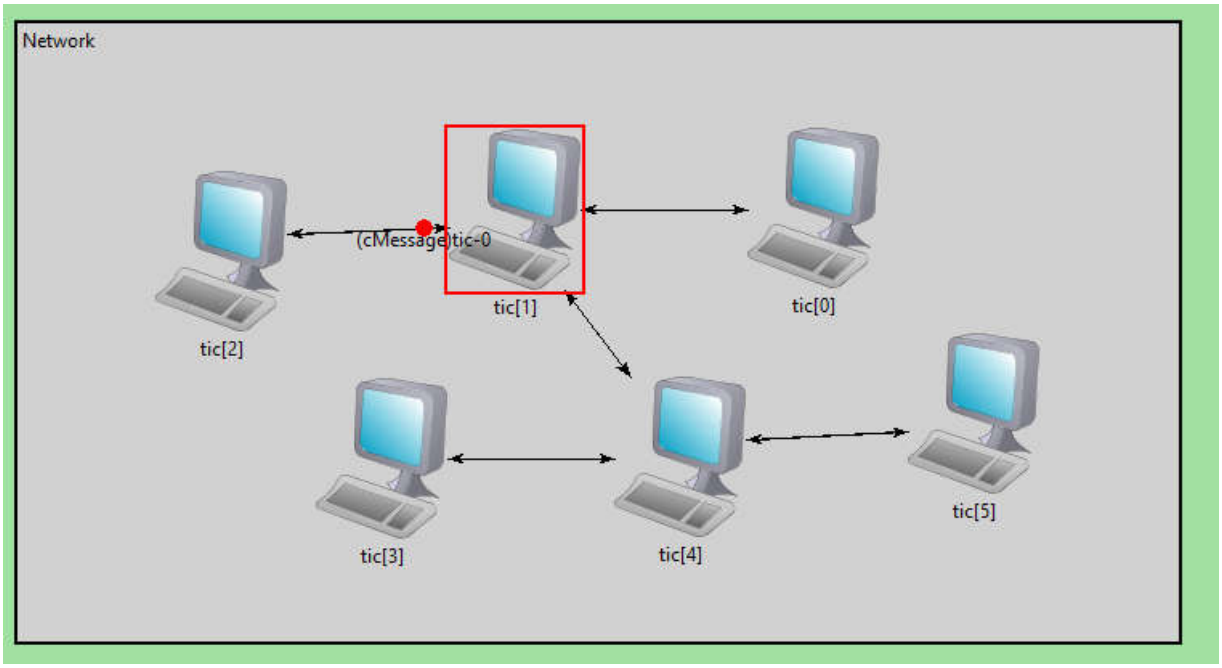
```
network Network
{
    @display("bgb=724,385");

    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 100ms;
        }

    submodules:
        tic[6]: Node {

            @display("is=vl");
        }
    connections:
        tic[0].out++ --> Channel --> tic[1].in++;
        tic[0].in++ <-- Channel <-- tic[1].out++;

        tic[1].out++ --> Channel --> tic[2].in++;
        tic[1].in++ <-- Channel <-- tic[2].out++;

        tic[1].out++ --> Channel --> tic[4].in++;
        tic[1].in++ <-- Channel <-- tic[4].out++;

        tic[3].out++ --> Channel --> tic[4].in++;
        tic[3].in++ <-- Channel <-- tic[4].out++;

        tic[4].out++ --> Channel --> tic[5].in++;
        tic[4].in++ <-- Channel <-- tic[5].out++;

}
```

After that, all that is left is to implement the behaviour of the nodes which will determine how it will handle the arriving messages. The following code implements the previously described behaviour:

```
Define_Module(Node);

void Node::initialize()
{
    if(getIndex()==0)
    {
        char msgname[20];
        sprintf(msgname,"tic-%d",getIndex());
        cMessage *msg= new cMessage(msgname);
        scheduleAt(0.0,msg);
    }
}

void Node::handleMessage(cMessage *msg)
{
    if(getIndex()==3)
    {
        EV << "Message " << msg << " arrived.\n";
        delete msg;
    }
    else
    {forwardMessage(msg);}

}


void Node::forwardMessage(cMessage *msg)
{
    int n = gateSize("out");
    int k = intuniform(0,n-1);

    EV << "Forwarding message " << msg << " on port out ["<< k << "]\n";
    send(msg,"out",k);



}
```

The handleMessage method will check whether the message has reached its goal node or not. If the message has reached its node, then it will output a message indicating that and terminate the simulation. However, if the message has not reached its goal node, then it will call the forwardMessage method which will randomly select the next node to which it will send the packet to.

Simulating this project will give the following results:



Event#	Time	Src/Dest	Name	Info
#1	0	tic[0] --> tic[1]	tic-0	id=0 kind=0
#2	0.1	tic[1] --> tic[2]	tic-0	id=0 kind=0
#3	0.2	tic[2] --> tic[1]	tic-0	id=0 kind=0
#4	0.3	tic[1] --> tic[2]	tic-0	id=0 kind=0
#5	0.4	tic[2] --> tic[1]	tic-0	id=0 kind=0
#6	0.5	tic[1] --> tic[2]	tic-0	id=0 kind=0
#7	0.6	tic[2] --> tic[1]	tic-0	id=0 kind=0
#8	0.7	tic[1] --> tic[0]	tic-0	id=0 kind=0
#9	0.8	tic[0] --> tic[1]	tic-0	id=0 kind=0
#10	0.9	tic[1] --> tic[4]	tic-0	id=0 kind=0
#11	1	tic[4] --> tic[1]	tic-0	id=0 kind=0
#12	1.1	tic[1] --> tic[0]	tic-0	id=0 kind=0
#13	1.2	tic[0] --> tic[1]	tic-0	id=0 kind=0
#14	1.3	tic[1] --> tic[4]	tic-0	id=0 kind=0
#15	1.4	tic[4] --> tic[3]	tic-0	id=0 kind=0

Project 3:

In this project the library Mixim will be used. The purpose of the project is to build a network of wireless sensor networks which are stationary deployed and are not mobile. These nodes will transfer packets between each other as long as they are within each others' broadcasting range.

First, the set of nodes and their types must be defined, a predefined node type is used which is Host802154A.

```
network Wireless extends BaseNetwork
{
    parameters:
        int numNodes; // total number of hosts in the network

        @display("bgb=300,200");
    submodules:
        node[numNodes]: Host802154A;
    connections allowunconnected:
    // all connections and gates are to be generated dynamically
}
```

Then the simulation must be defined in the omnetpp.ini file as follows:

```
[General]
cmdenv-express-mode = true
network = Wireless

##########################################################
#           Simulation parameters                        #
##########################################################
**.**.coreDebug = false
**.playgroundSizeX = 300m
**.playgroundSizeY = 300m
**.playgroundSizeZ = 300m
**.numNodes = 5  ##### control the number of nodes in the nerwork

##########################################################
#           WorldUtility parameters                      #
##########################################################
**.world.useTorus = false
**.world.use2D = true

##########################################################
#                    channel parameters                  #
##########################################################
**.connectionManager.sendDirect = false
**.connectionManager.pMax = 100mW
**.connectionManager.sat = -84dBm
**.connectionManager.alpha = 3.0
**.connectionManager.carrierFrequency = 2.412e+9Hz

##########################################################
#       Parameters for the Host                          #
##########################################################
**.node[*].nicType = "NicCSMA"
```

Given that there are many parameters, only the ones which are of interest to the purpose of the project are explained. First, numNodes parameter in the simulation parameter groups defines the number of nodes in the network.

```
################ PhyLayer parameters #####################
**.node[*].nic.phy.usePropagationDelay = false
**.node[*].nic.phy.thermalNoise = -100dBm
**.node[*].nic.phy.useThermalNoise = true

**.node[*].nic.phy.analogueModels = xmldoc("config.xml")
**.node[*].nic.phy.decider = xmldoc("config.xml")

**.node[*].nic.phy.timeRXToTX = 0s
**.node[*].nic.phy.timeRXToSleep = 0s

**.node[*].nic.phy.timeTXToRX = 0s
**.node[*].nic.phy.timeTXToSleep = 0s

**.node[*].nic.phy.timeSleepToRX = 0s
**.node[*].nic.phy.timeSleepToTX = 0s

**.node[*].nic.phy.sensitivity = -84dBm
**.node[*].nic.phy.maxTXPower = 100.0mW

**.node[*].nic.phy.initialRadioState = 0
```

```
################ MAC layer parameters ####################
**.node[*].nic.mac.queueLength = 5
**.node[*].nic.mac.headerLength = 24bit
**.node[*].nic.mac.bitrate = 15360bps
**.node[*].nic.mac.txPower = 100mW
**.node[*].nic.mac.stats = true
**.node[*].nic.mac.trace = true

**.node[*].nic.mac.ccaDetectionTime = 0.0005s
**.node[*].nic.mac.aTurnaroundTime = 0s #no radio switch times
**.node[*].nic.mac.rxSetupTime = 0s #no radio switch times

**.node[*].nic.mac.backoffMethod = "linear"
**.node[*].nic.mac.macMaxCSMABackoffs = 14
**.node[*].nic.mac.contentionWindow = 20
**.node[*].nic.mac.aUnitBackoffPeriod = 0.04s

# MAC Ack settings (disabled)
**.node[*].nic.mac.useMACAcks = false
**.node[*].nic.mac.ackLength = 0bit
**.node[*].nic.mac.macMaxFrameRetries = 0
**.node[*].nic.mac.macAckWaitDuration = 0s
**.node[*].nic.mac.sifs = 0s

**.node[*].nic.mac.macMinBE = 0 #only used for exponential backoffs
**.node[*].nic.mac.macMaxBE = 0 #only used for exponential backoffs
```

The previous two screenshots configure the parameters of both the physical and MAC layer.

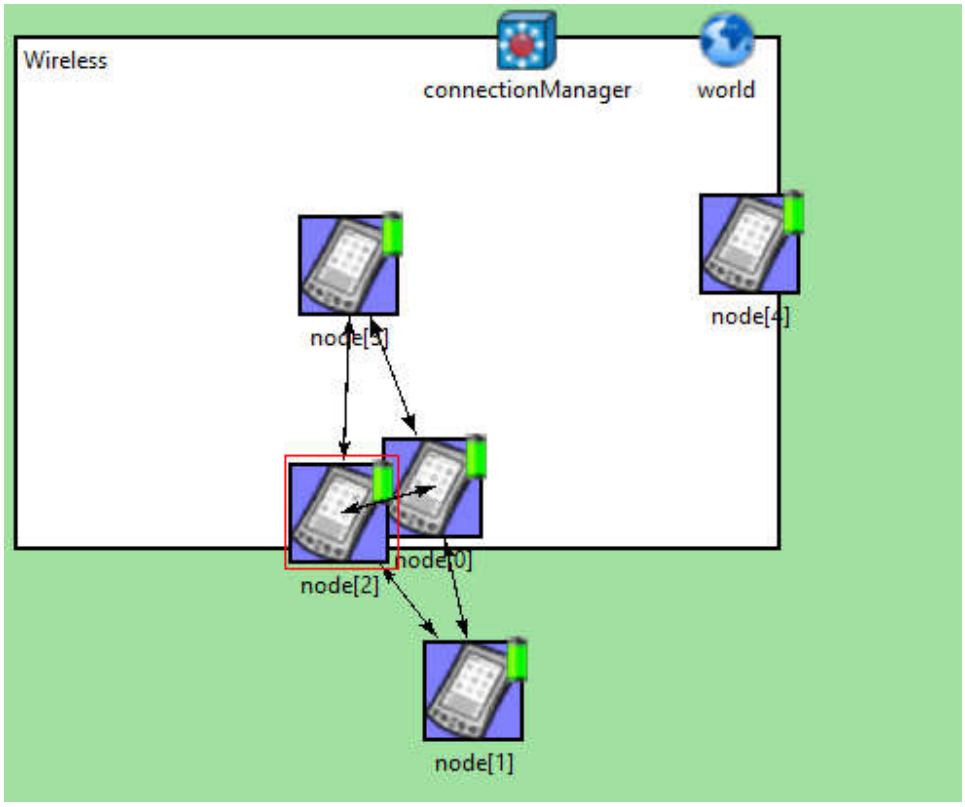
```
################ Application layer parameters ############
**.node[*].applicationType = "SensorApplLayer"
**.appl.trafficType = "periodic" #
**.appl.trafficParam = 1s #in seconds
**.appl.broadcastPackets = true
**.appl.nbPackets = 3
**.appl.initializationTime = 10s
################ NETW layer parameters ###################
**.node[*].networkType = "Flood"#
#**.node[*].netwl.debug = true
**.node[*].netwl.stats = true
**.node[*].netwl.headerLength = 24 bit
```

As for the application layer parameters, the traffic type is chosen to be periodic.

```
################ Mobility parameters #####################
**.node[*].mobilityType = "StationaryMobility"
**.node[*].mobility.debug = false
**.node[*].mobility.updateInterval = 0.1s
**.node[0].mobility.initialX = 150m
**.node[0].mobility.initialY = 200m
**.node[0].mobility.initialZ = 250m

**.node[1].mobility.initialX = 250m
**.node[1].mobility.initialY = 100m
**.node[1].mobility.initialZ = 100m

**.node[2].mobility.initialX = 250m
**.node[2].mobility.initialY = 200m
**.node[2].mobility.initialZ = 200m

**.node[3].mobility.initialX = 50m
**.node[3].mobility.initialY = 100m
**.node[3].mobility.initialZ = 110m

**.node[4].mobility.initialX = 150m
**.node[4].mobility.initialY = 180m
**.node[4].mobility.initialZ = 100m

**.node[5].mobility.initialX = 50m
**.node[5].mobility.initialY = 200m
**.node[5].mobility.initialZ = 10m

**.battery.capacity = 10 mAh # the real battery capacity
**.battery.nominal = 10 mAh # nominal battery capacity
**.battery.voltage = 3V # battery voltage
**.battery.resolution = 1 s # when the battery capacity is update
**.battery.publishTime = 1 s
**.battery.numDevices = 1 # number of battery for each node
**.battery.publishDelta = 0
```

The previous screenshot has the code which determines the location of each node and their mobility type which is stationary for the purposes of this project. It also configures the physical attributes of the nodes.

Simulating this project would produce the following results:



Event#	Time	Src/Dest	Name	Info
#106	10.337212975725	node[3] --> node[0]	Data	id=53 kind=22003 length=8 bytes
#106	10.337212975725	node[3] --> node[2]	Data	id=52 kind=22003 length=8 bytes
#129	10.661879642391	node[0] --> node[1]	Data	id=89 kind=22003 length=8 bytes
#129	10.661879642391	node[0] --> node[2]	Data	id=91 kind=22003 length=8 bytes
#129	10.661879642391	node[0] --> node[3]	Data	id=88 kind=22003 length=8 bytes
#154	10.906546309057	node[1] --> node[0]	Data	id=126 kind=22003 length=8 bytes
#154	10.906546309057	node[1] --> node[2]	Data	id=125 kind=22003 length=8 bytes
#167	10.941879642391	node[2] --> node[0]	Data	id=142 kind=22003 length=8 bytes
#167	10.941879642391	node[2] --> node[1]	Data	id=144 kind=22003 length=8 bytes
#167	10.941879642391	node[2] --> node[3]	Data	id=141 kind=22003 length=8 bytes
#202	11.271212975723	node[1] --> node[0]	Data	id=174 kind=22003 length=8 bytes
#202	11.271212975723	node[1] --> node[2]	Data	id=173 kind=22003 length=8 bytes
#219	11.346546309057	node[2] --> node[0]	Data	id=200 kind=22003 length=8 bytes
#219	11.346546309057	node[2] --> node[1]	Data	id=202 kind=22003 length=8 bytes
#219	11.346546309057	node[2] --> node[3]	Data	id=199 kind=22003 length=8 bytes
#245	11.40476574409	node[0] --> node[1]	Data	id=242 kind=22003 length=8 bytes
#245	11.40476574409	node[0] --> node[2]	Data	id=244 kind=22003 length=8 bytes
#245	11.40476574409	node[0] --> node[3]	Data	id=241 kind=22003 length=8 bytes
#271	11.649432410756	node[0] --> node[1]	Data	id=284 kind=22003 length=8 bytes
#271	11.649432410756	node[0] --> node[2]	Data	id=286 kind=22003 length=8 bytes
#271	11.649432410756	node[0] --> node[3]	Data	id=283 kind=22003 length=8 bytes
#296	11.857212975725	node[3] --> node[0]	Data	id=321 kind=22003 length=8 bytes
#296	11.857212975725	node[3] --> node[2]	Data	id=320 kind=22003 length=8 bytes
#313	11.901879642391	node[3] --> node[0]	Data	id=347 kind=22003 length=8 bytes

Reference List

1. O. Younis, S. Fahmy, HEED: A hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks, IEEE Trans. Mobile Computer, 366379, 2004.

2. E. Ever, R. Luchmun, L. Mostarda, A. Navarra, and P. Shah, red UHEED - An Unequal Clustering Algorithm For Wireless Sensor Networks. inSensornets 2012, 2012.

3. N.Aierken, R.Gagliardi, L.Mostarda, Z. Ullah , RUHEED-Rotated Unequal Hybrid Energy Efficient Distributed Algorithm For Wireless Sensor Networks. In Gwangiu ,2015.
4. Samayveer Singh, A K Chauhan, SanjeevRaghav, VikasTyagi,SherishJohri, Heterogeneous protocols for increasing the life time of wireless sensor networks, Journal of Global Researchin Computer Science Vol. 2, No.4, April 2011, pp.172-176.
5. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “An Application-Specific Protocol Architecture for Wireless Microsensor Networks,” IEEE Trans. Wireless Comm., vol. 1, no. 4, pp. 660-670, Oct. 2002.

6. Heinzelman, Wendi Rabiner, Anantha Chandrakasan, and Hari Balakrishnan. "Energy-efficient communication protocol for wireless microsensor networks."System sciences, 2000. Proceedings of the 33rd annual Hawaii international conference on. IEEE, 2000.
7. Handy, M. J., Marc Haase, and Dirk Timmermann. "Low energy adaptive clustering hierarchy with deterministic cluster-head selection." Mobile and Wireless Communications Network, 2002. 4th International Workshop on. IEEE, 2002.
8. Heinzelman, Wendi B., Anantha P. Chandrakasan, and Hari Balakrishnan. "An application-specific protocol architecture for wireless microsensor networks."Wireless Communications, IEEE Transactions on 1.4 (2002): 660-670.
9. Tong, Mu, and Minghao Tang. "LEACH-B: an improved LEACH protocol for wireless sensor network."Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on. IEEE, 2010.
10. Xu, Jia, et al. "Improvement of LEACH protocol for WSN." Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on. IEEE, 2012.