Evaluation Study of Leader Selection Algorithms in Wireless Mesh Networks

---

A dissertation submitted by

Nkosinathi Hendrick Zulu

(20042649)

(B.Sc. Hons. Computer Science)

Submitted to the Department of Computer Science at the University of Zululand's Faculty of Science and Agriculture, in fulfilment of the requirements for the degree of

Master of Science in Computer Science

Supervisor: Prof. MO Adigun

2015

DECLARATION

This dissertation represents the author's original work, conducted at the University of Zululand. It is submitted for the award of the degree of Master of Science in Computer Science in the Faculty of Science and Agriculture at the University of Zululand, KwaDlangezwa. No part of this research has been submitted in the past, or is being submitted, for a degree or examination at any other University. All sources used in this dissertation have been duly acknowledged. Parts of this work were published and presented at SATNAC 2014 in South Africa, and IEEE ICAST 2014 in Nigeria.

Nkosinathi Zulu

Signature: ________________________

DEDICATION

I dedicate this piece of work to My Mother Mrs N Zulu and Melokuhle Zulu.

ACKNOWLEDGEMENTS

I would like to begin by acknowledging all members of my family, particularly my mother, Mrs N Zulu, who gave me the rare opportunity to explore all my potential in studying by giving financial and `moral support. I would like to pass myheartfelt words of gratitude to my supervisors, Prof. M.O. Adigun and Mr. P Mudali of the University of Zululand for guiding me throughout this research work. I would also like to thank the Department of Computer Science at the University of Zululand for allowing me to pursue my studies. My appreciation also goes to Mr. B Mutanga for mentoring me in my first year of the Master's programme and making sure that I got a solid foundation and introduction to research and wireless mesh networks. Special thanks go to Mr. Oki Kayode for supporting and advising me throughout the duration of this project. Many thanks to the Wireless Mesh Networks Research Group and all research colleagues in the Department of Computer Science for helping me to make progress in the right direction throughout my project. I am honoured by the financial support provided by Telkom, Thrip, and Huawei to the Department of Computer Science at the University of Zululand through the CoE program. Above all, I would like to say that I am grateful to God, my creator and the custodian of knowledge and wisdom for giving me strength and courage to keep going throughout the duration of my study.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

List of Acronyms

ACE- An Emergent Algorithm Highly Uniform Cluster Formation

DEEAC- Distributive Energy Adaptive Clustering Protocol for WSN

CRPBLN- Clustering Routing Protocol on Location node in WSN

EDBCP- Energy and Distance Based Clustering Protocol for WSN

EDC- Event-Driven Clustering Routing Algorithm for WSN

EECED- Energy Efficient Clustering Algorithm for Event-driven in WSN

EECS- Energy Efficient Clustering Scheme in Wireless Sensor

EEDBC- Energy Efficient Distance Based Clustering

EEHCA- Energy Efficient Homogeneous Clustering Algorithm for WSN

MA-Mesh Authenticator

MAPs- Mesh Access Points

WMN - Wireless Mesh Network

WSN- Wireless Sensor Network

MKD Mesh Key Distributor

MKDD- Mesh Key Distributor Domain

MPs- Mesh Points

LEACH-C- Centralized low-energy adaptive clustering hierarchy

LEACH- Low Energy Adaptive Clustering Hierarchy

LSAs- Leader Selection AlgorithmsUDCA- An Energy Efficient Clustering Algorithm for Wireless Sensor Network

ABSTRACT

A Wireless Mesh Network (WMN) is a group of wireless devices which can dynamically communicate with one another in multi-hop manner. Wireless mesh networks (WMNs) are getting more attention and recognition as a scalable substitute for Wired Network infrastructure. The rising popularity of WMNs has necessitated the development of security mechanisms. The newly-ratified IEEE 802.11s mesh networking standard specifies a security mechanism that builds upon the IEEE 802.11i security standard meant for wireless local area networks. The IEEE 802.11s security mechanism requires the existence of a single Mesh key Distributor (MKD) which assists the authentication of new nodes that join the network. However, there is no mechanism for selecting a new MKD if the current MKD is unreachable or has failed. This scenario can occur due to the dynamic nature of WMN backbone topologies, wireless link variability in deployed networks, and battery depletion in battery-powered WMNs. MKD selection in WMN deployments can be performed by adapting Leader Selection Algorithms from Wireless Sensor Networks.

The goal of this research study is to evaluate the existing leader selection algorithms in the context of selecting an MKD for WMNs. This goal was achieved by evaluating the existing wireless ad hoc networks leader selection algorithms. The energy-based and position-based leader selection algorithms were evaluated in the context of MKD selection and were subjected to different leader selection rounds and network sizes. The evaluation shows that on the energy-based LSAs, the heterogeneous-based LSAs EECS and UDAC) outperform the homogeneous-based LSAs (LEACH and EECHA) based on communication overhead cost and the energy consumption rate. Whilst the homogeneous-based LSAs outperform the

heterogeneous LSAs in terms of leader selection delay. The evaluation further revealed that, for the position-based LSAs, the event-based algorithms (EDC & EECED) outperform the distance-based algorithms (EDBCP & EDBC) in the achieved performance for communication overhead cost, leader selection delay and the energy consumption rate.

CHAPTER 1

INTRODUCTION

1.1 Preamble

Wireless mesh networks (WMNs) create a dynamic infrastructure using a different wireless networking technology and ad hoc routing protocols, which together let service providers or communities establish networks in places without prior groundwork (Ishmael et al, 2008). Thus, WMNs are useful in rural scenarios. A rural African WMN deployment often means that mesh devices are battery-powered due to the lack of stable electrical supplies.

A typical WMN (See Figure 1) consists of two types of device: backbone devices and client stations (Akyildiz I.F et al, 2005). The backbone devices consist of Mesh Points (MPs) and Mesh Access Points (MAPs). The backbone of a WMN is a self-configuring network in which all MPs and MAPs can route traffic either directly to a destination (if possible) or via a multi-hop path (Salem N.B et al, 2006).

The dynamic nature of WMN topology allows both backbone devices and client stations to enter and exit the network backbone devices and client stations may exit the network due to battery drainage (Allen W., et al, 2005). Backbone devices may also experience a temporary lack of connectivity due to the transient nature of wireless links when WMNs are deployed (Lundgren H, et al 2006). Currently, security aspects of WMNs have not received as much attention as routing protocols and energy-efficiency (Salem N.B et al, 2006).

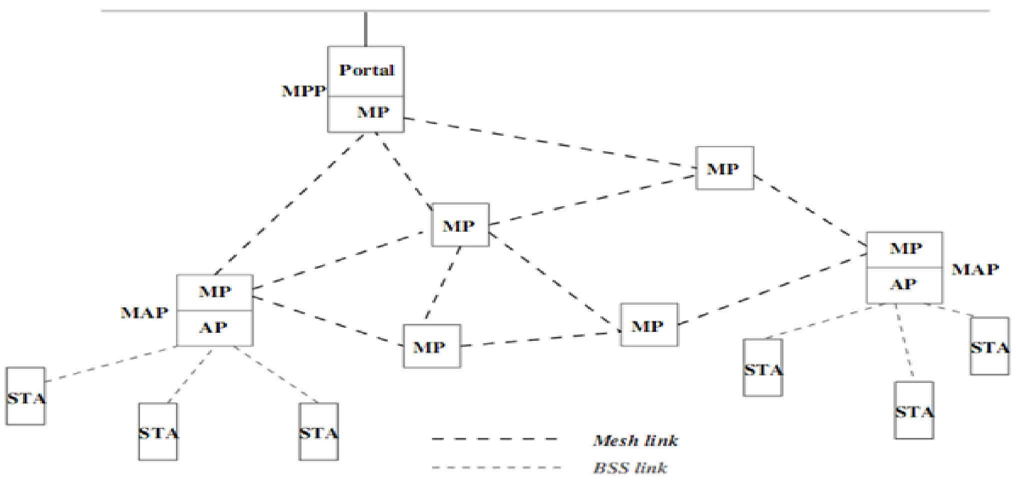


Figure 1.1 Wireless Mesh Network Architecture (Wang X, et al, 2008)

The scalable and *ad hoc* nature of WMNs makes it to be vulnerable to different security threats; hence, security is one of the important issues that need to be considered (Guido R. et al, 2005). The dynamic nature of WMN means that nodes can leave the network and rejoin the network at any given time and thus, the authentication of devices is a critical feature.

Authentication must take place between backbone devices as well as between MAPs and their associated stations. These authentication mechanisms are specified in the IEEE 802.11s standard for mesh networking using IEEE 802.11 technology. The IEEE 802.11s authentication mechanism is adapted from the IEEE802.11i standard designed for Wireless Local Area networks. The vital characteristic of the 802.11s authentication mechanisms is the presence of a Mesh Key Distributor (MKD).. The authentication process is fully dependent on the existence and availability of the MKD but, due to the *ad hoc* nature of the WMN backbone, the transient nature of wireless links when deployed and battery drainage (particularly in rural areas) there may be times when the MKD is neither present nor available. This scenario requires the efficient selection of a new MKD so that the device authentication process is not compromised. To the best of the researcher’s knowledge, there is no existing mechanism for the selection of a replacement for the unavailable MKD.

In most developing countries (particularly in Africa) where electricity supplies are not reliable, the MKD can become unavailable or not-reachable due to power outage, battery depletion or transient wireless links. Currently, there is no leader selection algorithm for selecting a new MKD if the current MKD fails or dies (Guido R. et al, 2005). Due to the ad hoc nature of WMN this study seeks to evaluate leader selection algorithms in the context of MKD selection from other ad hoc networks. Hence, this study seeks to evaluate the performance of energy-based leader selection algorithms (LSAs) and position-based leader selection algorithms (LSAs) in the context of MKD selection in WMNs. The evaluation reveals that under energy based LSAs, heterogeneous based LSAs (EECS and UDAC) do better than the homogeneous based LSAs (LEACH and EECHA) in the achieved performance for communication overhead cost and energy consumption rate. At the same time the homogeneous-based LSAs do better than the heterogeneous LSAs in terms of leader selection delay. The evaluation of four position-based LSAs (EECED, EDC, EDBCP and EDBC) shows that some of them can be utilized in the context of finding a new MKD in WMNs. Event position-based LSAs (EDC and EECED) outperform the distance position-based LSAs (EDBCP and EDBC) when communication overhead, leader selection delay and the energy consumption rate are considered.

1.2. IEEE 802.11S SECURITY

There are two types of security key holders: a Mesh Key Distributor (MKD) and Mesh Authenticators (MA) (Guido R. et al, 2005). A Mesh Point (MP) can assume the role of the MKD and a MA at the same time. Both roles are optional. The MKD is the centre for key generation and authentication, delegating some of its work to the MAs. MPs are regular stations which have to be authenticated by an MA or the MKD before they can participate in the network. A MP with MA functionality plays the 802.1X authenticator role and a MP without the MA functionality plays the 802.1X supplicant role.

In a 802.11s WMN, there exists one MKD, multiple MAs and supplicants. A supplicant can become an MA after it passes security key holder association with the MKD. Considering an MP in an IEEE 802.11s secure network, when the MP needs to establish a secure link with a peer MP, a peer link setup procedure is first executed (step 0 in Figure 1.2). In this initial step, the role of an MP is determined and security policy is selected. Whether an MP and its peer MP are an 802.1X authenticator or supplicant MP is determined in the peer link management. As shown in Figure 1.2 architecture, there is only one MKD with which multiple MAs are associated. A supplicant performs security authentication through MAs. The MKD domain (MKDD) (*Figure 1.2*) is made up of the set of MAs, Supplicants and single MKD. Optionally, the MKD is connected to an AS through which 802.1X authentication is executed.

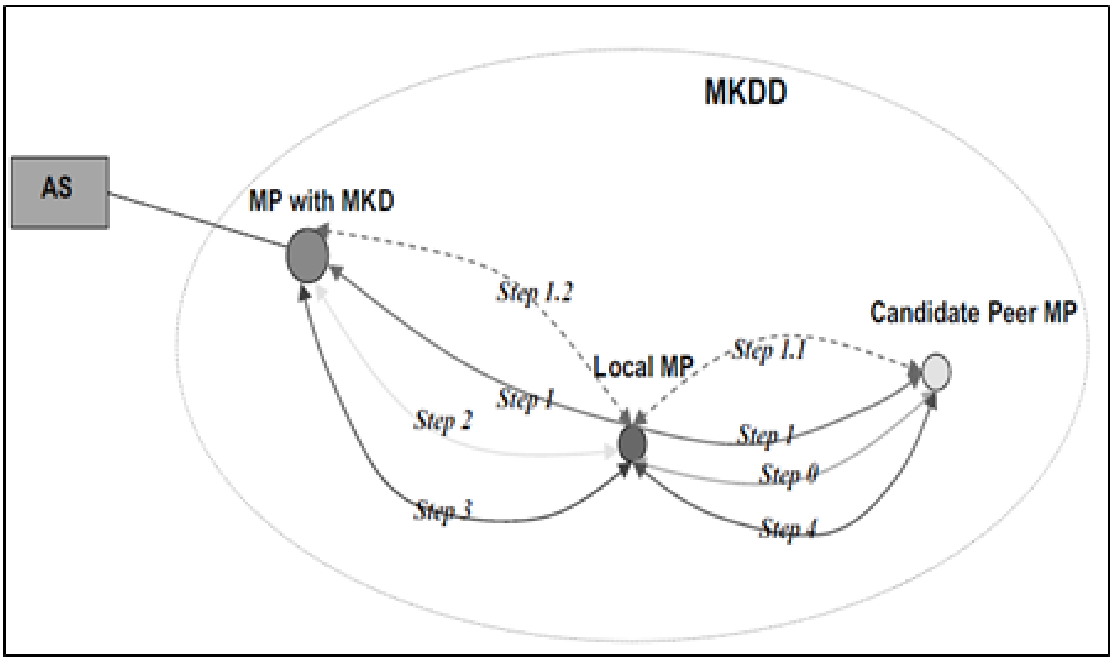


Figure 1.2 Major Function Blocks of 802. 11s Mesh Security (Kuhlman. D et al, 1997)

1.3 Statement of the Problem

Although the functionalities of MKD are specified in IEEE 802.11s standard there is no method for determining how an MKD is selected. According to 802.11s it is assumed that the first node to join the network becomes an MKD. However, when the need to select a new MKD arises, no mechanism is specified in the standard to accomplish the selection. There are various reasons why a new MKD may be required in the network, e.g. the current MKD may not be reachable, or failing to serve its purpose due to power failure. Randomly picking a new MKD is not the best solution (Akyildiz I.F et al, 2005). When security mechanisms are deployed for WMN it is always assumed that the MKD is there as the first station authenticated, but this is not always the case.

1.4 Research Questions

Pertinent to the study, three research questions were raised:

1. How is the process of evaluating existing leader selection algorithms (LSAs) in the context of MKD selection going to be conducted?
    a. How can we create a classification framework for existing LSAs?
    b. What are the selection algorithms that can be used for selecting the MKD?
    c. What are the evaluation considerations for MKD selection algorithms?

1.5 Rationale for the Study

The design features for real-world implementation will provide good mechanisms for selecting a new MKD in the case where the existing MKD is no longer present in the Wireless Mesh Network. This enables the network authenticate new stations without any interruption. This work centres on security based issues for deployment of wireless mesh networks in rural-based areas. Other researchers could benefit from this work since there is little research that has been undertaken on the 802.11s standard.

1.6 Research Goal and Objectives

In this section, the researcher presents the goal of this research study. The goal is further divided into three objectives:

1.6.1 Research Goal

The goal of this research study is to evaluate the existing Leader Selection Algorithms in the context of selecting an MKD for WMNs.

1.6.2 Research Objectives

i. To classify the existing LSAs.

ii. To select certain LSAs for evaluation.

iii. To evaluate the selected algorithms.

1.7 Research Methodology

In order to achieve the aforementioned objectives, both Simulation and Literature Survey research methods were used. Simulation and Literature Survey research methods were chosen for this study because they complement each other. In Sub-sections 1.6.1 and 1.6.2 a brief explanation of these two research methods is presented, while comprehensive details of how these methods were implemented in this study are discussed in Chapter 4.

1.7.1 Primary Research Method: Simulation

The primary research method involved simulation using Network Simulator (NS2) version 2.34. NS2 is a discrete event simulator for networking research and is based on standard OTcl and C++. The simulation was used to evaluate different LSAs from other wireless networks in order to recommend the LSA(s) that can be adopted for MKD selection in WMN. The simulation evaluation was done using four performance metrics: Communication Overhead, Leader Selection Delay, Energy Consumption Rate and Network Average Remaining Energy. The results obtained via this research method helped the study to achieve the third objective and to partly answer the second and third research questions.

1.7.2 Secondary Research Method: Literature Survey

The secondary research method involved classification and selection of LSAs using Literature Survey. This method entails surveying the background of the area of interest. The theoretical part of this research thus involves analysing previous work in the field of leader selection algorithms in wireless ad hoc networks. The results obtained via this research method, coupled with the simulation results, helped to fully achieve the three objectives and to convincingly answer the stated research questions.

CHAPTER 2

EXPLORING LEADER SELECTION ALGORITHMS FOR WIRELESS MESH NETWORKS

2.1. Introduction

Wireless mesh networks contains three types of nodes, Mesh Points, Mesh clients and a gateway. One of the mesh points acts as a leader in the network and is known as a Mesh Key Distributor (MKD) (IEEE 802.11s). The main problem is that there is no mechanism for electing a new MKD when the current MKD is failing due to energy constraints or the MKD becoming unavailable due to network partitioning. This chapter surveys literature on existing leader election algorithms. Such leader selection algorithms originate from other wireless networks such as Wireless sensor networks where cluster heads are selected as part of the WSN architecture. To the best of the researcher's knowledge, there is no work that has been done on the evaluation of leader selection algorithms in the context of selecting a MKD for WMN.

A taxonomy *(Figure 2.1) w*as used to classify existing leader selection algorithms. The literature review helped to devise a suitable framework that would help the study to select leader selection algorithms that this work is going to evaluate in the context of selecting a MKD for WMN. Section 2.2 discusses the comparison of wireless mesh networks and wireless sensor network, while section 2.3 discusses the classification of LSAs and comparison of LSAs Section 2.4 discusses selection frameworks for LSAs.

2.2. Comparing Wireless Sensor networks with Wireless Mesh Networks

The goal of this work is to evaluate the feasibility of employing leader selection algorithms to select a MKD in WMN. As outlined earlier in this chapter, there are no existing leader selection protocols in WMN. Hence, this study focuses on an evaluation of existing leader selection algorithms from WSNs with the aim of selecting a MKD for a WMN.

2.2.1. Wireless Sensor Networks

This section discusses features of wireless sensor networks. A wireless sensor network is a wireless network that consists of autonomous nodes that employ sensors to monitor environmental conditions (Baccarelli. E et al, 2005). A wireless sensor network is scalable because it can have thousands of sensor nodes, and the self-healing feature of WSNs makes the network reliable (Yick J, at el. 2008).   Wireless sensor network infrastructure consists of sensor nodes and a base station (BS), also known as a sink, which are responsible for transmitting data from the sensing area to the network monitors (Katiyar .V at el ,2011).

Figure 2.1 depicts the traditional WSN, where the single sink is responsible for relaying data from sensor nodes to control centre and this makes the network to be centralised (Prabhu S.R.B, at. el 2011). WSNs are formed by sensor nodes which are generally immobile, smaller in size and of low cost but they are burdened with comparatively low processing and memory capabilities, limited power supply and quite low link bandwidth. WSNs range from simple single-hop data collection mechanisms to intelligent multi-hop sensor networks (Bouckaert S. 2009). WSN sensor nodes are powered by battery, and as a result, wireless sensor networks performance goes down as node battery power goes down. However, power management plays a role in prolonging the network's life time (Zheng J. et al, 2009). In WSN There are many leaders selected per round since they are meant for clustering the network.

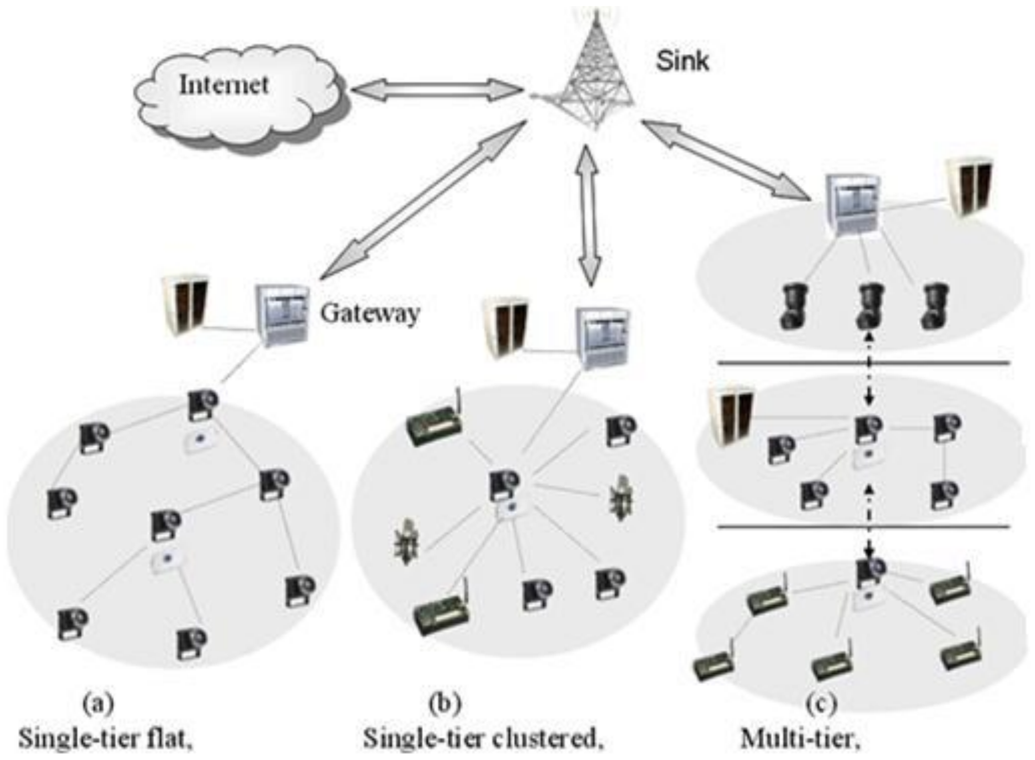


Figure 2.2 Wireless Sensor Networks (Baccarelli. E et al, 2005)

2.2.2. Wireless Mesh Networks

This section discusses features of wireless mesh networks. WMNs are multi-hop systems where nodes assist each other in transmitting packets throughout the network (Poor. R, 2003). WMNs are self-healing and a self-configuring in nature (Akyidiz. F, et.al, 2005).Wireless Mesh Networks are scalable and can handle hundreds and thousands of nodes and are also reliable and adaptable. In WMNs networking, nodes form part of network infrastructure and are dedicated to the routing task. A WMNs consists of mesh points, mesh clients and gateways as members of the infrastructure. In WMNs, all nodes are entitled to collect and transmit data, which makes WMNs to be distributed type of a network. In WMNs data is transmitted in a multi-hop manner [Poor R et al, 2003]. Mesh points are immobile and their processing, memory and bandwidth capacities generally exceed those of traditional ad hoc network nodes.

Table 2.2.1 WMN vs WSN.

Features to be compared	WMN	WSN
Configuration and Deployment	Self-healing and easy to be deployed	Self-healing and easy to be deployed
Scalability	Yes	Yes
Reliability	Yes	Yes
Types of nodes that form network architecture	Mesh Client, Mesh Access Points and Gateways.	Sensors node and Sink node
Data transmission mechanism	Multi-hop	Recently support multi-hop
Centralised	No	Yes
Distributed	Yes	No
Power Management	Required	Highly Required
Node Properties	Mesh Points are immobile, high processing power, High memory, relatively cheap.	Sensor nodes are immobile, small in size and of a low cost, battery powered.
Number of leader(s) selected	One leader selected (MKD)	Many leaders selected (Cluster leaders)
Purpose of leader selection	Security: Authentication	Energy efficiency: Clustering

In rural areas in most developing countries, there is unreliable electricity power supply while in some other areas, there is no electricity at all. As a result wireless mesh nodes suffer during power outages (Mudali P. et al, 2011). One of the most feasible solutions to the problem of node power outage is to use renewable sources such as wind or solar/battery powered technology (Pejovic V et al, 2009). In WMN an MKD is selected for security purposes, particularly for authenticating a new node and only one MKD is needed for the entire mesh network (Guido R. et al, 2005).

Table2.1 presents a comparison between Wireless Mesh Networks and Wireless Sensor Networks. Wireless Mesh Networks and Wireless Sensor Networks are both self-organizing and easy to easily deployed. This comparison guides the study towards deriving frameworks for analysing existing leader selection protocols. Features such as the Data transmission mechanism, which is multi-hop for both WMNs and WSNs inform us that sensor network is becoming more similar to WMN. This comparison reveals to us that relatively similar metrics can be used to evaluate both WSN and WMN protocols. This comparison shows that there a lot of commonalities between WMN and WSN. There are also differences such as the number of leaders, and network architecture.

2.3. Classification of and Comparisons of existing LSAs

In this section, we discuss both the classification framework and the selection framework that are guided by the comparison from section 2.2. In this study, taxonomy has been used for classifying LSAs from the literature. Figure 2.2 shows two categories of leader election algorithms, the first category is Energy Based LSAs which is also categorised to Heterogeneous Energy Based LSAs and the second category is Position Based LSAs, which is also categorised into Distance Based and Hotness Value Based LSA.

The first category is the energy based leader selection, which is divided into two subcategories, heterogeneous energy base selection and homogeneous energy based selection. Homogeneous Energy Based Selection: It is assumed that all nodes on the network have equal remaining energy and that the election of a new cluster head is being done stochastically (Gamwarige S, et al, 2009). The second category is position based election, which is also divided into two categories, distance based selection (distance from a base station) and Event based selection, (he nodes that are near by the event that has been reported) (Shirmohammadi, et al. 2009). **Distance Based Selection:** The node that is near the base station stands more chances to be elected cluster leaders. **Event Based Selection:**

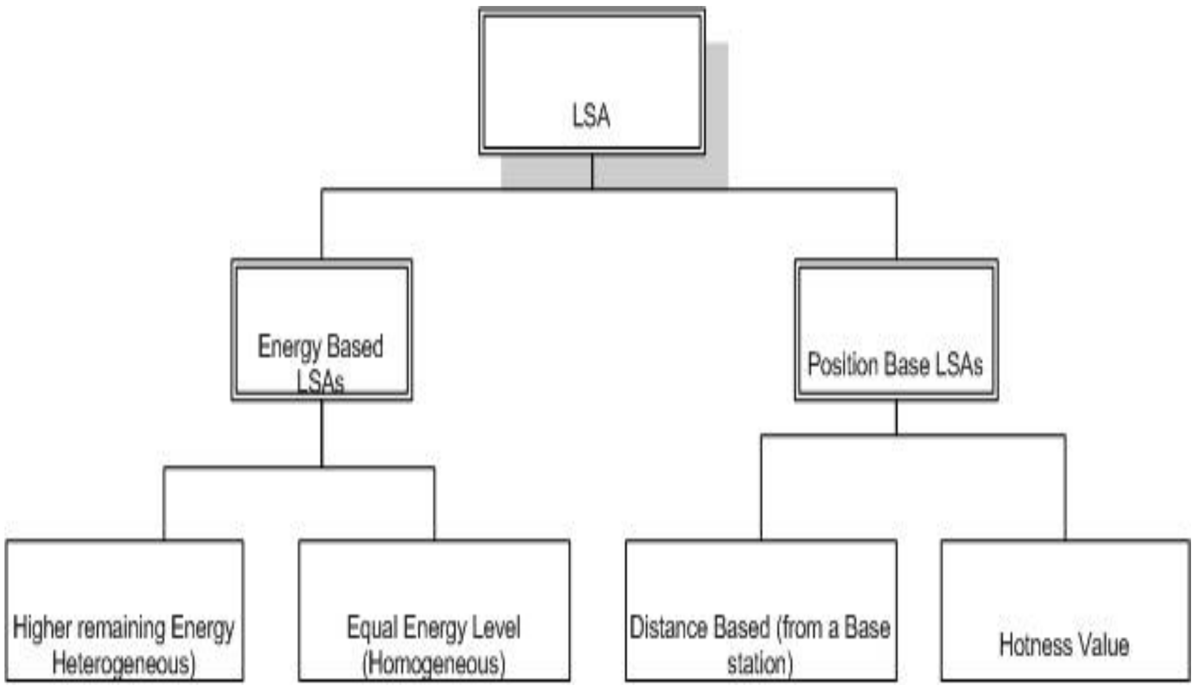

LSA
Energy Based LSAs
Position Base LSAs
Higher remaining Energy Heterogeneous)
Equal Energy Level (Homogeneous)
Distance Based (from a Base station)
Hotness Value

Figure 2.3 Leader Selection Algorithms Taxonomy (Wang Q, Hassanein H, 2004)

The node that is near the place where the event that is reported stands a good chance of being elected as a cluster leader. This study analysed exiting LSAs using the following parameters:

year of an algorithm, type of algorithm, criteria for selecting a leader, simulation environment, and performance metrics. The Year parameter will help to guide the study towards analysing and selecting algorithms that have been recently proposed.

2.3.1. Energy based selection

This section discusses heterogeneous energy selection based algorithms and homogeneous energy based selection algorithms. This study reviews three different leader selection algorithms for each category. Sections i) and ii) are based on the review of both heterogeneous and homogeneous leader selection algorithms.

i) Heterogeneous Energy Based Selection

Energy Efficient Clustering Schemes in Wireless Sensor Networks (EECS) (M Ye, et al, 2005), a Centralized low-energy adaptive clustering hierarchy (LEACH-C) (Heinzelman W. et al, 2002) and An Energy Efficient Clustering Algorithm for Wireless Sensor

Networks(UDCA) (Boregowda S.B et al, 2010) are heterogeneous LSAs. In heterogeneous energy based selection, nodes continuously compare their remaining energy with the current leader and other nodes in the network.

The process of selecting a new leader resumes when the remaining energy of the current leader is lower than the remaining energy of a normal node in the network. Energy based LSAs use an energy model to compute their remaining energy. Heterogeneous LSAs make the following assumptions: all nodes are energy constrained and perform a similar task, all sensor nodes have a unique ID, all nodes are transmitting to the Cluster Head, and the cluster head transmits to the Base station. In each, there is a single hop between the base station and the cluster head. Heterogeneous LSAs consist of different phases, namely the initial phase, setup phase and steady phase. In the initial phase, the elector sends the energy request message to the network. In the setup phase, the cluster head is being selected based level of remaining energy and on the steady phase node send data to the cluster head and the cluster head aggregate that data for different nodes and send it to the base station. Table2. 2, Table 2.3 and Table 2.4 show the parameters for each heterogeneous energy based selection algorithms respectively.

In this study, an EECS novel clustering Schema for WSN was presented. EECS study uses remaining energy as a criterion for selecting a cluster head. EECS introduces a novel technique to balance the load among the cluster leaders. EECS is a distributed type of leader selection algorithm and energy efficient algorithm in nature which makes it more suitable to be evaluated in the context of MKD selection for WMNs. The study was simulation-based using MATLAB. Two performance metrics were considered, namely network lifetime and total energy consumption. The results of this study show that EECS prolong network lifetime.

a) Energy Efficient Clustering Scheme in Wireless Sensor Networks (EECS)

Table 2.2.2 Parameters for EECS

Parameters	Value
Name of the protocol	An Energy Efficient Clustering Scheme in Wireless Sensor Networks(EECS)
Year	2005
Type Algorithm	Heterogeneous
Criteria of Leader selection	Remaining energy
Tools used for simulation	MATLAB
Performance metrics	Energy consumption and network lifetime

The study of EECS was simulated in MATLAB, which is good for simulating WSN algorithm but not for simulating WMN (Bai X. et al, 2011), to improve the study so that it can be used for evaluation of leader selection algorithms in the context of selecting a MKD of WMNs.

b) The Low Energy Adaptive Clustering Hierarchy-Centralized (LEACH-C)

This study presents a mechanism for clustering a WSN and determining cluster leaders. In this study, the cluster leader is chosen based on the remaining energy level. This was simulation based and the tool that was used for the simulation is network simulator 2 (Ns2). Total energy consumption, number of cluster leaders and network lifetime were used as performance metrics. The results of this study reveal that LEACH-C is energy efficient. The LEACH-C algorithm is a centralised type of leader selection algorithm, which makes it not to not recommended for evaluation in the context of MKD selection in WMNs, which are dynamic and highly scalable in nature.

Table2.2.3 Parameters for LECH-C

Parameters	Value
Name of the protocol	The Low Energy Adaptive Clustering Hierarchy-Centralized (LEACH-C)
Year	2005
Type Algorithm	Heterogeneous
Criteria of Leader selection	The node that is having highest level of energy become a CH.
Tools used for simulation	Ns2
Performance metrics	Number of CH per round ,Network lifetime, energy consumption rate

c) An Energy Efficient Clustering Algorithm for Wireless Sensor Networks (UDCA)

Table2.2.4 Parameters for UDCA

Parameters	Value
Name of the protocol	An Energy Efficient Clustering Algorithm for Wireless Sensor Networks
Year	2010
Type Algorithm	Heterogeneous
Criteria of Leader selection	The node that is having the highest level of energy becomes a CH. ID is used in a case where two or more nodes are having equal energy level
Tools used for simulation	Ns2
Performance metrics	Number of CH per round, percentage of living nodes, energy dissipation rates

This study presents a novel clustering algorithm which maximises the network lifetime by reducing the number of communications among sensor nodes. This study also incorporates a new distributed cluster formation method that enables the self-organization of a huge number of nodes - a feature which makes this study more suitable to be evaluated in the context of MKD selection for WMN. This approach maintains a constant number of clusters by prior selection of cluster leader and rotating the role of cluster leaders to evenly distribute energy load among all sensor nodes.

In this study, remaining energy is used as the criterion for selecting a cluster leader in a distributed manner. The study was simulation based and the tool that was used for simulation is network simulation 2(Ns2). The methodology and tool that is being used for proof of concept in this study make the evaluation of UDCA easy because ns2 is one of the WMN simulation environments. The three performance metrics that were considered were energy consumption, number of cluster leaders and network lifetime. The result of this study shows that UDCA reduces energy consumption by employing clustering techniques.

The study on heterogeneous algorithms shows that there are features that need to be improved so that heterogeneous algorithms can be evaluated in the context of selecting a MKD for WMN. Features such as number of cluster leader that are chosen per round in each above studies cannot be used for evaluation in the context of selecting a MKD, so they must be modified such that they choosing one cluster head. The performance metrics that are used for these algorithms are not enough for MKD selection algorithms. Hence, performance metrics such as leader selection delay need to be added for these algorithms to be evaluated in the context of the MKD selection for WMN This metric was used to compute time taken to select a one cluster leader per round.

ii) Homogeneous energy based selection

Leader election Protocol like Low energy adaptive clustering hierarchy (LEACH) (Heinzelman, W. et al, 2000) , Energy Efficient Homogeneous Clustering Algorithm for WSN (EEHCA)(Singh, et al.2010) , An Emergent Algorithm Highly Uniform Cluster Formation(ACE) (Haowen C, at el, 2004) were reviewed for this study under homogeneous energy based selection. In the homogeneous clustering algorithm the leader is elected randomly from the pool of nodes on the network and there is no requirement for a leader to be selected as a leader. This simply means that all the nodes on the network stand a chance of being elected as cluster leaders. Other homogeneous leader selection algorithms use node ID to select the cluster leader, a nodes compare its ID with other nodes on the network and the node with a smaller ID becomes a cluster leader. The main challenge with this type of election is that a node with low energy level can be elected as a cluster leader and that can compromise the network's reliability. Randomly based election requires minimal energy for electing a leader since all nodes have an equal chance of election and this make this type of election strategy applicable for Wireless Mesh Networks. Moreover, this type of election prolongs the life time of the network since the leader will be frequently elected (Heinzelman, W. et al, 2000). The tables in this section present parameters for a single homogeneous energy based leader selection algorithm.

a) Energy Efficient Homogeneous Clustering Algorithm for WSN (EEHCA)

This study presents a new approach to the use of an energy efficient homogeneous clustering algorithm for WSN in which the lifespan of the network is increased by ensuring a homogeneous distribution of nodes in the network. In this algorithm, a cluster leader is randomly selected initially. This was simulation-based and using the tool MATLAB.

Table 2.2.5 Parameters for EEHCA

Parameters	Value
Name of the protocol	Energy Efficient Homogeneous Clustering Algorithm for WSN (EEHCA)
Year	2010
Type Algorithm	Homogeneous
Criteria of Leader selection	Randomly selection by the Base Station.
Tools used for simulation	MATLAB and mathematical expression have been used for evaluation.
Performance metrics	Node Average remaining energy, and time , percentage of living nodes, energy dissipation rates

b) Low Energy Adaptive Clustering Hierarchy (LEACH)

Table 2.2.6 Parameters for LEACH

Parameters	**Value**
Name of the protocol	Low Energy Adaptive Clustering Hierarchy (LEACH)
Year	2002
Type Algorithm	homogeneous
Criteria of Leader selection	Random selection
Tools used for simulation	Ns2
Performance metrics	Number of CH per round, percentage of living nodes, energy dissipation rates

Mathematical modelling was also used for proof of concept. For this study, power consumption was used as a performance metric and the results show that the proposed algorithm extends the network lifetime. The LEACH algorithm presents an algorithm where the network is divided into small clusters and each cluster has its own leader. Generally, cluster leaders lose their energy faster than other nodes because cluster leaders require more energy to transmit data to the base station (BS). Hence, LEACH uses random selection as the criterion for interchanging cluster leaders. This study was simulation- based, using a tool known as network simulator 2. Three performance matrices that were considered were network lifetime, number of cluster leader per round and energy consumption. The results of this study show that LEACH is energy efficient as only 5% of the total number of nodes can be selected as cluster leaders per round.

c) An Emergent Algorithm for Highly Uniform Cluster Formation (ACE)

This section presents the ACE algorithm, an algorithm that result in highly uniform cluster development that can achieve a packing efficiency close to hexagonal close-packing. The ACE algorithm makes the emergent formation of clusters that are an efficient cover of the network, with significantly less overlap than the cluster created by ^ [an, the ??] exiting algorithm, by using the self-organizing properties of feedback between nodes. The algorithm that this study proposed needs no information of geographic location and needs only a minimal constant amount of communication overhead. In the ACE algorithm a node randomly elects itself as a leader for a particular cluster. This study was simulation-based and was simulated using a tool known as MATLAB. Results show that ACE is fast, robust against packet loss and node failure, and efficient in terms of communication. The literature on homogeneous algorithms shows that there are features that need to be improved
so that homogeneous algorithms can be evaluated in the context of selecting a MKD for WMN. Features such as the number of cluster leaders that are chosen per round in each

Table 2.2.7 Parameters for ACE

Parameters	Value
Name of the protocol	An Emergent Algorithm Highly Uniform Cluster Formation (ACE)
Year	2000
Type Algorithm	Homogeneous
Criteria of Leader selection	Random selection
Tools used for simulation	MATLAB
Performance metrics	Average Cluster size

of the above studies cannot be used for evaluation in the context of selecting a MKD so they must be modified so that they are choosing one leader per round.

Study of the EEHC and ACE algorithm EECS was simulated in MATLAB, which is good for simulating WSN algorithm but not for simulating WMN, to improve the study so that it can be used for evaluation of leader selection algorithms in the context of selecting a MKD of WMN. This study was simulated on Ns2. Performance metrics that are used for these algorithms are not sufficient for MKD selection algorithms. Hence, performance metrics such as leader selection delay need to be added in order for these algorithms to be evaluated in the context of MKD selection for WMN. This metric will be used to compute time taken to select a one cluster leader per round.

2.3.2. Position Based selection

In the study of position based selection, leader selection algorithms are divided into two categories, namely Event based selection algorithms and distance based selection algorithms. Sections i) and ii) discuss both of these position-based selection algorithms.

i) Event based selection

Leader election Protocol like Event-Driven Clustering routing algorithm for WSN (EDC)(Wei ZZ, et al,2004),Distributive energy adaptive clustering protocol for WSN (DEEAC) (Udit S at el. 2007), Energy efficient clustering Algorithm for Event-driven in WSN (EECED (Otgonchimeg B, et al, 2009) were reviewed in a context of WMNs MKD selection. In the event based leader selection algorithms, the BS initiates the process of electing a cluster leader by broadcasting a message that triggers the election process. In Event driven based leader selection, algorithm nodes that are in the place where the event is taking place stand a good chance of being elected as cluster heads. In this kind of leader selection algorithm, hotness value is also used together with remaining energy level to select the cluster leader. The node that has a higher remaining energy and high hotness value will be selected as a cluster head for the network. After the cluster head has been selected it sets the TDMA for normal node. The tables in this section present the parameters for a single event based leader selection algorithm.

a) Energy Efficient Clustering Algorithm (EECED)

The Energy Efficient Clustering Algorithm (EECED) is an Event Driven leader selection algorithm which is intended to extend network lifetime by balancing the energy usage of the node. In this study, nodes that are active and transmitting data have a high probability of being selected as cluster leaders based on remaining energy. The study of EECED is a distributed clustering algorithm . This work was simulation based and Ns2 is a tool that was used for simulating the proposed algorithm. The three performance metrics that were considered were energy consumption, total network remaining energy and network lifetime. The result showed that the proposed algorithm can sustain a balanced energy distribution among nodes in a sensor network and thus extend the network lifetime.

Table 2.2.8 Parameters for EECED

Parameters	Value
Name of the protocol	Energy Efficient Clustering Algorithm (EECED)
Year	2009
Type Algorithm	Event Driven Algorithm
Criteria of Leader selection	The node that is active stands a chance to be elected as a leader. The node that is having highest level of energy become a CH.
Tools used for simulation	Ns2 ,
Performance metrics	Node Average remaining energy, and time percentage of living nodes, energy dissipation rates

b) Event driven clustering routing algorithm(EDC)

Table 2.2.9 Parameters for ACE

Parameters	Value
Name of the protocol	An Event Driven Clustering Routing protocol for WSN (EDC)
Year	2004
Type Algorithm	Event Driven Algorithm
Criteria of Leader selection	Active node and having highest energy becomes a CH
Tools used for simulation	Ns2
Performance metrics	Node Average remaining energy, and time, percentage of living nodes, energy dissipation rates

The energy efficient event driven clustering routing algorithm (EDC) is based on the unique attribute of the event driven data model of WSN. In this leader selection algorithm a node that has a maximum residual energy amongst nodes which are sensing an event that has occurred in the network becomes a cluster leader. When an event of interest occurs on the network all nodes next to that event firstly switch to active state from the sleeping state. EDC is a semi-distributed type leader selection algorithm. The study was simulation based and the tool that was used for simulation is ns2. Three performance metrics were considered, namely network lifetime, Node energy quadratic mean deviation and Node average remaining energy. The study of EDC shows that this SLA can reduce energy consumption and improve evenness of dissipated network energy and has the ability to prolong the network's lifetime.

c) Distributive Energy Efficient Adaptive Cluster (DEEAC) Protocols for Wireless Sensor Networks

The study of DEEAC shows that the areas in the network that have high data generation rates are considered to be hot regions. DEEAC tries to optimise the energy utilization of the network by ensuring that nodes belonging to hot regions have a high chance of being chosen as cluster leaders. The proposed protocol selects cluster leaders based on hotness value and remaining energy. The study was simulation based and the tool that was used for simulation is Ns2. Three performance metrics were considered, namely network lifetime, total energy consumption, and total amount of data received at BS. The results of the DEEAC study show that this SLA is able to prolong network lifetime while delivering more data for the same amount of energy consumption.

From the literature on event driven algorithms, it is evident that there are aspects that need to be improved so that event driven algorithms can be evaluated in the context of selecting a MKD for WMN. Such aspects include the number of cluster leaders that are chosen per round in each above studies cannot be good to be used for evaluation in the context of selecting a MKD and these feature must be modified such that they choosing one leader per

Table 2.2.10 Parameters for DEEAC

Parameters	Value
Name of the protocol	Distributive Energy Efficient Adaptive Cluster Protocols for Wireless Sensor Network (DEEAC)[
Year	2007
Type Algorithm	Event Driven Algorithm
Criteria of Leader selection	High hotness value and remaining energy
Tools used for simulation	Ns2
Performance metrics	Node Average remaining energy, and time, percentage of living nodes, energy dissipation rates

round. Performance metrics that are used for these algorithms are not adequate for the MKD selection algorithm. Hence, a performance metric such as leader selection delay needs to be added for these algorithms to be evaluated in the context of MKD selection for WMN. This metric will be used to calculate time taken for a cluster leader to be selected per round.

ii) Distance based leader selection

The leader election protocol like Energy efficient distance based clustering (EEDBC) (Han Y, et al, 2007), Clustering Routing Protocol on location node in WSN (CRPBLN) (Nurhayati N, et al, 2011), Energy and Distance Based Clustering Protocol for WSN (EDBCP) (Tong. H et al, 2011) were reviewed for this work under distance based leader selection. In distance based leader selection algorithms, the leader is selected based on the distance between the node and the base station. Cluster leader selection algorithms use a common energy model to compute energy consumed by each node on the network. From the energy model it is shown that the nodes that are far from the BS consume more energy in transmitting a packet to the BS, making it important to consider distance when we are selecting a leader for the cluster (Liu Y, et al, 2009). The tables in the section below present parameters for a single distance based leader selection algorithm.

a) Distance Based and Energy leader selection Algorithm

Table2.2.11 Parameters for EDBC

Parameters	Value
Name of the protocol	EDBC
Year	2009
Type Algorithm	Distance Based and Energy leader selection Algorithm
Criteria of Leader selection	Nodes that are near the BS and have high energy level
Tools used for simulation	*MATLAB*
Performance metrics	Node Average remaining energy, and time, percentage of living nodes, energy dissipation rates

The study of EDBC proposed algorithm that considers both distance of each node to the BS and remaining energy when selecting a cluster leader. The EDBC algorithm reduced energy consumption and prolonged network lifetime. This work was simulation based and the tool that was used for simulation was MATLAB. Three performance metrics were considered, namely Node Average remaining energy, percentage of living nodes, and energy dissipation rates. The result of this study showed that EDBC provides better performance and is able to increase the lifetime of the network.

b) Clustering Routing Protocol on location node in WSN (CRPBLN)

In the CRPBLN, the cluster leader selection is based on node distance to the BS and remaining energy. CRPBLN was proposed in order to prolong the life span of the network, by reducing the energy consumption for each node during transmission.

Table2.2.12 Parameters for RPBLN

Parameters	Value
Name of the protocol	Clustering Routing Protocol on location node in WSN
Year	2006
Type Algorithm	Distance based Algorithm
Criteria of Leader selection	Distance Based and Energy leader selection Algorithm
Tools used for simulation	MATLAB
Performance metrics	Node Average remaining energy, and time, percentage of living nodes, energy dissipation rates

The study of CRPBLN was simulation based and the tool that it used for simulation was MATLAB. Three performance metrics were considered, namely Node Average remaining energy, percentage of living nodes, and energy dissipation rates.

c) Energy and Distance Based Clustering Protocol for Wireless Sensor Network

An EDBCP the cluster leader selection is based on node distance to the sink and remaining energy. In this algorithm, the node with a minimal distance to the sink and having high remaining energy has a higher probability of becoming a cluster leader. In the study of EDBCP the cluster leader was selected in a distributed manner. The study was simulation based and the tool that was used for simulation was Ns2. Three performance metrics were considered, namely the number of packets received by sink, percentage of living nodes, and energy dissipation rates. The study showed that the proposed protocol balances the energy consumption among sensor nodes.

Table 2.2.13 Parameters for EDBCP

Parameters	Value
Name of the protocol	Energy and Distance Based Clustering Protocol for Wireless Sensor Network
Year	2011
Type Algorithm	Distance based Algorithm
Criteria of Leader selection	Distance Based and Energy leader selection Algorithm
Tools used for simulation	Ns2
Performance metrics	number of packet received by sink, and time, percentage of living nodes, energy dissipation rates

The literature on distance based leader selection algorithms shows that there are features that need to be enhanced so that distance based leader selection algorithms can be evaluated in the context of selecting a MKD of WMN. Features such as the number of cluster leaders that are chosen per round in each of the above studies cannot be used for evaluation in the context of selecting a MKD, therefore they must be modified in such a way that they choose one leader. Performance metrics that are used for these algorithms are not enough for MKD selection algorithms since the WMN has to be reliable and highly secure at all times. Hence, performance metrics such as leader selection delay need to be added for these algorithms to be evaluated in the context of MKD selection for WMN, and it was this metric which was used to compute time taken to select a one cluster leader per round.

2.4. Selection frameworks for exiting leader selection algorithms

The literature above helped the study to come up with the frameworks that are going to be discussed in this section. The following selection frameworks were employed to identify different leader

Table 2.2.14. Selection framework for heterogeneous Algorithms.

Algorithms	Leader selection process	Optimal CH	Condition for selection
EECS	Distributed	No	Energy
LEACH-C	Centralized	No	Energy
UDCA	Distributed	Yes	Energy

Table 2.2.15 Selection framework for homogeneous Algorithms.

Algorithms	Leader selection process	Optimal CH	Condition for selection
LEACH	Distributed	yes	Energy
ACE	Distributed	No	Energy
EHCA	Distributed	Yes	Energy

Table 2.2.16 Selection Framework for Event Based Algorithms.

Algorithms	Leader selection process	Optimal CH	Condition for selection
EDC	Distributed	yes	Active node and Remaining energy
DEEAC	Distributed	No	Active node and Remaining energy
EECED	Distributed	Yes	Active node and Remaining energy

Table 2.2.17 Selection Framework for Distance Based protocols.

Protocols	Leader selection process	Optimal CH	Condition for election
EDBC	Distributed	yes	Distance
CRPBLN	Distributed	No	Distance and remaining Energy
EDBCP	Distributed	Yes	Distance and remaining Energy

selection algorithms that were going to be evaluated for this study. The process of selecting a leader selection algorithm is guided by the differences and similarities between WSN and WMN that were discussed in section 2.2. The following tables represent the selection frameworks for all leader selection algorithms per category. Table 2.14 shows the heterogeneous LSA selection framework. Based on the Table 2.14, this work has selected heterogeneous leader selection algorithms such as UDCA and EECS to be evaluated in the context of Wireless Mesh network because they meet most of the requirements.

From the selection framework *(Table 2.*15) of the homogeneous energy based leader selection algorithms, it has transpired that LEACH and EHCA meet the requirements that have been stipulated from section 2.2. Both LEACH and EHCA provide energy efficiency which makes them a good protocol to be employed for security purposes.

The selection framework *(Table 2.16)* above for event based protocols revealed that EECED and EECED both have the requirements that have been stipulated in section 2.2. Both EECED and EECED provide energy efficiency which makes them a good protocol to be employed for security purposes in locations that have high power constraints.

From the selection framework *(Table 2.17)* of the distance based protocol, it transpired that EEDBC and EDBCP meet the requirements that have been stipulated in section 2.2. Both EEDBC and EDBCP provide energy efficiency, which makes them a good protocol to be employed for security purposes.

2.5. Summary

The introduction of key concepts such as leader selection algorithm (SLA), classification frameworks, comparison of Wireless Mesh Network (WMN) and Wireless Sensor Network (WSN) helped in deriving a selection framework for choosing leader selection algorithms to be evaluated in the context of selecting a MKD for a WMN. The comparison of WMN and WSN guided the classification and selection frameworks. This chapter reviewed the different leader selection algorithms for other wireless networks. Taxonomy and frameworks have been used to categorise and select leader selection algorithm to be evaluated. The next chapter provides the design view of algorithms that are going to be evaluated for the study. In the next chapter, eight leader selection algorithms (LSAs) that have been chosen for simulation and evaluation are going to be discussed. The pseudo-code and the flowcharts for these eight LSAs are presented in Chapter 3.

CHAPTER 3

SELECTED LEADER SELECTION ALGORITHMS FOR THE STUDY

In Chapter 2, leader selection algorithms were classified into four different categories, namely i) heterogeneous energy based selection, ii) homogeneous energy based selection, iii) event based selection, and iv) distance based selection. From the literature two algorithms have been selected for evaluation per each category. EECS and UDAC have been selected for evaluation under heterogeneous energy based selection. Under homogeneous energy based selection, LEACH and EEHCA have been selected to be evaluated in the context of selecting an MKD for WMNs. EECED and EDC algorithms have been selected for evaluation in the context of selecting an MKD for WMNs. LSAs such as EEDBC and EDBCP have been selected for evaluation in the context of selecting an MKD for WMNs. This chapter discusses the design view of all chosen leader selection algorithms from different categories.

3.1. Heterogeneous Energy Based selection leader Algorithms.

This section discusses the design view of two heterogeneous energy based leader selection algorithms. The study used the flow chart and pseudo code for the design view of each heterogeneous energy based leader selection algorithm.

3.1.1. Energy Efficient clustering Scheme in WSN (EECS)

In EECS, the network is divided into clusters, where each cluster has one cluster head. The cluster head is responsible for gathering information from different nodes on the network and sending it to the base station (BS).

```
Nodei choose num(0<=num <=1)                    //nodes choose number between 1 and 0
If(num<T)                              // check if the number that nodes chosen is </> threshold
        Nodei broadcast_CompetHeadMsg()    // if number is < nodei sends a competeHeadMsg
  Else
         waitCH_AdvMsg()          //else nodes waits for a CH advrtisementMsg
if(nodei_RE > nodeJ_RE)          //if nodei  RE is greater than the neighbours RE   nodei becomes a CH
         nodei=CH && nodej==nonCH
  else
        waitCH_AdvMsg()
        Ch_sendCH_AdvMsg()
Non_CH_ReplyJoinMsg() //When receives a ChAdvMsg non-CHs reply with joining Msg
```

Figure 3.1 Pseudo Code for Energy Efficient Clustering Scheme in WSN

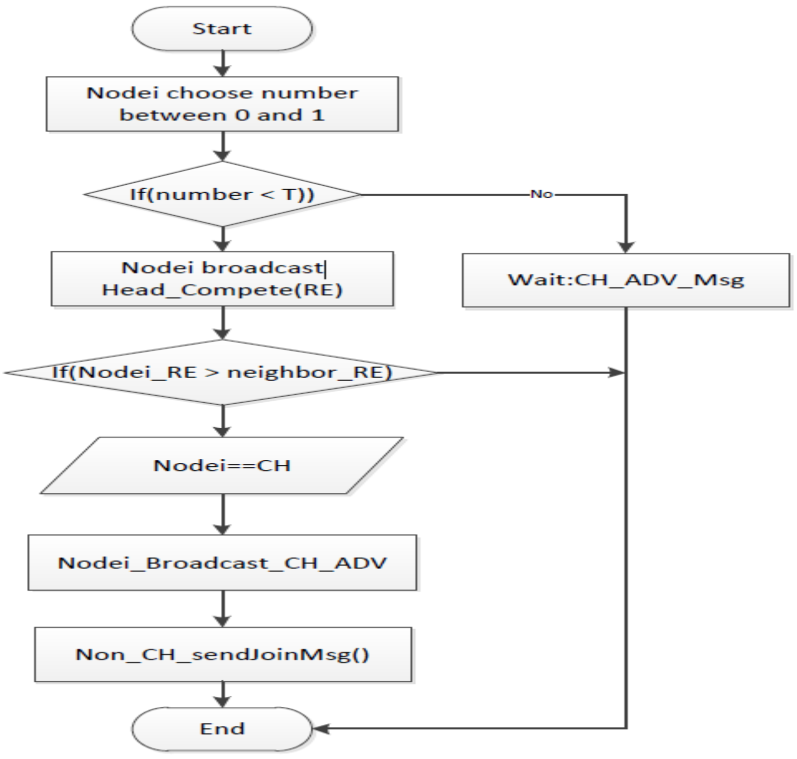

Start
Nodei choose number between 0 and 1
If(number < T))
No
Nodei broadcast Head_Compete(RE)
Wait:CH_ADV_Msg
If(Nodei_RE > neighbor_RE)
Nodei==CH
Nodei_Broadcast_CH_ADV
Non_CH_sendJoinMsg()
End

Figure 3.2 Flowchart for Energy Efficient Clustering Scheme in WSN

The base station send Hello messages to all nodes at a certain power level. Nodes will compute their distance to the base station and this will help the node to select the proper power level to communicate with the base station.

On the cluster head election phase, the nodes become a candidate node with the probability of T, where T is a number between 0 and 1. The candidate nodes broadcast the compete_Head_MSGs within the radio range or RCompete to advertise their interest. The candidate nodes compare their remaining energy with the remaining energy of other nodes within the radius RCompete. If there is a node with higher remaining energy within RCompete the candidate will give up the competition otherwise it will be selected as the cluster head. After the cluster head has been selected it will broadcast head_AD_MSGs over the whole network. When non cluster head nodes receive the Head_AD_MSGs they respond with a joining message to follow the cluster head. The how process of selecting a leader will be taking place periodically.

Figure 3.1 shows the Pseudo code that contains the steps that are followed by the Energy Efficient Clustering Scheme to select the leader. At the initial stage nodes become candidates based on the threshold T. Node_i compares its remaining energy with its neighbour's remaining energy and if the remaining energy of the node_i is greater than the remaining energy of its neighbours that node_i becomes a cluster head for that particular round. Figure 3.2 shows the flow chart for the Energy Efficient clustering Scheme that reflects what is happening at each level of the leader selection protocol. Figure3.2 shows the flow of processes like broadcasting different messages and conditions that are part of selecting a cluster leader.

3.1.2. Energy efficient clustering algorithm for wireless sensor networks (UDCA)

In UDCA, nodes with higher energy levels are the only nodes that can be elected as cluster heads and those nodes will with lower energy levels perform sensing tasks that require low amounts of energy.

```
Start
    BS broadcast REQ()   // BS broadcasts the Request signal
      Node reply()           // all nodes reply with message with their location, ID , and Energy level
      Msg(LOC, ID, EL)            // this is the message with  LOC and EL

        if  (node_EL =highest_E)  // if the node energy level is the highest, BS selects the node as the CH
              BS set node =CH
              BS broadcast CH_ID  // the BS broadcasts the ID of a node that is having the highest EL
         if  (CH_ID =Nodei_ID )         // if nodei_ID is equal to the node_ID broadcasted by BS nodei is
equal to CH
            nodei =CH
        else
             nodei =normal node     // or nodei will be equal to normal node.
          CH broadcast CH_AD // the selected cluster head broadcasts advertisement message so that node
will join
               Normal node Reply()            // normal node replies with a joining messages
                   Join_CH Msg
                   CH: selects CH for next round ()
End
```

Figure 3.3 Pseudo Code for Energy Efficient Clustering Algorithm for Wireless Sensor Network

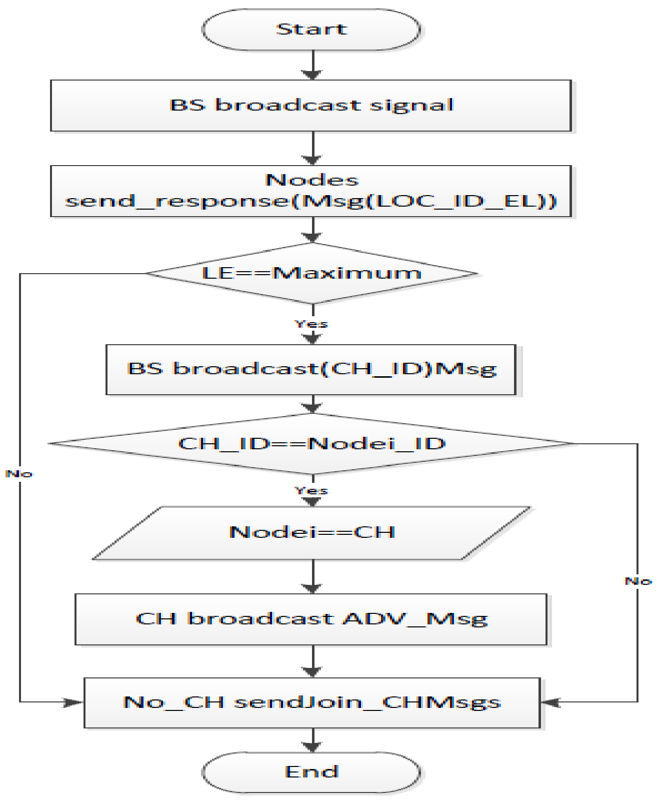


Figure 3.4 Flow Chart for Energy Efficient Clustering Algorithm for Wireless Sensor Network (UDCA)

In the process of cluster head election the base station broadcasts the request message into the network requesting the location, node ID and energy level of each node. When nodes receive this message they reply with a message containing their location, ID, and their remaining energy level. The BS selects one node as a cluster head of the network based on the remaining energy of that selected node. When two or more nodes are having the same energy level the BS uses the smallest/largest ID to break the tie. The BS broadcasts a CH_ID message signal which contains the CH ID and energy level into the network. All nodes on the network determine if the node ID contained in the message matches its own ID which means that the node would become a cluster head and broadcast the CH-Adv message into the network. Other nodes would make the decision to join the cluster based on signal strength, and relay their decision via a joining message to the Cluster Head. After the cluster head receives the join-Msg from other nodes in the network, the Cluster Head selects the probable cluster head for the next round.

Figure3.3 presents the pseudo code that has steps that are used in the Energy efficient clustering algorithm to select a cluster leader based on remaining energy. Figure 3.3 shows all messages being sent between the BS and normal nodes on the network throughout the whole process of selecting a cluster leader.

Figure 3.4 shows a flow chart for UDCA that shows how the BS collects information about other nodes in the network and how the BS selects the cluster leader based on the remaining energy. Flow chart shows how the BS informs the node that it has been selected as a cluster head.

3.2 Homogeneous Energy based selection algorithms

This section discusses the design view of two homogeneous energy based leader selection algorithms. The study uses flow chart and pseudo code for the design view for each homogeneous energy based selection algorithm. This section discusses leader selection algorithms such as ECHA and LEACH. These will be evaluated in the context of MKD selection.

3.2.1. Energy Efficient Homogeneous Clustering Algorithm for WSN (ECHA)

In ECHA, during cluster leader selection process, the BS collects all information about the nodes on the network and then virtually partitions the whole network into zones. In this study it is assumed that all nodes possess equal maximum energy (Emax), and that the node in each zone has a probability p to become a Cluster leader where p = 1/number of nodes in the zone. The BS randomly selects the cluster leader using the grid algorithm. The following figure shows the randomly selected cluster leaders. If a node has been selected as a cluster leader it broadcasts its ID to the network. When a normal node receives a message it sends a joining message to the cluster leader. Figure 3.5 shows the visualisation of the cluster network with one base station and each cluster having one cluster leader. Figure 3.6 depicts the pseudo code for the Energy Efficient Homogeneous Clustering Algorithm; it contains the step by step process of selecting a cluster head. This figure reveals to us that the process of selecting a cluster head is being taken care of by the BS, where the BS divides the network into zones and randomly selects a node as a cluster head.

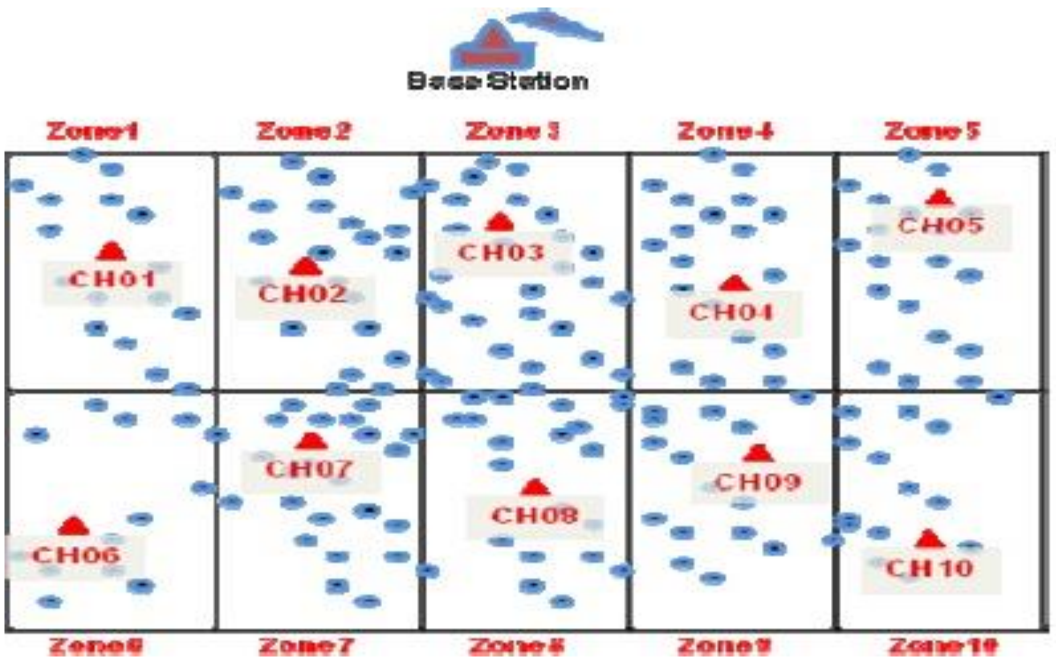


Figure 3.5 Selected Cluster Leaders (Singh SK, et al, 2010)

```
Start
    BS: divide network                          //base station divides the network into zones
    BS:Randomly selects CH()                        //base station randomly selects one node as a CH for one
zone
        If(node i=CH)                                   // if a node has been selected as a CH it broadcasts a
CH the CH-ID
            Nodei broadcasts CH-ID()
       Else
            sendsJoin_Req()                           //normal node sends the joining request to the CH
          normal_node sendsJoin_req()
          CH sends DataSendingSchedule()           //after CH has received a Joining request CH creates
TDMA
        Else
            Ch waits join_Req()
 End
```

Figure 3.6 Pseudo Code for Energy Efficient Homogeneous Clustering Algorithm for WSN

Figure3.7 shows a flow chart for that Energy Efficient Homogeneous clustering algorithm .

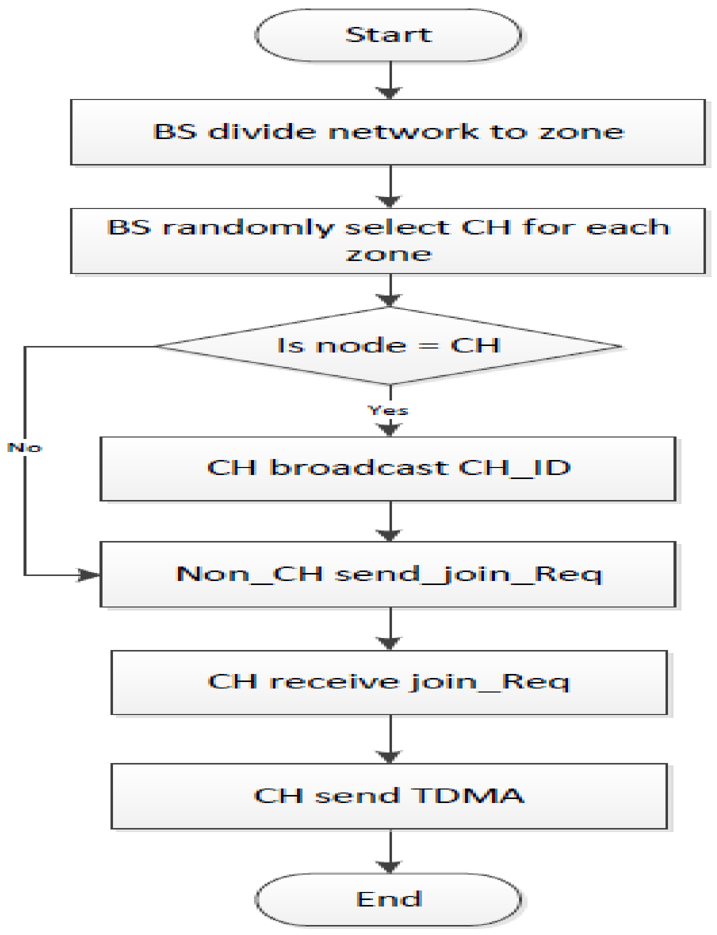


Figure 3.7 Flowchart for Energy Efficient Homogeneous Clustering Algorithm for WSN

3.2.2. Low Energy Adaptive Clustering Hierarchy (LEACH)

In LEACH, all the nodes have 1/p chances to be elected as a cluster leader. The LEACH algorithm stochastically selects cluster heads. In LEACH protocol nodes generate numbers between 0 and 1. A node that generates a number that is less than the threshold T(n) becomes a cluster leader. The threshold is given as follows:

$$T(n) = \frac{p}{1-p\,x(\,r\,mod\;\;1|p)} \qquad \text{(Heinzelman, W, et al, 2000)} \qquad (3.1)$$
$$T(n)=0$$

P is a cluster leader probability, r is the number of the current round and G is the number of nodes that have not been cluster leader in the last 1/p rounds.

```
Start
   Calculate_threshold(T)                                   // Bs calculates threshold

   $T(n) = p/1-p * (r \bmod 1/p)$
      All_nodes Choose_RondNum(0 and 1)          // nodes choose a random number between 0 and 1

  If (randNum < threshold)//if the random number for nodei is less than the threshold nodei becomes a CH
        Nodei=CH
   Else
        Nodei = none_CH  // else a node remains a normal node on the network

 If(nodei=CH) // if nodei is a cluster leader it sends a cluster leader advertisement to the network
       Nodei_Sends CH_AdvMsg()              // formula for calculating threshold

  If(transmission signal is good)  // if the node transmission signal is good the node sends back the join req
     Normal_node_sends JoinMsg()

  Ch_Set_TDMA() // Ch sets the TDMA for normal nodes to send data.

End
```

Figure 3.8 Pseudo Code for Low Energy Adaptive Clustering Hierarchy

The cluster leader is responsible for gathering information from different nodes and sends it to the base station. The cluster leader exhausts its energy supply faster because cluster leaders use more energy than other nodes on the network and this may lead to network failure. Figure 3.8 depicts the Pseudo code for the LEACH protocol which contains the step by step process of selecting cluster heads and the creation of clusters. The Pseudo code shows how to calculate the threshold that is used for randomly selecting a cluster head. Figure3.9 shows a flow chart for the LEACH protocol that clearly depicts the sequence of processes that is directly involved in the random selection of the cluster head in the LEACH protocol. This flow chart also shows the conditions or criteria that lead to a node being selected as a cluster head.

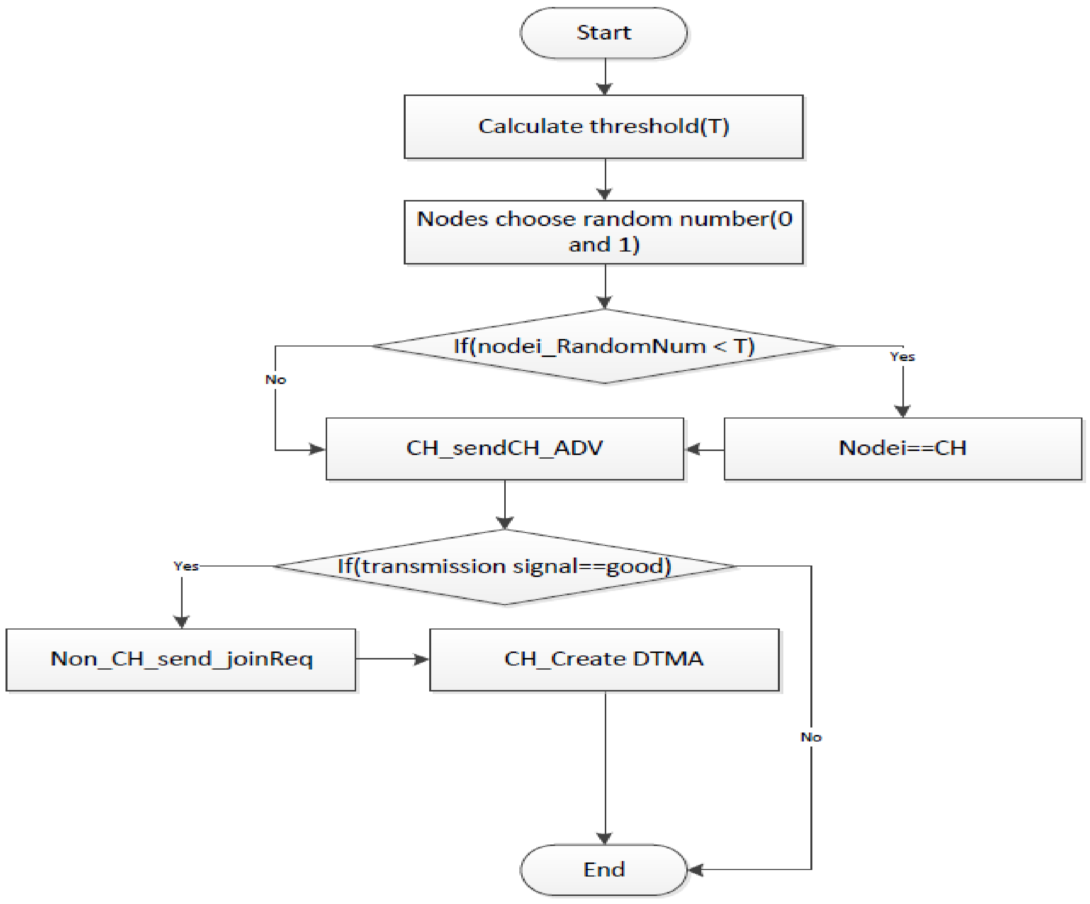


Figure 3.9 Flowchart for Low Energy Adaptive Clustering Hierarchy

3.3 Event Driven Leader Selection Algorithms

Section 3.3 discusses the design view of two event driven leader selection algorithms. In this section flow chart and Pseudo code have been used to construct the design view for each leader selection algorithm. Leader selection algorithms such as EECED and EDC have been analysed.

3.3.1. Energy efficient clustering Algorithm for Event-driven in WSN (EECED)

In the EECED cluster leader selection process, the elector node collects the energy information for its nearest nodes and based on that selects the cluster head. The node that is has a higher remaining energy level stands a good chance of being elected as a cluster leader. When a node has become an elector node it broadcasts a request message to the network with its own remaining energy level information to the nearest nodes.

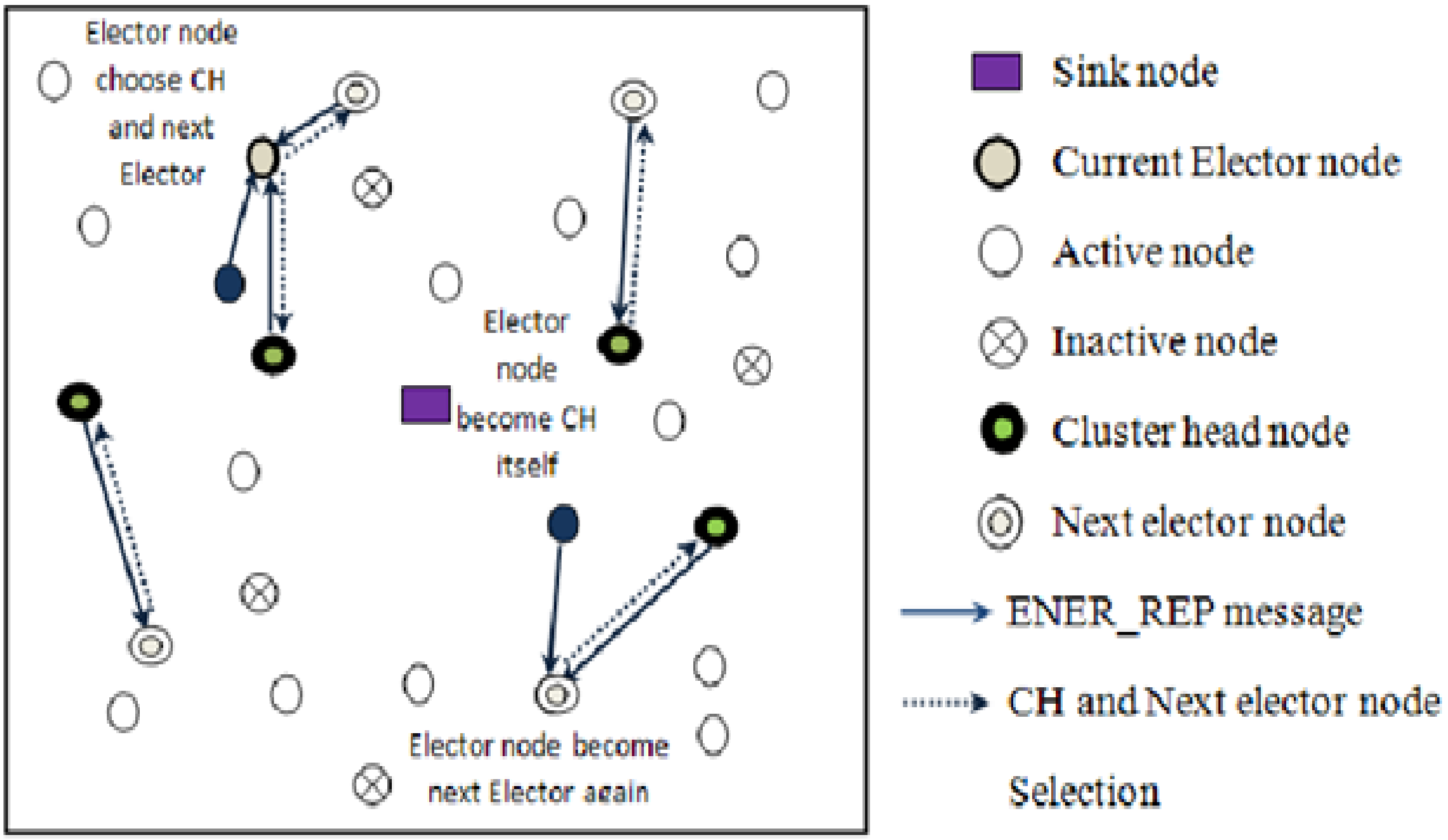

Elector node choose CH and next Elector
Elector node become CH itself
Elector node become next Elector again
Sink node
Current Elector node
Active node
Inactive node
Cluster head node
Next elector node
ENER_REP message
CH and Next elector node Selection

Figure 3.10 Cluster Head and Next Elector Node Selection (Otgonchimeg B, et al, 2009)

When a normal node receives that message, the node compares its remaining energy with the one from the elector and if it has a remaining energy level that is less than the one of the elector node it waits for the CH_ADV message, otherwise it will sends a reply, then the elector node will become an ordinary node. When a cluster head has been elected, it broadcasts an advertising message for a cluster head (CH_ADV) containing the CH ID. When a none CH receives CH_ADV message it selects the most relevant CH based on communication signal strength and then sends a joining request to the CH. Figure 3.10 represents the visualisation for the leader selection process and its stages for the EECED algorithm.

```
Start
  sink chooses ENode()                // BS randomly selects the elector node
  sink broadcasts Elec_ADV            // the BS broadcasts the elector node advertisement message

  if(nodei_ID=Nodei_ID)  // if the advertised node ID is for nodei the nodei becomes an elector node then
      ElectNode sends  Ener_ReqMsg (nodei_EL)  //the elector node sends the En_Req with its own energy
level

 if nodej R_Ener > ElectNode R_Ener   //if node j remaining energy is greater than the one    for electNode
      nodej=CH                         //nodej with becomes a CH
 else
      nodejWaits_for CH_AvdMsg;        // it will wait for a CH advertisement
      nodei selects node with higher EL // Elector node selects node with remaining energy and selects it as a
CH

 if(nodej=CH)
     sendCH_AdvMsg(nodej_ID)       // electNode broadcasts the CH advertisement message
     none_CH  sendReplyMsg(joinCH)   // noneCH sends joining messages to join CH
 CH set_TDMA()                        //set TDMA for each node
End
```

Figure 3.11 Pseudo Code for Energy Efficient clustering Algorithm for Event Driven in WSN

Figure 3.10 depicts how the elector node collects node information on the network and how the cluster head is being selected, Figure 3.11 shows the pseudo code for EECED. The EECED Pseudo code symbolically shows how the whole process of selecting the new cluster head. Figure3.12 shows a flow chart for the EECED protocol that clearly depicts the sequence of processes that is involved in the selection of the cluster head in the EECED protocol. From the flow chart it can be seen that the BS randomly selects the elector node. This flow chart also shows the conditions or criteria that lead to a node being selected as a cluster head.

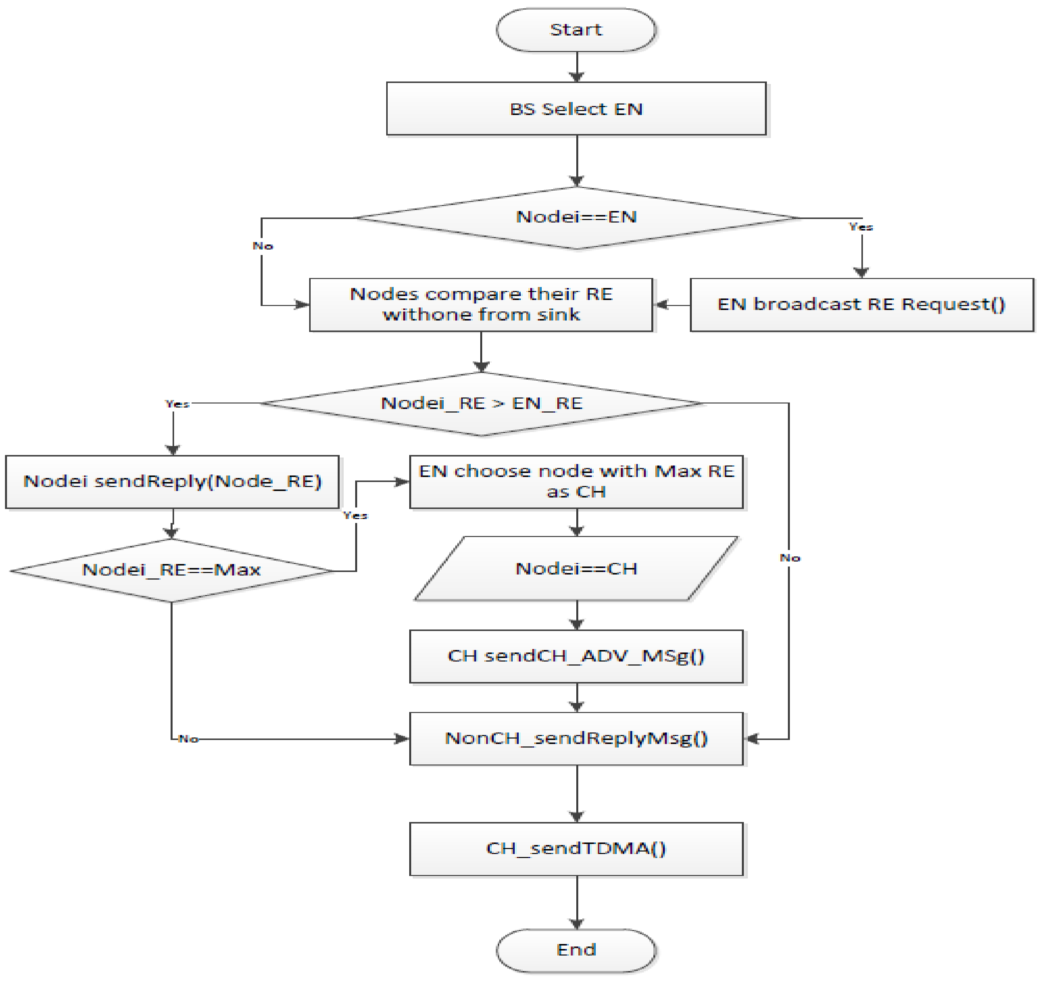


Figure 3.12 Flowchart for Energy Efficient Clustering Algorithm for EventDriven ^^ in WSN

3.3.2. Event-Driven Clustering Routing Algorithm for Wireless Sensor Networks (EDC)

When the clustering based routing protocol is used in the event driven data type a lot of energy can be saved. In EDC nodes can have different states on the network; a node can be at the sleeping stage or at the active stage. When a node senses an event it changes from a sleeping state to an active state so that it can participate in the leader election process. The base station broadcasts a control message with a threshold value on it to the network. This message makes the node change its current state to another state. When an event of interest takes place sensor nodes on the network broadcast their remaining energy to the nearest gateway node.

```
Start
    BS: BroadcastCPk(T)   // BS Broadcasts control Packet with Threshold is where
       T=p/1-p*(r mode 1/p)  ///used to     //change node state
       nodesSave CPk(T)    // nodes save the control packet that they received on the CACHE
        Nodes SwitchState(from Active to Sleep)     // nodes change their states to sleep

       if(Event == True)      // if there is an Even nodes change their state to be active
              Node SwitchState(From sleep to Active)
               Nodes Broadcast_RE()               // nodes broadcast their remaining energy to BS

         if(nodei_state == Active && nodei_RE == Max)  //if a node is having a maximum //remaining energy
                                                            //and  Active state  come a CH
                Nodei == CH
             BS_SendCH_INF() //BS informs nodes that has been selected  their new status
             CH_BroadcastCH_AdvMsg()
       noneCHNodesSend_JoiningMsg()                  // noneCh joins the CH
      CHISetTDMA()                                   // CH set TDMA for each node on the cluster
End
```

Figure 3.13 Pseudo Code for Event-Driven Clustering Routing Algorithm for Wireless Sensor Networks (EDC)

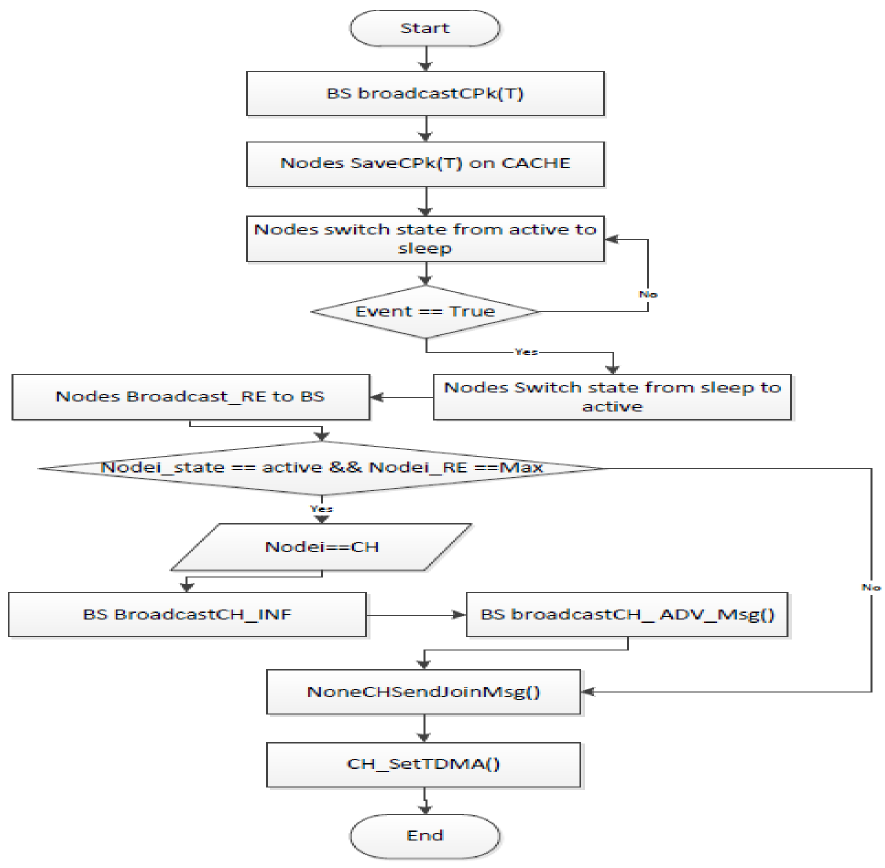

Start
BS broadcastCPk(T)
Nodes SaveCPk(T) on CACHE
Nodes switch state from active to sleep
Event == True
No
Yes
Nodes Switch state from sleep to active
Nodes Broadcast_RE to BS
Nodei_state == active && Nodei_RE ==Max
Yes
No
Nodei==CH
BS BroadcastCH_INF
BS broadcastCH_ ADV_Msg()
NoneCHSendJoinMsg()
CH_SetTDMA()
End

Figure 3.14 Flowchart for Event Driven Clustering Routing Algorithm for Wireless Sensor Networks (EDC)

One of the Gateways takes the responsibility of collecting energy information from all active nodes on the network. The Gateway selects the number of Cluster leads from active nodes based on their remaining energy, with the node with the highest remaining energy level being selected as a cluster head. After the BS has elected the Cluster head it then broadcasts the information of cluster heads on the network, and the elected node (CH) will then broadcast the advertising message to the network. The normal node will then choose to join the cluster based on the transmission power or communication power for the advertisement message.

Figure 3.13 shows the pseudo code for the EDC protocol that contains steps that are followed during the cluster head selection process in the event-driven environment. The pseudo code shows how the threshold is calculated and how nodes change from one state to another. Figure 3.14 shows a flow chart for the EDC protocol that clearly depicts the sequence of processes that is involved in the selection of the cluster head in the EDC protocol. From the flow chart, it can be seen that the BS broadcasts the control message with the threshold T. This flow chart also shows the conditions that lead to a node being selected as a cluster head.

3.4. Distance based selection

This section discusses the design view of two distance based selection algorithms. The study used flow chart and pseudo code for the design view for each distance based selection algorithm. This section discusses leader selection algorithms such as EDBC and EDBCP which are going to be evaluated in the context of MKD selection.

3.4.1. Energy and Distance Based Protocol for WSN (EDBC)

In EDBC, the distance between nodes and the base station and remaining energy are both used as criteria for selecting a cluster leader. The node with the greatest distance from the BS and has low remaining energy has a minimum chance of being elected as cluster head. The node that is closer to the base station needs less energy to transmit data to the BS than the node that is far from the BS. The criteria of using distance and energy to select a leader help to select the node that is closest to the BS and has highest remaining energy.

In this protocol, the network is divided into concentric circular segments around the BS. During this process of partitioning some segments are located closer to BS and some segments are located far from the BS. The nodes that are in the closer segment have a high probability of being selected as cluster leader. At the same time the nodes that have more energy in different segments have a bigger probability of being elected as cluster leaders. It is assumed that nodes know the distance between themselves and the BS. The innermost segment has the smallest index, in the segment j, node i that has a possibility of becoming a cluster-head at round r with below threshold:

$$T(n) = \frac{p}{(1-p) r mod \frac{1}{p}} + \left(\frac{m+1}{2} - j\right) * \left[\left(\frac{En-curr}{En-max}\right)^{\frac{m+1}{2}-1} + \left(\frac{rs}{epoch}\right)\right] \quad \text{(Hn Y. et al, 2007)} \quad (3.2)$$

In this equation j is the segment number, m is the total number of segments in the network field, En-curr and En-max are current energy and initial energy of each node, and rs is the number of rounds in which a node has not been cluster-head. Thus, the node that has a highest remaining energy and that is near to BS have higher chances to become CH because of a higher threshold. Figure 3.15 depicts the pseudo code for the Energy and Distance Based Protocol that contains steps that are followed during cluster head selection process. The pseudo code shows how node compete during cluster head selection.

```
Start

  Compute T                               //compute Threshold
    where T =p/1-p*(r mode 1/p)
    selectRandNumb[0>=Num>=1] // nodes select random number between 1  and 0

   if(nodei_num < T)                      //nodej compares number with a Threshold
         Nodei=Candidate
        nodeiBroadcast_INF(p) // node j broadcasts information about its status
        nodej   receive INF from nodei
     if(nodej_Ctable==Empty)             //if nodej Candidate Table is empty
           Nodej==CH
    Else
          Compute_AbilityToCompt()                  //nodej computes ability to compete for other nodes
  if(nodex_AbilityToCompt()==Max)                     //if nodex Ability to compute is equal to //Maximum
        Nodex==Ch && Nodej==normal_no)  // nodex will become CH and Nodej will //remain as nonCH
        nodeBroadcast_CH_ADVMsg(p)              // CH will send CH_ADVMsg
        nodeSendJoiningMsg()                     // and nod will send joining Msg
End
```

Figure 3.15 Pseudo Code Energy and Distance Based Protocol for WSN

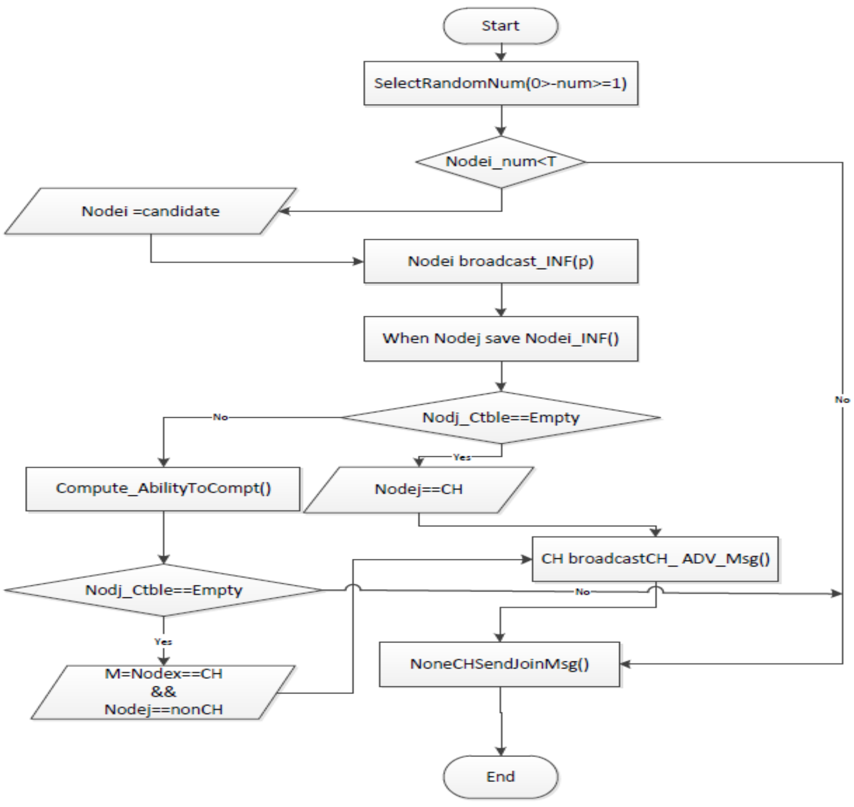


Figure 3.16 Flowchart for Energy and Distance Based Protocol for WSN

Figure 3.16 shows the flow chart for the Energy and Distance based protocol that contain steps that are followed during cluster head selection process. Figure 3.9 presents all methods and conditions that are involved during cluster head selection process.

3.4.2. Energy and Distance Based Clustering Protocol for Wireless Sensor Network (EDBCP)

EDBCP was proposed in order to prolong the life span of the network, by reducing the energy consumption for each node during transmission. In EDBCP, each node randomly selects a number between 0 and 1. If there is a node which has selected a number that is smaller than the threshold T, where T is denoted by the formulary below, that node becomes a candidate for that particular round.

$$T(i) = \begin{cases} \frac{p}{1 - p\times[r \bmod (1/p)]} \times \frac{Eres}{Eini} & i \in G \\ 0 & else \end{cases} \qquad \text{(Tong H. et al, 2007)} \qquad (3.3)$$

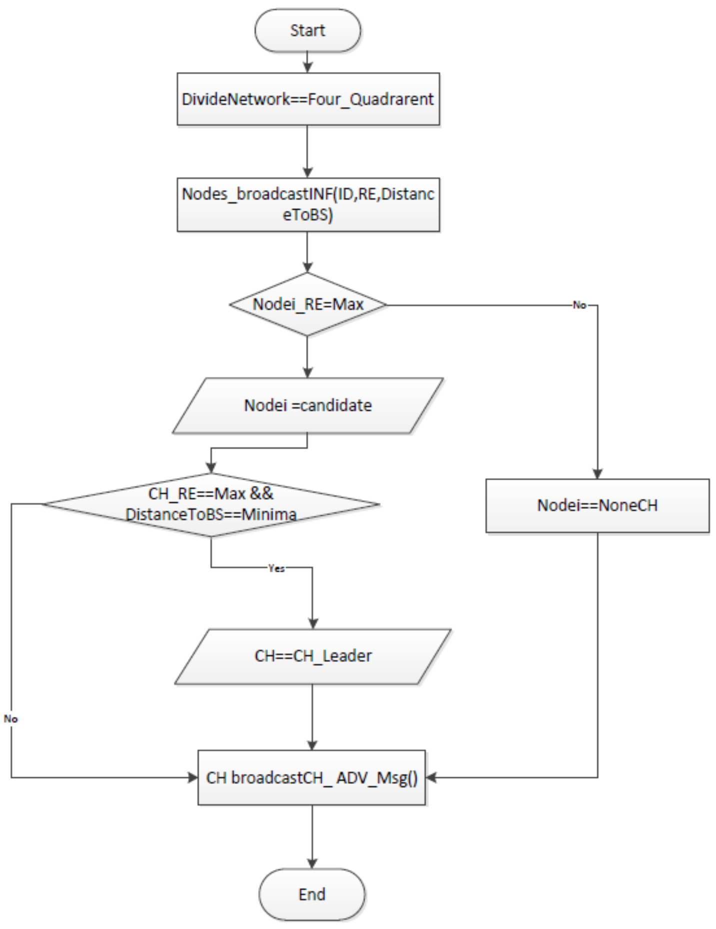
Where Eres is the current remaining energy of node i and Eini is the initial energy for node i during cluster head selection candidate nodes broadcast Information Messages (INFs) to all other nodes. The INF message contains node ID, level of remaining energy and node's distance from the base station. When nodes receive the INF messages from other nodes, they save them to an election information table. If the election information table of the candidate node contains its own INF message only, a candidate elects itself to be cluster head. In EDBCP nodes transmit to the BS through the Cluster leader instead of direct transmission.

Greed algorithm was used to chain the Cluster leader and to use the multi-hop transmission of data from one node to another before it reaches the Cluster leader. After nodes have transmitted data to their cluster leader, the cluster leader aggregates the data and transmits it to their Cluster head leader in multi-hop fashion and then the cluster head leader aggregates

```
Star
    DividNetwork=4 Quadrant       // network is divided into four quadrants
    Nodes_Broadcast_INF(ID,RE,Distance //nodes broadcast their information about ID, RE and Distance to
                                                    //BS   nested if statement
         if(nodei_RE==Max)                          //if nodei has the maximum remaining energy it becomes a CH
       Nodei==CH
       if(CH_RE==Max && CH_DistanceToBS==Minimal)  // if the Cluster head i remaining energy is max
                                                     // and its distance to BS is minimal CH becomes a CH leader
      CH == CH_Leader
      CH_Broadcast_CH_ADVMsg()                                // CH broadcast the CH advertising message
      normalNode_SendJoinReq()                       // non CH send their joining request to the CH
 CHSetTDMA()                                         //CH set the TDMA for each node on the network

End
```

Figure 3.17 Pseudo Code for Energy and Distance Based Clustering Protocol for Wireless Sensor Network



Start
DivideNetwork==Four_Quadrarent
Nodes_broadcastINF(ID,RE,DistanceToBS)
Nodei_RE=Max
No
Nodei =candidate
CH_RE==Max && DistanceToBS==Minima
Yes
No
Nodei==NoneCH
CH==CH_Leader
CH broadcastCH_ ADV_Msg()
End

Figure 3.18 Flowchart code for Energy and Distance Based Clustering Protocol for Wireless Sensor Network

data for other cluster leaders and transmits it to the BS. In EDBCP the network is divided into four quadrants, each quadrant having two clusters. The node that stands a good chance of being selected as a cluster leader is a node with high remaining energy and shorter distance to the BS, and the Cluster head which has the highest remaining energy becomes the cluster head leader. All nodes broadcast their information about node Id, Node Energy and node Distance to the BS. Figure 3.17 presents the pseudo code for an Energy and Distance Based Clustering protocol that contains the steps that are followed to select a cluster head. The Pseudo code shows how nodes broadcast the information to the network and how the cluster head gets selected. *Figure3.10* shows the flow chart for EDBCP that contains the steps that are followed during the cluster head selection process.

3.5 Summary

This chapter has discussed the design view of eight leader selection algorithms that were selected after conducting a review (see chapter 2) of different existing leader selection protocols. Chapter 2 discussed in detail how these eight leader selection protocols were selected over the other leader selection protocols. The study has presented and discussed the pseudo-code and the flowcharts for these eight leader selection protocols. These pseudo-code and flowcharts help us when hard coding the leader selection. Chapter 4 presents and discusses the simulation results for the eight leader selection protocols.

CHAPTER 4

PERFORMANCE EVALUATION OF SELECTED LEADER SELECTION ALGORITHMS

4.1. Introduction

A number of Leader Selection Algorithms have been proposed for clustering purposes in other wireless networks such as wireless sensor networks. Chapter three discussed eight leader selection algorithms selected for evaluation. A consistent comparison and evaluation of different leader selection algorithms in the context of selecting an MKD for WMNs is achieved by keeping the simulation environment and parameters the same for all of this study simulations. Simulation environment, evaluation parameters, and WMNs experimental setup are presented and described in this chapter. The crux of this chapter is the simulation experiments and the results of the eight Leader Selection Algorithms selected for evaluation.

The next section presents the simulation setup details that were used to conduct the experiments. Section 4.3 describes the evaluation parameters used for this study, while the obtained results for the simulation experiments are outlined and discussed in Section 4.4.

4.2. Simulation Environment

Leader Selection Algorithms (LSAs) are originally intended to select many Cluster Heads (CHs) since they are mainly designed for network clustering. For the idea of this study, the selected LSAs were amended so that only one CH is selected. This limitation allows for the selection of only one mesh backbone device to serve as a replacement MKD. Thus, the IEEE 802.11s specification that there should exist only one MKD will not be violated. Simulation tools such as OMNET++, Network Simulator (NS2), and MATLAB are tools that can be used for simulating networks. The simulation environment used for this research work is made up of a set of extensions designed for both static and mobile wireless networks. Other researchers have made wide use of these extensions and the release of the standard VINT which leads to the release of NS-2, was as a result of the adoption of a static version for the wireless networks extensions.

The Network Simulator version 2.35 (NS2) software running on the Ubuntu 12.04 operating system was used to conduct an extensive simulation for this study. NS2 is an open-source event-driven simulator tool that was designed particularly for research in Computer communication networks (Fall. K et al, 2008). NS2 can be used to simulate both wired and wireless networks and is primarily Linux based and NS2 contains modules for numerous network components such as application, MAC, routing and transport layer protocols. NS2 uses two languages, namely, an object oriented simulator (written in C++), and an OTcl (an object oriented extension of Tcl language) interpreter used to execute user's command scripts (www.isi.edu/nanam/ns).

Various network sizes, ranging from 50 to 500 wireless nodes were statically spread over a rectangular 1000m x 1000m flat space for 1000s of simulated time. The detailed trace files generated from the various simulation experiments were stored and analysed using an AWK script, while Microsoft Excel and GNU plot were used to plot the graphs.

Eight Leader Selection Algorithms (EDBC, EDBCP, EEHCA, EDC, EECED, EECS, LEACH and UDAC) were simulated and the results were evaluated using four metrics, so as to be able to come up with efficient LSAs that can be used in WMNs.

Table 4.4.1 Simulation Parameters for all the Experiments

Parameters	**Environment**
Number of Nodes (nn)	50-500 nodes
Number of Rounds	1-15
Network Area	1000m x 1000m
Simulation Time	1000 seconds
Initial Energy	5.0 Joule
Transmission Power	0.6 watts
Receiving Power	0.3 watts
Idle and Transition Power	0.2 watts
Nodes Movement	Static
Performance metrics	Node Average remaining energy, percentage of living nodes, energy dissipation rates

The four metrics used for evaluation in this work were Leader Selection Delay, Energy Consumption Rate, Network Average Energy and Communication Overhead. Table 4.1 summarises the simulation parameters that were used for all the experiments in this study. This section presents the parameters used to evaluate the selected Leader Selection Algorithms in the context of a Wireless Mesh Network. The following measurement procedures were used for each of the metrics being measured.

4.3.1 Communication Overhead

Communication Overhead refers to the sum of the total number of data packets sent and the total packets received between the nodes in the network during the process of selecting a cluster leader. This metric is used to compute the total communication cost between the nodes in the network. The lower the communication overhead value, the better the algorithm performance. The communication overhead is calculated using the formula below:

$$\text{Communication Overhead} = \sum_{0}^{n} \text{SentMessages} + \text{ReceivedMessages} \quad (4.1$$

4.3.2 Leader Selection Delay

Leader Selection Delay is the time taken for Leader Selection Algorithms to successfully select one node as a cluster head. It is calculated based on the time taken for all events to exchange messages between nodes on the network. This time ends when the selected node sends an advertising message for a cluster head. The Leader selection delay metric will help us to identify which Leader Selection Algorithm will take the minimal time to select a leader, which is very important for selecting an MKD in WMN, since MKD is meant to perform security measures such as authentication of new nodes on the network. The lower the leader selection delay value, the better the algorithm performance.

4.3.3 Energy Consumption Rate

Energy consumption rate refers to the rate at which energy is being consumed by the nodes in the course of selecting a new cluster leader. The Energy consumption rate can be the result of different types of energy consuming events such as packet sending, packet receiving and packet routing. Also, different network node states such as sleeping state, active state and idle state consume a certain amount of energy. Hence, the lower the energy consumed by the node, the better the performance.

4.3.4 Network Average Remaining Energy

Network Average Remaining Energy is determined by the sum of remaining energy for all nodes in the network over the number of nodes in the network. This metric will help us to find out which leader selection algorithm consumes most energy per leader selection round.

The higher the remaining energy value, the better the algorithm performance. The network average energy is calculated using the formula below:

$$\text{NetworkAverageRemainingEnergy} = \frac{\sum_{0}^{n} \text{remainingEnergy}}{n} \qquad (4.2)$$

4.4. Simulation Experiments and Results

This section presents the results of the experiments that were carried out. The simulation parameters used for various experiments are given in Table 4.1. Each of the reported results in the following subsections is the average of five experiments for each scenario that was considered.

4.4.1 Experiment I: Communication Overhead

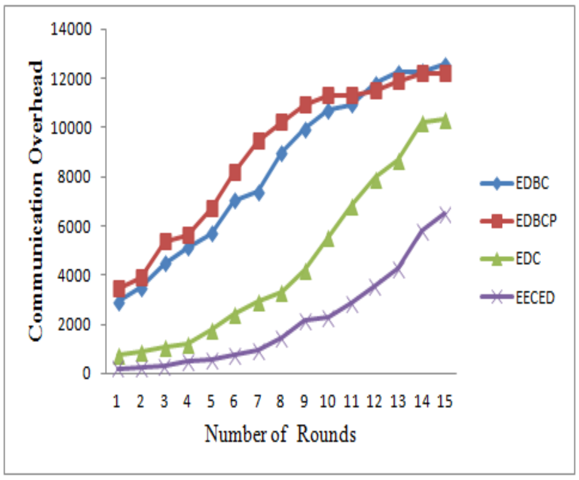
This section present results of the experiments that were carried out to investigate the total Communication Overhead cost of the selected eight LSAs when subjected to different leader selection Rounds and various Network sizes.

The next two sub-sections i) and ii) present the results of the effects of Rounds and Network sizes respectively on communication overhead.

i) The Effect of Rounds on Communication Overhead

The purpose of this experiment was to determine the communication overhead cost for both the Energy-based and the position-based LSAs when respectively subjected to various leader selection rounds. This metric is considered in order to find out which Leader Selection Algorithm incurs low communication cost among the nodes in the network, while selecting a cluster leader after every round. Figures 4.1 and 4.2 depict the results of the communication overhead cost for both the Position-based and Energy-based LSAs, when respectively subjected to the various leader selection rounds. It can be observed from Figure 4.1 that the communication overhead increases gradually as the number of rounds increases linearly. It can also be observed that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. The low total communication overhead cost incurred by the event-based leader selection algorithms can be attributed to the shorter path lengths being traversed between the nodes and the base station. This shorter path length prevents the event-based algorithms from congestion which could arise from control message aggregation queuing at the base station in order to select the leader. The poor performance of distance-based LSAs can be attributed to the congestion caused by the continuous cluster leader selection process, in which all nodes in the network keep sending and receiving control messages from the base station. As the number of hops between the nodes and base station increases, the communication links get broken in communication paths, and this often leads to the creation of more control message packets, thereby increasing the overhead cost.

In Figure 4.2, it can be observed that both the homogeneous- and heterogeneous-based LSAs communication overhead increase gradually as the number of rounds increases linearly. It can also be observed that the Heterogeneous-based (EECS and UDAC) LSAs outperform the Homogeneous-based LSAs. The poor performance achieved by the homogeneous-based LSAs can be attributed to the random selection of leader by the homogeneous-based LSAs. Based on random selection of leader by homogeneous-based LSAs, nodes with low remaining energy can be selected as a leader. Hence, it compromises the entire network reliability and increases the network communication overhead cost, since a new leader has to be selected every time the current leader fails.



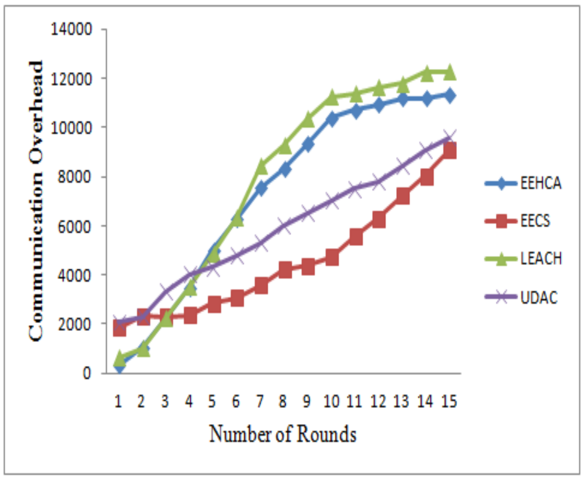


Figure 4.1 Effect of Rounds on Position-based Communication Overhead

Figure 4.2 Effect of Rounds on Energy-based Communication Overhead

ii) The Effect of Network Size on Communication Overhead

The purpose of this experiment was to determine the communication overhead cost for both the Energy-based and the position-based LSAs when respectively subjected to various network sizes. This metric is considered in order to find out which Leader Selection Algorithms incur low communication overhead cost among the nodes in the network in the course of selecting a cluster leader when subjected to various network sizes.

Figures 4.3 and 4.4 depict the results of the communication overhead cost for both the Position-based and Energy-based LSAs when respectively subjected to the various network sizes. It can be observed from Figure 4.3 that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. It can also be observed that the event-based algorithm (EDC and EECED) has a very low communication overhead cost and the increase in network size does not really affect the event-based LSAs. The low communication overhead cost incurred by the Event-based leader selection algorithms can be attributed to the decrease in leader selection update time. In event-based LSAs, leaders are being selected only when there is an event that has occurred and it is only the nodes around that event that will participate in the process of leader

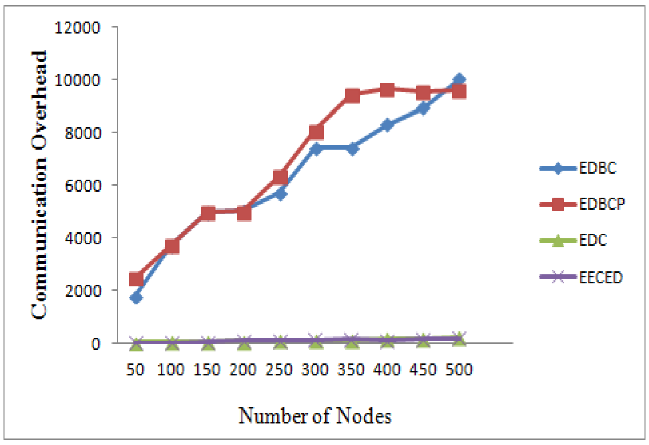


Figure 4.3 Effect of Network Size on Position-based Communication Overhead

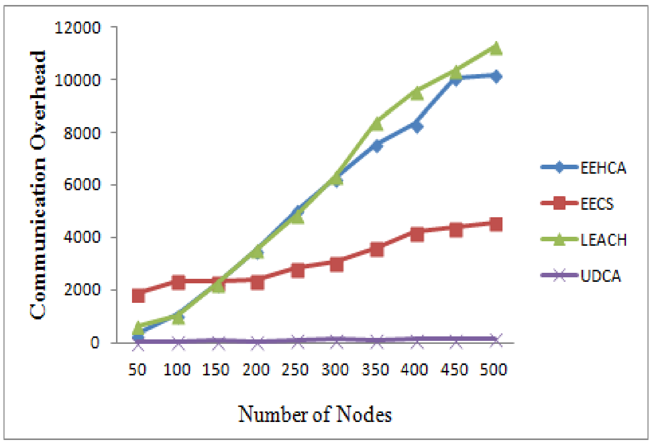


Figure 4.4 Effect of Network Size on Energy-based Communication Overhead

selection; hence, it decreases the time it takes to select another leader and this in turn reduces the communication overhead cost (Kori and Baghel, 2013). However, the poor performance of distance-based LSAs can be attributed to congestion and the increasing update time. Distance-based LSAs select cluster leaders continuously and all the nodes in the network participate in the selection process, hence it increases the leader selection update time significantly, which in turn results in high communication overhead cost (Kori and Baghel, 2013).

In Figure 4.4, it can be observed that the Heterogeneous-based (EECS and UDAC) LSAs outperform the Homogeneous-based LSAs. It can also be observed that when the numbers of nodes are 50 and 100, the communication overhead for Homogeneous-based algorithms were lower than that of EECS However, as the number of nodes increases, the homogeneous algorithms begin to incur higher communication overhead than other LSAs considered. The poor performance of the homogeneous-based LSAs can be attributed to the random selection of leader by the homogeneous-based LSAs. Based on random selection of leader by homogeneous-based LSAs, nodes with low remaining energy can be selected as a leader. Hence, it compromises the entire network reliability and increases the network communication overhead cost, since a new leader has to be selected every time the current leader fails.

4.4.2 Experiment II: Leader Selection Delay

This section presents the results of the experiments that were carried out to investigate the average delay incurred by the selected eight LSAs when subjected to different leader selection Rounds and various Network sizes. The lower the leader selection delay, the better the network performance.

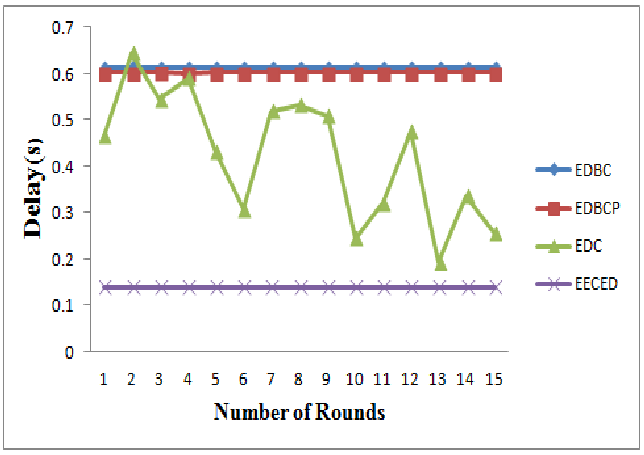
The next two sub-sections i) and ii) present the results of the effects of Rounds and Network sizes on leader selection delay respectively.

i) The Effect of Rounds on Leader Selection Delay

The purpose of this experiment was to determine the Leader Selection Delay for both the Energy-Based and Position-Based LSAs when subjected to different leader selection rounds. This metric is considered in order to find out which Leader Selection Algorithm experiences more delay in the course of selecting a leader after every round.

Figures 4.5 and 4.6 depict the results of the average network leader selection delay for both the Position-based and the Energy based, when respectively subjected to the various leader selection rounds. High delay decreases the overall performance of the network, hence an optimal leader selection algorithm for WMNs should have low delay. The delay was measured in seconds (s).

It could be seen from Figure 4.5 that the Event-based (EECED and EDC) leader selection algorithms outperformed the Distance-based (EDBCP and EDBC) leader selection algorithms. The better performance achieved by the event-based leader selection algorithms can be attributed to the leader selection process. In both event-based and distance-based leader selection process, the base station broadcast the control message to the entire network.



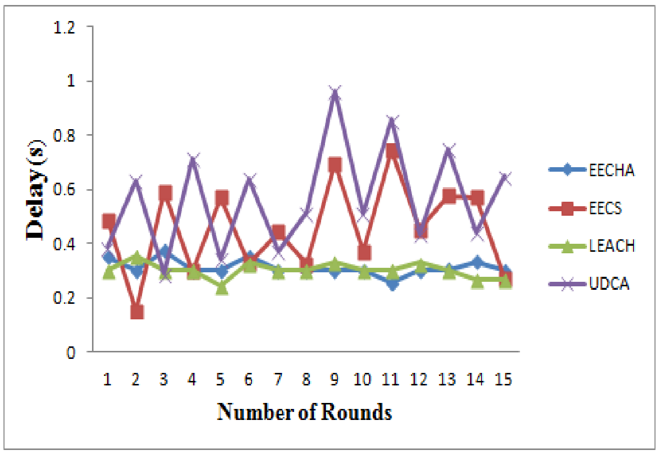


Figure 4.5 Effect of Rounds on Position-based Leader Selection Delay

Figure 4.6 Effect of Rounds on Energy-based Leader Selection Delay

However, in event-based leader selection processes, the base station calculates the threshold value for all the active nodes in the Network and chooses the leader, based on their threshold value. In distance-based leader selection processes, each of the nodes calculates their threshold value and sends to the base station, which will now select the leader for the next round. Based on this process, the distance-based selection process incurs more time, which in turn causes more delay, as observed in Figure 4.5.

In Figure 4.6, it can be observed that both the LEACH and EECHA of Homogeneous-based leader selection algorithms outperform Heterogeneous-based algorithms. The better performance average delay value achieved by the homogeneous-based leader selection algorithm can be attributed to the fact that the leader has been selected at random, which does not require any specific process, as against that of Heterogeneous-based, which consists of three different phases for selecting a leader. Hence, each of these phases introduces some delay, which accumulates and leads to the poor and inconsistent behaviour of the heterogeneous-based leader selection algorithms.

ii) The Effect of Network size on Leader Selection Delay

The purpose of this experiment was to determine the effect of various network sizes on the average network Leader Selection Delay for both the energy based and position based LSAs. This metric is considered, in order to investigate which of the Leader Selection Algorithms incur more time in the course of selecting a leader when subjected to various network sizes.

Figures 4.7 and 4.8 depict the results of the network leader selection average delay for both the Position-based and the Energy-based, when respectively subjected to the various network sizes.

It can be observed from Figure 4.7 that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. It can also be observed that the two event-based algorithms (EDC and EECED) behave in the same way. The better performance value achieved by the event-based leader selection algorithms can be attributed to the event-based leader selection process. The Event-based leader selection process only selects a leader when an event occurs, and it is only the nodes around that event that will participate in the process of selecting a leader at that particular given time, which prevents the network from congestion and high interference among the network nodes, which usually results in high delay. However, in the distance-based leader selection process, cluster leader are being selected continuously and all the nodes participate in the selection of cluster leader by calculating and sending their respective control messages to the base station. The process of calculating and sending of control messages to the base station by the nodes incurs some congestion and interference among the participating nodes, which results into high delay value experienced by the distance-based (EDBC and EDBCP) leader selection algorithms. In Figure 4.8, it can be observed that the EECS of heterogeneous-based leader selection algorithms outperform the other three algorithms (UDCA, LEACH and EEHCA) considered.

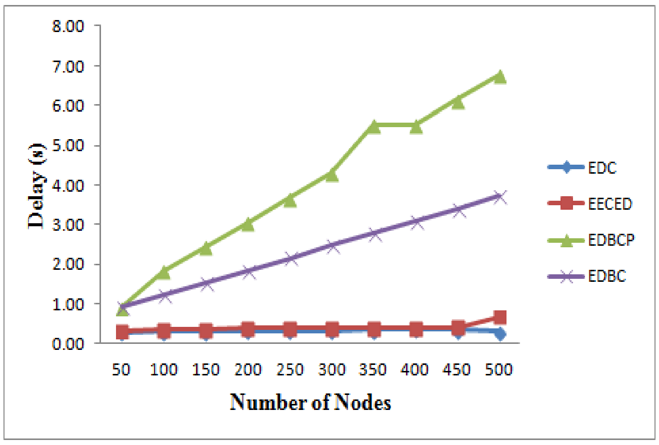


Figure 4.7 Effect of Network Size on Position-based Leader Selection Delay

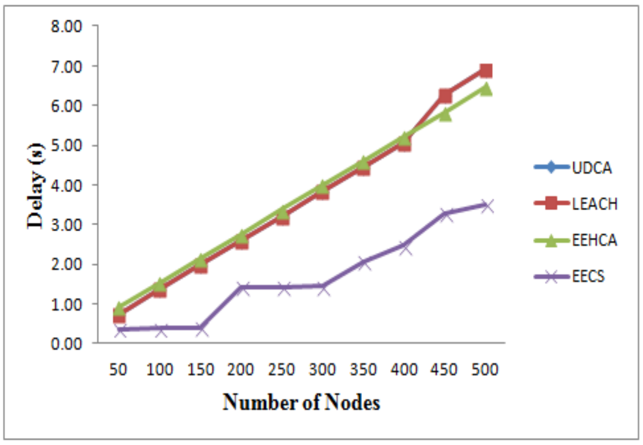


Figure 4.8 Effect of Network Size on Energy-based Leader Selection Delay

This work shows that UDCA, LEACH and EEHCA behave the same way. The better performance achieved by EECS leader selection algorithms in terms of low average delay value can be attributed to the optimal cluster head selection behaviour of the four energy-based LSAs considered. EECS always create more clusters as the need arise, which reduces the burden on the cluster head while transmitting the control message to the base station. However, the remaining three leader selection algorithms (UDCA, LEACH and EEHCA) always optimise the number of clusters by creating a smaller number of clusters and this leads to overburden on the cluster head through congestion, which eventually leads to the high delay value incurred by the three algorithms.

4.4.3 Experiment III: Network Average Remaining Energy

This section presents the results of the experiments that were carried out to investigate the average remaining energy for the selected eight LSAs when subjected to different leader selection Rounds and various Network sizes. The higher the network average remaining energy, the better the algorithm performance.

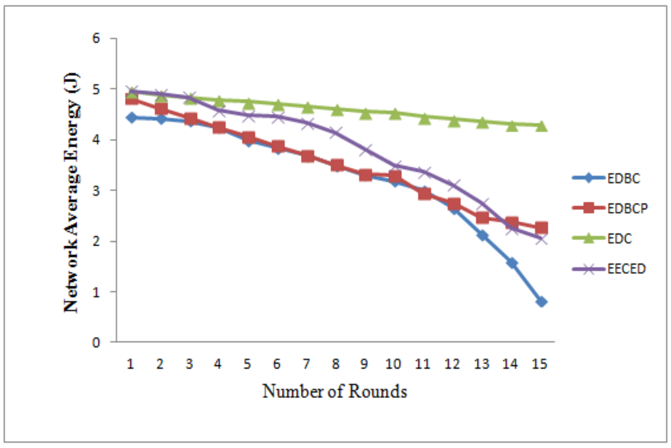
Sub-sections i) and ii) present the results of the effects of Rounds and Network size on the network average remaining energy respectively.

i) The Effect of Rounds on Network Average Remaining Energy

The purpose of this experiment was to determine the effect of leader selection rounds for both the energy based and the position based LSAs on the Network Average Energy. This metric is considered in order to find out which Leader Selection Algorithm consumes more energy, for the purpose of selecting a leader after every round. Figures 4.9 and 4.10 depict the results of the network average remaining energy for both the Position-based and the Energy-based when respectively subjected to the various leader selection rounds.

It can be observed from Figure 4.9 that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. The higher remaining energy value achieved by the event-based leader selection algorithms can be attributed to the leader selection process. In event-based leader selection processes, each round is triggered by the occurrence of an event that has been sensed, hence the time between leader selection rounds varies. The longer it takes for a process to select a new leader, the more it helps to save a considerable amount of energy in the network. However, the low performance of distance-based leader selection algorithms can be attributed to the great distance between the nodes and the base station

Nodes, and the base station which makes the nodes, consume more energy while sending the messages to the base station. Also, the cluster head consumes some energy while aggregating



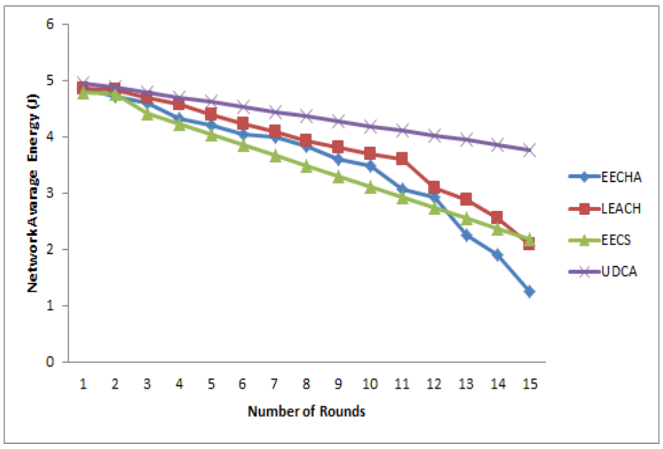


Figure 4.9 Effect of Rounds on Position-based Network Average Remaining Energy

Figure 4.10 Effect of Rounds on Energy-based Network Average Remaining Energy

information from different nodes in the network, all of which contributed to the low average remaining energy value experienced by the distance-based algorithms.

In Figure 4.10, it can be deduced that the UDCA of Heterogeneous-based leader selection algorithms outperform other considered Energy-based leader selection algorithms. Although, there is inconsistency in the performance of the remaining three algorithms (LEACH, EECS & EECHA) considered this behaviour can be attributed to the stochastic selection of leader by the homogeneous-based leader selection. Based on stochastic selection of leader by homogeneous-based leader selection algorithms, nodes with low remaining energy can be selected as leader. Hence, it dissipates the entire network energy faster, since a new leader has to be selected every time the current leader fails. Also, the cluster-heads consume extra energy while aggregating data and performing multi-hop data transmission from different nodes in the network to the base station.

ii) The Effect of Network size on Network Average Remaining Energy

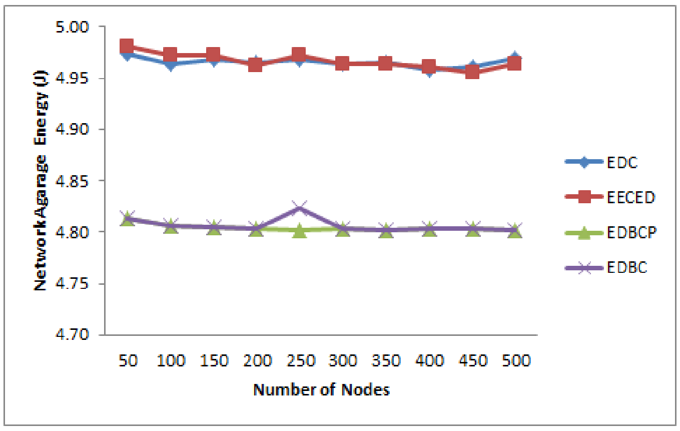
The purpose of this experiment was to determine the effect of different network sizes on the Network Average Remaining Energy for both the energy-based and the position-based LSAs.

This metric is considered in order to find out which Leader Selection Algorithm consumes more energy in the course of selecting a leader when subjected to various network sizes.

Figures 4.11 and 4.12 depict the results of the network average remaining energy for both the Position-based and the Energy-based ^ ^, when respectively subjected to various network sizes.

It can be observed from Figure 4.11 that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. The higher remaining energy value achieved by the event-based leader selection algorithms can be attributed to the event-based leader selection process. The Event-based leader selection process only selects a leader when an event occurs and it is only the nodes around that event that will participate in the process of selecting a leader at that particular given time, which prevents the network from consuming a lot of energy. However, in the distance-based leader selection process, cluster leaders are being selected continuously and all the nodes participate in the selection of cluster leader. This process consumes a lot of energy, since all nodes in the network will be active by sending and receiving messages during the process of leader selection.

In Figure 4.12 it can be observed that none of the four energy-based algorithms (EECS, UDCA, LEACH and EECHA) considered for both heterogeneous- and homogeneous-based leader selection performs better than the others. This behaviour can be attributed to the stochastic selection of leader by the homogeneous-based leader selection. Based on stochastic selection of leader by homogeneous-based leader selection algorithms, nodes with low remaining energy can be selected as a leader. Hence, it dissipates the entire network energy faster, since a new leader has to be selected every time the current leader fails. Also, the cluster-heads consumes extra energy while aggregating data and performing multi-hop data transmission from different nodes in the network to the base station.



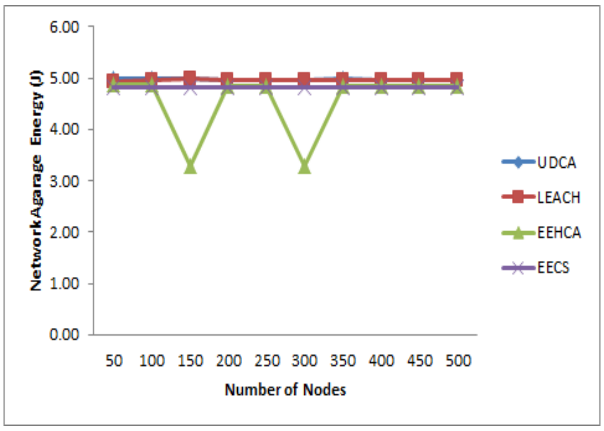


Figure 4.11 Effect of Network size on Position-based Network Average Remaining Energy

Figure 4.12 Effect of Network size on Energy-based Network Average Remaining Energy

4.4.4 Experiment IV: Network Energy Consumption Rate

This section presents the results of the experiments that were carried out to investigate the network energy consumption rate for the selected eight LSAs, when subjected to different leader selection Rounds and various Network sizes.

Sub-sections i) and ii) present the results of the effects of Rounds and Network sizes on the network energy consumption rate respectively.

i) The Effect of Rounds on Network Energy Consumption Rate

The purpose of this experiment was to determine the effect of leader selection rounds on the Network Energy consumption rate for both the energy based and the position based LSAs. This metric is considered in order to find out which Leader Selection Algorithm consumes less energy in selecting a network leader after every round.

Figures 4.13 and 4.14 depict the results of the network energy consumption rate for both the Position-based and the Energy-based LSAs when respectively subjected to the various leader selection rounds.

It can be observed from Figure 4.13 that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. The low energy consumption rate achieved by the event-based leader selection algorithms can be attributed to the network leader selection process, as explained earlier in section 4.5.3. Event-based leader selection processes only select a leader when an event has occurred and it is only the nodes around that event that will participate in the process of selecting a leader at that particular given time, which in turn reduces the network’s energy consumption.

The cluster leaders are being selected continuously and all the network nodes normally participate in the leader selection, which in turn leads to a high energy consumption rate for the network, since all the nodes will be active during the process of selecting a leader. Also, the cluster leaders consume more energy while aggregating information from different nodes in the network, all of which contribute to the poor energy consumption rate experienced by the distance-based algorithms.

In Figure 4.14 it can be seen that the UDCA of Heterogeneous-based leader selection algorithms outperform other Energy-based leader selection algorithms considered. Although there is inconsistency in the performance of the LEACH and EECHA, t EECS is the least performing algorithm among all the LSAs considered in terms of energy consumption rate. The high energy consumption rate of the EECS can be attributed to its poor clustering optimization. In the EECS leader selection process, it creates high number of clusters, which leads to a high number of cluster heads and this in turn reduces the energy efficiency of the network. The LEACH and EECHA (homogeneous-based leader selection) inconsistency behaviour can be attributed to their stochastic selection of leader. Based on stochastic selection of leader by homogeneous-based leader selection algorithm, nodes with low remaining energy can be

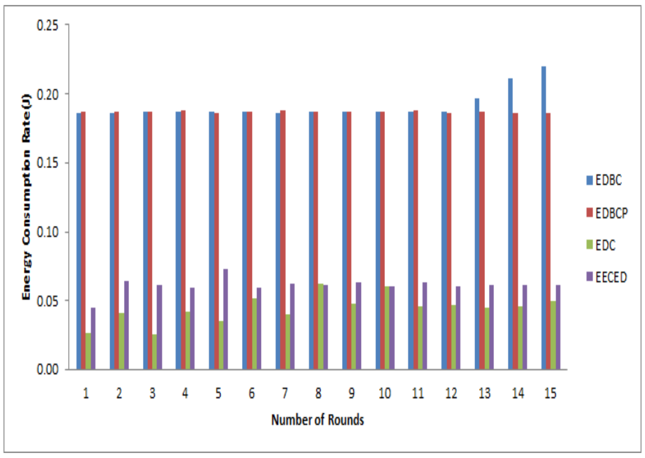


Figure 4.13 Effect of Rounds on Position-based Network Energy Consumption Rate

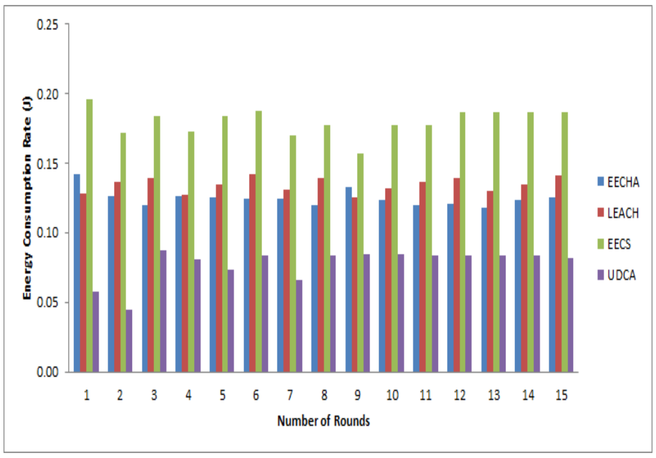


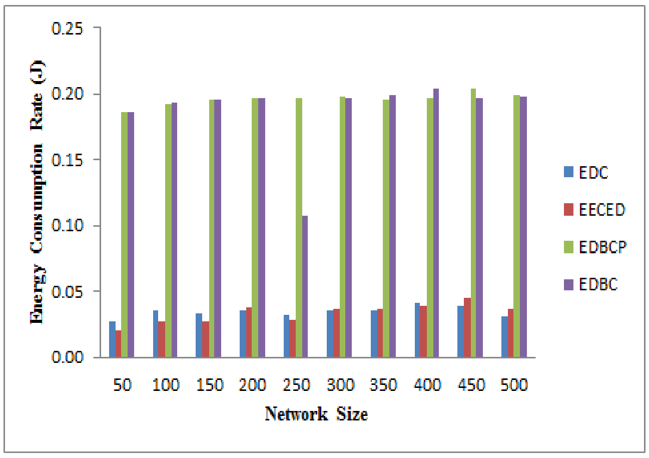
Figure 4.14 Effect of Rounds on Energy-based Network Energy Consumption Rate

selected as a leader. Hence, it dissipates the entire network energy faster, since a new leader has to be selected every time the current leader fails.

iii) The Effect of Network Sizes on Energy Consumption Rate

The purpose of this experiment was to determine the effect of different network sizes on the average Energy Consumption Rate for both the energy-based and the position-based LSAs. This metric is considered in order to find out which Leader Selection Algorithm consumes less energy in the course of selecting a leader when subjected to various network sizes.

Figures 4.15 and 4.16 depict the results of the network energy consumption rate for both the Position-based and the Energy-based, when respectively subjected to the various network sizes. It can be observed from Figure 4.15 that the Event-based (EDC and EECED) leader selection algorithms outperform the Distance-based (EDBCP and EDBC) leader selection algorithms. The lower energy consumption rate achieved by the event-based leader selection algorithms can be attributed to the event-based leader selection process. The Event-based leader selection process only selects a leader when an event occurs and it is only the nodes around that event



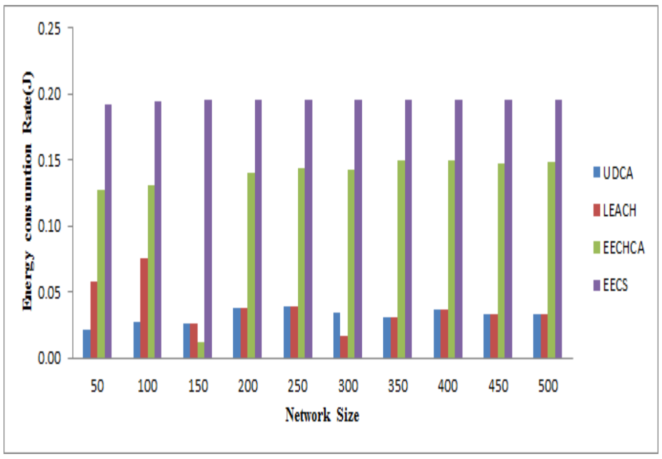


Figure 4.15 Effect of Network Size on Position-based Network Energy Consumption Rate

Figure 4.16 Effect of Network Size on Energy-based Network Energy Consumption Rate

that will participate in the process of selecting a leader at that particular given time. This prevents the network from consuming a lot of energy. However, in the distance-based leader selection process, cluster leader are being selected continuously and all the nodes participate in the selection of cluster leader. This process consumes a lot of energy, since all nodes in the network will be active by sending and receiving messages during the process of leader selection.

In Figure 4.16, it can be observed that both the UDCA and LEACH leader selection algorithms outperform other Energy-based leader selection algorithms considered. Also, it can be observed that they both have similar behaviour in most scenarios. EECS is the poorest performing algorithm among all the LSAs considered in terms of energy consumption rate. The poor performance of the EECS algorithm in terms of energy consumption rate can be attributed to its network clustering process. In the EECS leader selection process, the number of clusters normally increases as the network grows and each of these clusters has their cluster head. Due to the continuous increase in the number of cluster heads and the communication among those cluster heads and base station, more energy is being consumed, which in turn reduces the energy efficiency of the network.

The low energy consumption rate achieved by both the UDCA and LEACH algorithms can be attributed to their clustering optimization process. In this clustering optimization, few clusters are normally created with few cluster heads; the lower the number of clusters, the lower the cluster heads and the lower the cluster heads, the lower the energy consumption rate.

4.5 Summary

Table 4.2 summarizes the experimental results for both Position-based and Energy-based LSAs when respectively subjected to various rounds and network sizes using four metrics (Communication overhead, Leader selection Delay, Network average remaining energy and Energy consumption rate). The results obtained from the experiments show that for the Position-based ^ ^, which comprises both the event- and distance-based scenarios the event-based algorithms outperform the distance-based algorithms when respectively subjected to various rounds and different network sizes. While for the Energy-based, which comprise both the heterogeneous- and homogeneous-based scenarios, the heterogeneous-based algorithms outperform the homogeneous-based algorithms. Although there are some inconsistencies in the behaviour of both the heterogeneous- and homogeneous-based scenarios, homogeneous algorithms are not recommended, because of their stochastic selection of leader, which can lead to the compromising of network reliability.

Table 4.2. Summary of the Experimental Results

Metrics	Considered Scenarios Summary
Communication Overhead	Position-based scenarios: Event (EDC & EECED) outperform Distance (EDBC & EDBCP) for both Rounds and Network sizes.
	Energy-based scenarios: Heterogeneous (UDCA & EECS) outperform Homogeneous (LEACH & EECHA) for both Rounds and Network sizes.
Leader Selection Delay	Position-based scenarios: Event (EDC & EECED) outperform Distance (EDBC & EDBCP) for both Rounds and Network sizes.
	Energy-based scenarios: Homogeneous (LEACH & EECHA) outperform Heterogeneous (UDCA & EECS) for Rounds while EECS outperform others (LEACH, EECHA & UDCA) for Network sizes.
Network Average Remaining Energy	Position-based scenarios: Event (EDC & EECED) outperform Distance (EDBC & EDBCP) for both Rounds and Network sizes.
	Energy-based scenarios: Heterogeneous (UDCA) outperform Homogeneous (LEACH & EECHA) for Rounds while none of the algorithms outperform others for Network sizes.
Energy Consumption Rate	Position-based scenarios: Event (EDC & EECED) outperform Distance (EDBC & EDBCP) for both Rounds and Network sizes.
	Energy-based scenarios: Heterogeneous (UDCA) outperform Homogeneous (LEACH & EECHA) for Rounds while UDCA & LEACH algorithms outperform others for Network sizes.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This study is a successful attempt to explore leader selection algorithms to find out if there is any leader selection algorithms among existing leader selection algorithms (LSAs) that can work better in Wireless Mesh Networks (WMNs) for Mesh Key Distributor (MKD) selection, as all the existing leader selection algorithms were designed with wireless sensor network (for cluster leader selection in WSN) in mind. After comparing different LSAs, eight leader selection algorithms were selected and simulated in NS2 with WMNs simulation environment, and then their performance was evaluated. The main goal of this study was to evaluate the performance of LSAs in the context of MKD selection WMNs. It was important to first evaluate LSAs that already exist in other wireless networks since there are no LSAs in WMNs, so as to ascertain whether there is a LSA that works for WMNs, although they were designed for selecting cluster leaders in WSNs.

This study answered the following main research question: How is the process of evaluating existing leader selection algorithms (LSAs) in the context of MKD selection going to be conducted?

a. How can we create a classification framework for existing LSAs?
b. What are the selection algorithms that can be used for selecting the MKD?
c. What are the evaluation considerations for MKD selection algorithms?

The goal of this study was divided into three objectives that needed to be achieved in order to complete the study. Achieving the set objectives also provided answers to the research questions defined in Chapter One. This study had the following objectives: (1) to classify the existing leader selection algorithms (LSAs); (2) To select certain existing leader selection algorithms (LSAs) for evaluation; (3) to evaluate the selected leader selection algorithms.

The first objective was achieved by reviewing existing studies on the selection of a leader in the context of wireless sensor networks since there are no studies of leader selection in the context of wireless mesh network. This review of the literature answered the first research question. Leader selection algorithms were later organised into two different categories (Energy based leader selection and Position based leader selection) and also sub-divided into two groups per category. The leader selection algorithms in each group were then compared with each other in order to find two leader selection algorithms that were going to represent the group during simulation and evaluation. The second research objective was achieved through the comparison of leader selection algorithms and the subsequent selection of eight of them for evaluation. The third objective was achieved by the implementation and evaluation of the eight selected leader selection algorithms in Ns2.

In this study four performance metrics (Leader Selection Delay, Network Average Remaining Energy and Network Energy Consumption Rate) were used to evaluate each of the leader selection algorithms. Based on the evaluation of results this work drew the conclusion that, considering all performance metrics, Event based leader selection algorithms outperformed the distance based leader selection algorithms, considering both network size and number of leader selection rounds. The result shows that there is inconsistency in the performance of homogeneous energy based leader selection algorithms and heterogeneous energy based leader selection algorithms when taking into consideration both number of leader selection rounds and network size under network average energy and leader selection delay. However results also reveal that the Heterogeneous energy based leader selection algorithm outperforms the Homogeneous energy based leader selection algorithm in all other evaluating metrics. Therefore heterogeneous energy based leader selection is considered as the best leader selection algorithm in the energy based leader selection category. Based on the evaluation, it can be concluded that Event based leader selection algorithms and homogeneous energy based leader selection algorithms are the best leader selection algorithms to be used in the context of MKD selection in WMNs. By reaching this conclusion, we answered the last research question. The next section (5.2) presents the limitations of this study and indications for future work.

5.2 Limitation and Future Work

This section outlines the limitations and future work of this study. One of the limitations of this study is that simulation results may not mirror real world results, since they do not consider factors such as external interference. Doing the same experiments on a wireless test bed still needs to be considered in order to further validate the results obtained. This would not have been possible with the test bed that is running in the wireless mesh lab at the University of Zululand, because it contains only fourteen nodes. Further test bed implementation constraints were time and financial issues. With regard to future work, this study should consider using a test bed which will reflect real world results which should be compared with the simulation results described in this study. The only leader selection algorithms evaluated in this study were all from wireless sensor networks since there are no leader selection algorithms for Wireless Mesh Networks. New leader selection algorithms for WMNs need to be proposed and implemented for selecting an MKD.

5.3 Contribution to Knowledge

The aim of this study was to evaluate the performance of leader selection algorithms in the context of MKD selection in wireless mesh networks. To the best of my knowledge, the leader selection algorithms had not previously been evaluated in the context of MKD selection in WMNs. This study has catered to this evaluation. The evaluation has assisted this study to draw the conclusion that some leader selection algorithms (Event based leader selection and Heterogeneous energy based leader selection algorithms) from Wireless Sensor Networks can be implemented, enhanced and adopted for MKD selection in WMNs.

BIBLIOGRAPHY

Akyildiz, F. Wang, X and Wang W. “Wireless Mesh Networks” USA: John Wiley & Sons, Inc , 2009.

Akyldiz, F. Wang, X. “A Survey on Wireless Mesh Networks,” Computer Network, Vol 47, no 4, pg. 445-487 March 2005, September 2005.

Bai, X. Wu, X. ” simulation and visualization platform for fractionated spacecraft attitude control system”, International Conference on 7-10 August 2011.

Boregowda, S.B. Hemanth, K. Babu, N.V. Puttamadappa, C. and Mruthyunjaya, XZ., “UDCA: An Energy Efficient Clustering. Algorithm for Wireless Sensor Network.” World Academy of Science, Engineering and Technology conference No 46 January 2010.

Chan, H. and Perrig, A., “An Emergent Algorithm Highly Uniform Cluster Formation” , In proceedings of the first European Workshop on Sensor Network(EWSN) , 2004.

Chen, Y.P. Liestman, A.L. Liu, J.,“A hierarchical energy-efficient framework for data aggregation in wireless sensor networks” IEEE Transactions on Vehicular Technology, Vol 55, no. 3, pg. 789 - 796 June 2006.

Cetintemel, U. Keleher, PJ., ” Light-weight currency management mechanisms in Deno”, Proc. 10th IEEE Workshop on Research, 2000.

Cheng, W. Shi, H., “AEEC: An adaptive energy efficient clustering algorithm in sensor networks.”, 4th IEEE Conference. May 2009.

Di Francesco, M. Das, S.K. Anastasi, G., “Data Collection in Wireless Sensor Networks with Mobile Elements: A Survey.”, Journal ACM Transactions on Sensor Networks (TOSN), Volume 8 no 1, August 2011.

Gamwarige, S. Sankalpa, C. and Kulasekere, E., "An energy efficient distributed clustering algorithm for ad-hoc deployed wireless sensor networks in building monitoring applications." Electronic Journal of Structural Engineering (eJSE) Special Issue: Sensor Network on Building Monitoring: from Theory to Real Application pg.11-27, 2009.

Golam, M.D. and Hasnat, K.M. "Weighted election protocol for clustered heterogeneous wireless sensor networks." Journal of Mobile Communication Vol 4 no.2 : pg.38-42, 2010.

Handy, M.J. Haase, M. and Timmermann, D., "Low energy adaptive clustering hierarchy with deterministic cluster-head selection." Mobile and Wireless Communications Network, 2002. 4th International Workshop on. IEEE, 2002.

Han, Y. Park, S. Eom, J. and Chung, T., "Energy-efficient distance based clustering routing scheme for wireless sensor networks." In Computational Science and Its Applications–ICCSA 2007, Springer Berlin Heidelberg, 2007.

Hassanein, H. Takahara, G. Wang, Q. Xu K., "Relay node deployment strategies in heterogeneous wireless sensor networks." IEEE Transactions on Mobile Computing 2 2009.

Heinzelman, W.R. Chandrakssan, A.P. Balakrishan, H., “An application-specific protocol architecture for wireless microsensor networks” , IEEE Transactions on Wireless Communications, 2002.

Heinzelman, W.R. Chandrakasan, A.P. & Balakrishnan, H., “Energy-efficient communication protocol for wireless microsensor networks.“|, Proceedings of the 33rd annual Hawaii international conference on IEEE January 2000.

Katiyar, V. Chand, N. Soni, S. Int, J., “A Survey on Clustering algorithms forHeterogeneous Wireless Sensor Networks.”Advanced Networking and Applications Vol.02, no 04, pg. 745-754, 2011.

Lindsey, S. Raghavendra, C.S., “ PEGASIS: power efficient gathering in sensor information systems”, Proceedings of the IEEE Aerospace Conference, March 2002.

Liu, X., *“*A Survey on Clustering Routing Protocols in Wireless Sensor Networks.*”*

School of Electronic and Information Engineering, 9 August 2012

Liu, Y. Zhu, J.G.L. Zhang, Y., “CABCF: Cluster Algorithm Based on Communication Facility in WSN(CABCF).” ’09 WRI International Conference on Jan 2009.

Lukas, G. Fackroth, C., ”WMNSec – Security for Wireless Mesh Networks”, Distributed Systems of Magdeburg ,2009.

Matthee, K. W. Mweemba, G. Pais, A.V. Van Stam, G. Rijken, M., “Bringing Internet connectivity to rural Zambia using a collaborative approach”, Information and Communication Technologies and Development conference, 2007.

Mhatre, V. Rosenberg, C. , “Design guidelines for wireless sensor networks: communication, clustering and aggregation” Ad Hoc Networks, Vol 2, no 1, pg 45-63 January 2004.

Nurhayati, N. Lee, KO., “Clustering routing protocol based on location node in wireless sensor networks”, SEPADS'11 Proceedings of the 10th WSEAS international conference on Software engineering, parallel and distributed systems, 2011

Otgonchimeg B.,

Kwon, Y., “EECED: Energy Efficient Clustering Algorithm for Event-Driven Wireless Sensor Networks.” NCM '09. Fifth International Joint Conference, August 2009.

Pejovic, V. Belding, E. Marina, M., 2009 "An energy-flow model for self-powered routers and its application for energy-aware routing." NSDR’09 ,2009

Poor, R. Bowman, C. Auburn, C., May 2003, “self-healing networks”, ACM Queue, Vol 1, no 2 pg 52-59 May 2003.

Prabhu, B. Sophia, S. Maheswaran, S. Navaneethakrishnan M., “Real - World Applications of Distributed Clustering Mechanism in Dense Wireless Sensor Networks.”, International Journal of Computing, Communications and Networking, Volume 2, No.4, October-December 2013,

Singh, S. Singh, M. and Singh, D ., “Energy Efficient Homogeneous Clustering Algorithm for Wireless Sensor Networks.” International Journal of Wireless & Mobile Networks (IJWMN), Vol.2, No.3, August 2010.

Smaragdakis, G. Matta, I. Bestavros A., ” SEP: A stable election protocol for clustered heterogeneous wireless sensor networks” Second International Workshop on, 2004 .

Shirmohammadi, M.M. Islamic, AU. Hamedan, B. H. , Faez, K. and Chhardoli M.“LELE: Leader Election with Load Balancing Energy in Wireless Sensor Network”,WRI International Conference on Jan. 2009.

Udit, S. and Mitra P., ” Distributive Energy Efficient Adaptive Clustering Protocol for Wireless Sensor Networks,”Mobile Data Management, 2007 International Conference on ,2007.

Wang, Q. Hassanein, H. Takahara, G., "Stochastic Modelling of Distributed, Dynamic, Randomized Clustering Protocols for Wireless Sensor Networks", Proceedings of the 2004 International Conference on Parallel Processing Workshops, 2004.

Wei, Z.Z. Hui, W,Z. Zhong, L.H., "An Event-Driven Clustering Routing Algorithm for Wireless Sensor Networks", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. Sep - Oct. 2004.

Ye, M.   Li, C. Chen, G. Wu, J., "EECS: an energy efficient clustering scheme in wireless sensor networks", Performance, Computing, and Communications Conference, IPCCC 2005. 24th IEEE International, April 2005.

Yick, J. Mukherjee, B. Ghosal, D., "Wireless sensor network survey.", Department of Computer Science, University of California, Davis, CA 95616, April 2008.

Zheng, J. and Jamalipour A., "Wireless Sensor Networks: A NetworkingPerspective", USA: John Wiley & Sons, Inc, 2009.

APPENDIX A Source Code for EECS

The complete source code for the eight algorithms considered is stored in the attached CD

```
#ifndef _EECS_AGENT_H_
#define _EECS_AGENT_H_
#include <agent.h>
#include <mobilenode.h>
#include <packet.h>
#include <vector>
#include <map>
#include <timer-handler.h>
#include <config.h>
#define INITIAL_ENERGY 5.0
#define DESIRED_CLUSTERS 1
class EECS_Agent;
class EECSInfoBroadcastTimer : public TimerHandler
{
        public:

        EECSInfoBroadcastTimer(EECS_Agent* _agent)
: TimerHandler() {
                                        mAgent =
_agent;
                }
                virtual void expire(Event* evt);
        private:
                EECS_Agent* mAgent;
};
class remainingEnergyBroadcastTimer : public
TimerHandler
{
        public:

        remainingEnergyBroadcastTimer(EECS_Agent*
_agent) : TimerHandler() {
                                        mAgent =
_agent;
                }
                virtual void expire(Event* evt);
        private:
                EECS_Agent* mAgent;
};

class EECS_Agent: public Agent {

        friend class EECSInfoBroadcastTimer;
        friend class remainingEnergyBroadcastTimer;

        public:
                EECS_Agent();
                ~EECS_Agent();
                void recv(Packet*, Handler*);
                static std::vector<MobileNode*>
mNoneClusterHeads;
                double mLeaderElectedTimestamp;
                double mInfBroadcastTimestamp;

            protected:
    int command(int, const char* const*);
    void setUp();

    //Packet sending functions
    void sendInfMsg();
    void sendAdvMsg();
    void sendremainingEnergyMsg();
     //Packet receiving functions
    void recvEECSMsg(Packet*);
    void recvHelloMsg(Packet*);
    void recvInfMsg(Packet*);
    void recvAdvMsg(Packet*);
    void recvCHMsg(Packet*);
    //Utility functions
    double getThreshold();
    double getRemainingEnergy();
    double getTotalResidualEnergy();
    double getTotalDistance();

        protected:

                enum NodeTypes
{NORMAL,CANDIDATE,CLUSTER_HEAD};

                int mNodeType; // Node type

                MobileNode *mCurrentNode; // A
pointer to the current node

                MobileNode *mBSNode;   // A
pointer to the base station node

                int mNodeId; // Current node id

                double mResidualEnergy; // Current
node residual energy

                double mDistance; // Distance from
current node the base station

                NsObject *mLL; // a link layer target

                std::vector<Packet*> mInfoCache; //
Local cache for info packets

                int max_rounds; // Maximum rounds

            int mCurrentRound; // Current round
number

                EECSInfoBroadcastTimer mIBTimer;
                remainingEnergyBroadcastTimer
mABTimer;

                };

#endif
```

```
#include <bs-edbc/edbc_packet.h>
#include <bs-edbc/edbc_agent.h>
#include <bs-edbc/edbc_bsagent.h>
#include <random.h>
#include <cmu-trace.h>
#include <god.h>

//Static variables
//int EDBC_Agent::mCurrentRound = 0;
std::vector<MobileNode*>
EDBC_Agent::mNoneClusterHeads;

static class EDBC_AgentClass : public TclClass
{
        public:
                EDBC_AgentClass()                :
TclClass("Agent/EDBC_Agent") { }
                TclObject*  create(int  argc,  const
char* const* argv)
                {
                        return              (new
EDBC_Agent());
                }
} class_EDBC_Agent;

EDBC_Agent::EDBC_Agent(): Agent(PT_EDBC),

                mCurrentRound(0),

                mNodeType(NORMAL),

                mIBTimer(this),

                mABTimer(this),

                mLeaderElectedTimestamp(0.0),

                mInfBroadcastTimestamp(0.0)
{
        bind("max_rounds" , &max_rounds);
}

EDBC_Agent::~EDBC_Agent()
{
}

int EDBC_Agent::command(int argc, const char* const*
argv)
{
        if(argc == 2)
        {

          if (strcmp(argv[1], "start") == 0)
                {
                        mIBTimer.sched(0.3);
                        //printf("CAN        Start
clock%.6f\n", Scheduler::instance().clock());
                        return (TCL_OK);
                }       }
        else if(argc == 3)
        {
                if (strcmp(argv[1], "index") == 0)
                {
                                        //Current   node
id
                                mNodeId = atoi(argv[2]);

                                mBSNode               =
(MobileNode*)(Node::get_node_by_address(0));

                                //Current node
                                mCurrentNode          =
(MobileNode*)(Node::get_node_by_address(mNodeId))
;



                                        mDistance=
mCurrentNode->distance(mBSNode);

                                return (TCL_OK);
                }
                else if (strcmp(argv[1], "set-ll") == 0)
                {
                                        //Initialise    the
link layer target
                                        mLL=
(NsObject*)TclObject::lookup(argv[2]);

                                        //Checks
whether the target is not null
                                        if (mLL == 0)
                                        {
                                          printf("no such
object %s", argv[2]);
                                                return
(TCL_ERROR);
                                        }
                                        return (TCL_OK);
                }
        }
        return Agent::command(argc, argv);
}

//==========================================
========================
// Timers
//==========================================
========================
void EDBCInfoBroadcastTimer::expire(Event*){

   if(mAgent->mNodeType != mAgent->CLUSTER_HEAD)
   {
    //printf("CAN           Inf            clock%.6f\n",
Scheduler::instance().clock());
    //Get the random probability value

    mAgent->mInfBroadcastTimestamp              =
Scheduler::instance().clock();

    double probs =  Random::uniform(0.0, 1.0);

    mAgent->mCurrentRound++;
        printf("---------------------------------------------------
---------\n");
```

```
                    printf("Current            Round%d\n",
mAgent->mCurrentRound);
                    printf("--------------------------------------
--------------------------\n");

   printf("Node Id->%d\n", mAgent->mNodeId );

   //Check the probability against the threshold
   if(probs < mAgent->getThreshold()){

      mAgent->mNodeType = mAgent->CANDIDATE;

      if(mAgent->mNodeType ==  mAgent->CANDIDATE)
      {
        mAgent->sendInfMsg();
        mAgent->mABTimer.resched(0.3);

      }
   }  }
}
void EDBCAbilityBroadcastTimer::expire(Event*){
  //printf("CAN            Ability            clock%.6f\n",
Scheduler::instance().clock());
         mAgent->sendAbilityMsg();
         if(mAgent->mCurrentRound     ==     mAgent-
>max_rounds)
         {
                  EDBC_BSAgent::killBSTimers();
         }
         else
         {
                  mAgent->mIBTimer.resched(0.6);
         }
}

void EDBC_Agent::setUp(){
}

//===========================================
================
// Packet sending funtions
//===========================================
=================
void EDBC_Agent::sendInfMsg(){


         Packet* p = Packet::alloc();
         struct hdr_cmn* ch = HDR_CMN(p);
         struct hdr_ip* ih = HDR_IP(p);
         struct hdr_edbc* ah = HDR_EDBC(p);


         ch->ptype() = PT_EDBC;
         ch->size() = IP_HDR_LEN + ah->size();
         ch->addr_type() = NS_AF_NONE;
         ch->direction() = hdr_cmn::DOWN;
         ch->prev_hop_ = mNodeId;
         ch->next_hop() = IP_BROADCAST;

         ih->saddr() = mNodeId;
         ih->daddr() = IP_BROADCAST;
         ih->ttl() = 1;

         ah->pkt_type() = EDBCTYPE_INF;
         ah->pkt_src() = mNodeId;
         ah->node_energy() = mResidualEnergy;
         ah->node_distance() = mDistance;

         //Add this packet as initial entry in the inf
cache
         //mInfoCache.push_back(p);

         Scheduler::instance().schedule(mLL, p, 0.0);

         printf("all nodes sends ID,Remaining Energy
and Distance to the BS...\n", mNodeId);
         }

void EDBC_Agent::sendAbilityMsg(){

         Packet* p = Packet::alloc();
         struct hdr_cmn* ch = HDR_CMN(p);
         struct hdr_ip* ih = HDR_IP(p);
         struct hdr_edbc* ah = HDR_EDBC(p);
         ch->ptype() = PT_EDBC;
         ch->size() = IP_HDR_LEN + ah->size();
         ch->addr_type() = NS_AF_NONE;
         ch->direction() = hdr_cmn::DOWN;
         ch->prev_hop_ = mNodeId;
         ch->next_hop() = IP_BROADCAST;

         ih->saddr() = mNodeId;
         ih->daddr() = IP_BROADCAST;
         ih->ttl() = 1;

         ah->pkt_type() = EDBCTYPE_AB;
         ah->pkt_src() = mNodeId;
         ah->pkt_dst() = 0; // Goes to the base station
         ah->node_ability() = getAbility();

         Scheduler::instance().schedule(mLL, p, 0.0);

         //printf("Candidate node %d is sending its
ability value to base station...\n", mNodeId);

}
//===========================================
===================
// Packet receiving functions
//===========================================
===================
void EDBC_Agent::recv(Packet* p, Handler*)
{

         struct hdr_cmn* ch = HDR_CMN(p);
         struct hdr_ip* ih = HDR_IP(p);

         //UDCA Packets
         if(ch->ptype() == PT_EDBC)
         {
                  //Drop if time to live has expired
                  if(ih->ttl() == 0)
                  {
                           drop(p, DROP_RTR_TTL);
                           return;
                  }
```

```
                        else
                        {
                                    //Decrement the ttl and
send to the appropriate receive method
                                    ih->ttl()--;
                                    recvEDBCMsg(p);
                                    return;
                        }
            }
            else
            {
                        //Any Packet type
                        // Must be a packet I'm originating
                        if((ih->saddr() == mNodeId) && (ch-
>num_forwards() == 0))
                        {

                                    // Add the IP Header. TCP
adds the IP header too, so to avoid setting it twice,
                                    // we check if this packet
is not a TCP or ACK segment.

                                    if (ch->ptype() != PT_TCP
&& ch->ptype() != PT_ACK)
                                    {
                                                ch->size()      +=
IP_HDR_LEN;
                                    }

                        }

                        // I received a packet that I sent.
Probably routing loop.
                        else if(ih->saddr() == mNodeId)
                        {
                                    drop(p,
DROP_RTR_ROUTE_LOOP);
                                    return;
                        }

                        //Packet I'm forwarding...
                        else
                        {
                                    if(--ih->ttl_ == 0)
                                    {
                                                drop(p,
DROP_RTR_TTL);
                                                return;
                                    }
                        }
            }
}

void EDBC_Agent::recvEDBCMsg(Packet* p)
{
            struct hdr_edbc* ah = HDR_EDBC(p);

            switch(ah->pkt_type()){

              case EDBCTYPE_HELLO:
                 recvHelloMsg(p);
                        break;
                        case EDBCTYPE_INF:
                 recvInfMsg(p);
                        break;
                        case EDBCTYPE_ADV:
                 recvAdvMsg(p);
                        break;
                        case EDBCTYPE_CHEL:
                 recvCHMsg(p);
                        break;
               default:
                        //Do nothing
                break;

            }

}
void EDBC_Agent::recvHelloMsg(Packet* p)
{
  //Current node residual energy
  mResidualEnergy = mCurrentNode->energy_model()-
>energy();

  //printf("Received a broadcast\n");
  Packet::free(p);
}
void EDBC_Agent::recvInfMsg(Packet* p)
{
  //Adds received  packet to the cache
  if(mNodeType == CANDIDATE){
     mInfoCache.push_back(p);
     //printf("Node id:%d Info size%d\n",mNodeId,
mInfoCache.size());
  }
}
void EDBC_Agent::recvAdvMsg(Packet* p)
{

}
void EDBC_Agent::recvCHMsg(Packet* p)
{


            if(mNodeType == CANDIDATE){
             //printf("Node %d is recv CH message\n",
mNodeId);

              hdr_edbc *ah = HDR_EDBC(p);

              nsaddr_t ch_index = ah->pkt_dst();

              if(ch_index == mNodeId){

                 mNodeType = CLUSTER_HEAD;

                 if(mNodeType == CLUSTER_HEAD){
                   mLeaderElectedTimestamp              =
Scheduler::instance().clock();
                   int nn =  God::instance()->nodes();
                   printf("Node %d is selected as cluster
head\n", mNodeId);
                   printf("Leader Election Delay ::%.6f\n",
mLeaderElectedTimestamp                              -
mInfBroadcastTimestamp+0.01230);
```

```
    }         }              }            }

//==========================================
==========================
// Utility functions
//==========================================
==========================
//Calculate the threshold for node i
double EDBC_Agent::getThreshold()
{

   //Threshold value
   double threshold = 0.0;

   std::vector<MobileNode*>::iterator it;

   for(it = mNoneClusterHeads.begin(); it <
mNoneClusterHeads.end();it++){

       MobileNode *tmpNode  = *it;

       //Checks whether node i has been selected before
as a leader
       if(mCurrentNode->nodeid() == tmpNode-
>nodeid()){

         threshold = (DESIRED_CLUSTERS / (1 -
DESIRED_CLUSTERS *(mCurrentRound * (1 %
DESIRED_CLUSTERS) )))*



            (mResidualEnergy/INITIAL_ENERGY);
         break;
       }     }
   return threshold; }

double EDBC_Agent::getAbility()
{

  double weight_factor = 0.7;

  double ability = (double)(weight_factor *
(mResidualEnergy/getTotalResidualEnergy()) +

                                                 (1 +
weight_factor)/(mDistance/getTotalDistance()));
  mInfoCache.clear();
  return ability;
}

double EDBC_Agent::getTotalResidualEnergy()
{
                    double totalResidualEnergy =  0.0;
                    std::vector<Packet*>::iterator it;

                    for(it = mInfoCache.begin(); it <
mInfoCache.end();it++)
                    {
                       Packet *pkt = *it;
                       hdr_edbc *ah = HDR_EDBC(pkt);
                       double residualEnergy = ah-
>node_energy();
                              totalResidualEnergy =
totalResidualEnergy + residualEnergy;

                    }
                    return totalResidualEnergy;
}

double EDBC_Agent::getTotalDistance()
{
                    double totalDistance =  0.0;
                    std::vector<Packet*>::iterator it;
                    for(it = mInfoCache.begin(); it <
mInfoCache.end();it++)
                    {
                       Packet *pkt = *it;
                       hdr_edbc *ah =
HDR_EDBC(pkt);
                       double distance = ah-
>node_distance();
                              totalDistance =
totalDistance + distance;

                    }
                    return totalDistance;
}

#include <eecs/eecs_packet.h>
#include <cmu-trace.h>
int hdr_eecs::offset_;
static class EECSHeaderClass : public
PacketHeaderClass
{
public:              EECSHeaderClass() :
PacketHeaderClass("PacketHeader/EECS",
sizeof(hdr_eecs))
        {
           bind_offset(&hdr_eecs::offset_);
        }
} class_EECShdr;
```

APPENDIX B: EECED Source Code

```
#ifndef _eeced_h_
#define _eeced_h_
#include <cmu-trace.h>
#include <priqueue.h>
#include <classifier/classifier-port.h>
#include <stats.h>
#include <mobilenode.h> // Included for requesting
node energy

//Agent constants
#define NETWORK_DIAMETER 64
#define DEFAULT_EREQ_INTERVAL 10 //seconds
#define DEFAULT_LDELECT_INTERVAL 15 //seconds
#define                    DEFAULT_ROUTE_EXPIRE
2*(DEFAULT_EREQ_INTERVAL                      +
DEFAULT_LDELECT_INTERVAL) //seconds
#define ROUTE_PURGE_FREQUENCY 2 // seconds
#define ENERGY_THRESHOLD 0.8
class EECED;
//==========================================
==============
// Timers : Energy Request Timer, Leader Election Timer,
Route Cache Timer
//==========================================
==============
class EnergyRequestTimer : public TimerHandler {
        public:
                EnergyRequestTimer(EECED*
_agent) : TimerHandler() {
                                mAgent       =
_agent;
                }       virtual void expire(Event*
evt);
        private:
                EECED* mAgent;
};
class LeaderElectionTimer : public TimerHandler
{
        public:
                LeaderElectionTimer(EECED*
_agent) : TimerHandler() {
                                mAgent       =
_agent;
                }
                virtual void expire(Event* evt);
        private:
                EECED* mAgent;
};
class EECEDRouteCacheTimer : public TimerHandler
{
        public:
                EECEDRouteCacheTimer(EECED*
_agent) :TimerHandler() {
                                mAgent       =
_agent;
                } virtual void expire(Event* evt);
        private:
                EECED* mAgent;
};
//==========================================
====================
//Route Cache Table
//==========================================
====================
class RouteEntry
{       friend class EECED;
        public:
                RouteEntry(int _bid, nsaddr_t _bsrc)
                {       rt_seqno = _bid;
                        rt_dst = _bsrc;
                }
        protected:
                int rt_seqno;  //  route  sequence
number
                nsaddr_t rt_dst; // route destination
                nsaddr_t  rt_nexthop;  //  Next  hop
towards the destination
                double rt_energy;  // energy level of
the destination
                int rt_hopcount; // Number of hops
towards the destination
};
#include <eeced/eeced_packet.h>
#include <eeced/eeced.h>
#include <cmu-trace.h>
#include <mobilenode.h>
#include <random.h>
#include <god.h>
using namespace std;
int hdr_eeced::offset_;
//double EECED::election_receive_timestamp;
//Packet Header implementation
static     class     EECEDHeaderClass     :     public
PacketHeaderClass {
        public:
                EECEDHeaderClass()                 :
PacketHeaderClass("PacketHeader/EECED",
sizeof(hdr_eeced_all))   {
                bind_offset(&hdr_eeced::offset_);
      } } class_EECEDhdr;
// Agent class implementation
static class EECEDClass : public TclClass {
        public:
                EECEDClass()                       :
TclClass("Agent/EECED") { }
                TclObject*  create(int  argc,  const
char* const* argv)
                {
                        return (new EECED());
                }
} class_EECED;
// Command line Tcl interface to the protocol
int EECED::command(int argc, const char* const* argv)
{       Tcl& tcl = Tcl::instance();
                if(argc == 2)
        {       if(strncasecmp(argv[1], "id", 2) == 0)
          {
                //Displays the node address
                tcl.resultf("%d", mNodeIndex);
                        return TCL_OK;
```

```
			} else if (strcmp(argv[1], "start") ==
0){
					printf("Node %d Broadcast
energy request message \n", mNodeIndex);
					if(mNodeRole		==
ELECTOR){
					  // add_neighbor_list();
					  startSim();
						return (TCL_OK);
					}	}	}
		else if(argc == 3){
			if (strcmp(argv[1], "index") == 0){
						//Initialise	the
node address/id
					mNodeIndex		=
atoi(argv[2]);
						//Initialise	a
pointer to the current node
						mCurrentNode =
(MobileNode*)(Node::get_node_by_address(mNodeInd
ex));
						//Initialise	a
pointer to the energy model

		mMobileNodeEnergyModel = mCurrentNode-
>energy_model();
					//Update node energy
					mNodeEnergy		=
mMobileNodeEnergyModel->energy();

			return (TCL_OK);
			} else if (strcmp(argv[1], "set-ll") ==
0){
						//Initialise	the
link layer target
						mLLTarget	=
(NsObject*)TclObject::lookup(argv[2]);
						//Checks
whether the target is not null
						if (mLLTarget ==
0){

		tcl.resultf("no such object %s", argv[2]);
							return
(TCL_ERROR);
						}return
(TCL_OK);
			} }
		return Agent::command(argc, argv);
}
//==========================================
==============================
// Default Constructor
//==========================================
==============================
EECED::EECED(): Agent(PT_EECED),

		mEnergyRequestTimer(this),

		mLeaderElectionTimer(this),

		mRouteCacheTimer(this),

		mEnergyRequestSeqno(1),

		mEnergyReplySeqno(1),

		mLeaderElectionSeqno(1),

		mClusterAdvertisementSeqno(1),

		mNodeRole(NORMAL){
		//Bind node role to the tcl interface
		bind("node_role", &mNodeRole);
		bind("max_rounds", &mMaxRounds);
}
//Reclaiming memory
EECED::~EECED() {
		delete mCurrentNode;
		delete mElectorNode;
		delete mMobileNodeEnergyModel;
		delete mLLTarget;
		}



// Adding neighbor: did not work
//==========================================
==============================
//		EECED TIMERS
//==========================================
==============================
void EECED::log_energy_clock(){
    mEREQTime = Scheduler::instance().clock();
    printf("Energy Request Clock:%0.6f\n", mEREQTime);
   }
void EECED::log_selection_clock(){
   mCHBRTime = Scheduler::instance().clock();
		printf("Leader	Selection	Clock:%0.6f\n",
mCHBRTime);
}
void EnergyRequestTimer::expire(Event* evt) {
		mAgent->mEnergyRepliesCache.clear();

   mAgent->log_energy_clock();
		mAgent->send_energy_request();

		double delay = 0.14;
		mAgent-
>mLeaderElectionTimer.resched(delay);
}
void EECED::startSim() {
		mEnergyRequestTimer.resched(0.0);
		//mLeaderElectionTimer.sched(2.0);
		}
void EECED::reset_ereq_timer()
{
}
void EECED::printRounds(){
     printf("Current Rounds%d\n", mCurrentRounds);
}
//Timer to elect a leader
void LeaderElectionTimer::expire(Event* evt)
{
   mAgent->log_selection_clock();
```

```
            mAgent->broadcast_leader_election();

            mAgent->mReplyTimestamps    =    mAgent-
>mStats.getReplyTimestamps(mAgent-
>mEnergyRepliesCache);
            //terminator
            mAgent->mCurrentRounds++;
            mAgent->printRounds();
            if(mAgent->mEnergyRepliesCache.size() > 0)
            {
            //Calculates leader election delay
            mAgent-
>mStats.getClusterHeadElectionDelay(mAgent-
>mEnergyRequestTimestamp,
                        mAgent->mReplyTimestamps,

                                            mAgent-
>mElectionSendTimestamp,

             mAgent->mElectionReceiveTimestamp);
                    //Calculates average network energy
                    mAgent-
>mStats.getAverageNetworkEnergy();
                    //Calculates the energy consumption
rate
                    mAgent-
>mStats.getEnergyConsumptionRate(mAgent-
>mCurrentRounds);
            }
            if(mAgent->mCurrentRounds    ==    mAgent-
>mMaxRounds){
                    return;
            }
            double delay = 0.15;
            mAgent-
>mEnergyRequestTimer.resched(delay);
}
//Route purge timer(every 2 seconds)
void EECEDRouteCacheTimer::expire(Event* evt){
                    //mAgent->rt_purge();
}
//===========================================
===============================
//  Packet forwarding routines
//===========================================
===============================
// Forward the energy request packet
void EECED::forward_request( int ereq_pkt_bid,

nsaddr_t ereq_pkt_src,

nsaddr_t erep_pkt_nexthop,
                                            int
ereq_pkt_hopcount,
                                            int
ereq_node_role,

double ereq_pkt_timestamp) {
            Packet* p = Packet::alloc();
            struct hdr_cmn* ch = HDR_CMN(p);
            struct hdr_ip* ih = HDR_IP(p);
            struct      hdr_eeced_ereq*      ah      =
HDR_EECED_EREQ(p);
            ch->ptype() = PT_EECED;
            ch->size() = IP_HDR_LEN + ah->size();
            //ch->error() = 0;
            ch->prev_hop_ =  mNodeIndex;
            ch->direction() = hdr_cmn::DOWN;
            ih->saddr() =  mNodeIndex;
            ih->ttl() = 1;
            // Forwards to a particular address
  if(erep_pkt_nexthop != (nsaddr_t) IP_BROADCAST) {
                    ch->next_hop() = erep_pkt_nexthop;
                    ch->addr_type() = NS_AF_INET;
                    //Dest address is a specific address
                    ih->daddr() = erep_pkt_nexthop;
            }
  else    {
     ch->next_hop() = IP_BROADCAST;
     ch->addr_type() =  NS_AF_NONE;
                    //Destination address is a broadcast
                    ih->daddr() = IP_BROADCAST;
            }
            //Copying from previous packet
            ah->ereq_pkt_type() = EECEDTYPE_EREQ;
            ah->ereq_pkt_bid() = ereq_pkt_bid;
            ah->ereq_pkt_src() = ereq_pkt_src;
            ah->ereq_pkt_hopcount()                    =
ereq_pkt_hopcount;
            ah->ereq_pkt_timestamp()                   =
ereq_pkt_timestamp;
            ah->ereq_node_role() = ereq_node_role;
            //double delay =  0.1 + Random::uniform();
            //Schedules immediately
            Scheduler::instance().schedule(mLLTarget,    p,
0.0);
            printf("Node %d is forwarding the energy
request to %d\n", mNodeIndex, erep_pkt_nexthop);
}
//Forwards the reply packet
void EECED::forward_reply( int erep_pkt_bid,
                                    int
            erep_pkt_rep_id,
                                    nsaddr_t
erep_pkt_src,
                                    nsaddr_t
erep_pkt_dst,

nsaddr_t erep_pkt_nexthop,
                                            int
erep_pkt_hopcount,
                                            double
erep_pkt_timestamp,
                                            int
erep_node_role,
                                            double
erep_node_energy)
{

            Packet* p = Packet::alloc();
            struct hdr_cmn* ch = HDR_CMN(p);
            struct hdr_ip* ih = HDR_IP(p);
            struct      hdr_eeced_erep*      ah      =
HDR_EECED_EREP(p);
            ch->ptype() = PT_EECED;
            ch->size() = IP_HDR_LEN + ah->size();
            //ch->error() = 0;
            ch->prev_hop_ =  mNodeIndex;
            ch->direction() = hdr_cmn::DOWN;
            ih->saddr() =  mNodeIndex;
            ih->ttl() = 1;
            // Forwards to a particular address
   if(erep_pkt_nexthop != (nsaddr_t) IP_BROADCAST)
   {
                        ch->next_hop() = erep_pkt_nexthop;
                        ch->addr_type() = NS_AF_INET;
                        //Dest address is a specific address
                        ih->daddr() = erep_pkt_nexthop;
            }
   else  {
      ch->next_hop() = IP_BROADCAST;
      ch->addr_type() =  NS_AF_NONE;
                        //Destination address is a broadcast
                        ih->daddr() = IP_BROADCAST;
            }
            //Copying from previous packet
            ah->erep_pkt_type() = EECEDTYPE_EREP;
            ah->erep_pkt_bid() = erep_pkt_bid;
            ah->erep_pkt_rep_id() = erep_pkt_rep_id;
            ah->erep_pkt_src() = erep_pkt_src;
            ah->erep_pkt_dst() = erep_pkt_dst;
            ah->erep_pkt_hopcount()                          =
erep_pkt_hopcount;
            ah->erep_pkt_timestamp()                         =
erep_pkt_timestamp;
            ah->erep_node_role() = erep_node_role;
            ah->erep_node_energy() = erep_node_energy;
            //Schedules immediately
            Scheduler::instance().schedule(mLLTarget,     p,
0.0);
            printf("Node %d is forwarding the energy reply
to %d\n", mNodeIndex, erep_pkt_nexthop);
}
//===========================================
==============================
//
            ENERGY REQUEST AND REPLY FUNTIONS
//===========================================
==============================
void EECED::send_energy_request() {
            Packet* p = Packet::alloc();
            struct hdr_cmn* ch = HDR_CMN(p);
            struct hdr_ip* ih = HDR_IP(p);
            struct        hdr_eeced_ereq*        ah         =
HDR_EECED_EREQ(p);
            ch->ptype() = PT_EECED;
            ch->size() = IP_HDR_LEN + ah->size(); //   We
are going to put a number: 7 bytes
            ch->addr_type() = NS_AF_NONE;
            ch->direction() = hdr_cmn::DOWN;
            ch->prev_hop_ = mNodeIndex;
            ch->next_hop() = IP_BROADCAST;
            ih->saddr() = mNodeIndex;
            ih->daddr() = IP_BROADCAST;
            ih->ttl() = 1;
            ah->ereq_pkt_type() = EECEDTYPE_EREQ;
            ah->ereq_pkt_bid() = mEnergyRequestSeqno;
            ah->ereq_pkt_src() = mNodeIndex;
            ah->ereq_pkt_hopcount() = 1;
   //Record timestamp for a broadcast
            mEnergyRequestTimestamp                          =
Scheduler::instance().clock();
            ah->ereq_pkt_timestamp()                         =
mEnergyRequestTimestamp;
            ah->ereq_node_role() = mNodeRole;
            Scheduler::instance().schedule(mLLTarget,     p,
0.0);
            mEnergyRequestSeqno++;
            printf("Elector node %d is broadcasting an
energy request...\n", mNodeIndex);


}
void EECED::send_energy_reply(int bid,

                              nsaddr_t nexthop,

                              nsaddr_t dst)
{
            Packet* p = Packet::alloc();
            struct hdr_cmn* ch = HDR_CMN(p);
            struct hdr_ip* ih = HDR_IP(p);
            struct        hdr_eeced_erep*        ah         =
HDR_EECED_EREP(p);
            ch->ptype() = PT_EECED;
            ch->size() = IP_HDR_LEN + ah->size();
            ch->addr_type() = NS_AF_INET;
            ch->direction() = hdr_cmn::DOWN;
            ch->prev_hop_ = mNodeIndex;
            ch->next_hop() = nexthop;
            ih->saddr() = mNodeIndex;
            ih->daddr() = nexthop;
            ih->ttl() = 1;
            ah->erep_pkt_type() = EECEDTYPE_EREP;
            ah->erep_pkt_bid() = bid;
            ah->erep_pkt_rep_id() = mEnergyReplySeqno;
            ah->erep_pkt_src() = mNodeIndex;
            ah->erep_pkt_dst() = dst;
            ah->erep_pkt_hopcount() = 1;
            ah->erep_pkt_timestamp()                         =
Scheduler::instance().clock();
            ah->erep_node_role() = mNodeRole;
            ah->erep_node_energy() = mNodeEnergy;
            //double            reply_time                   =
Scheduler::instance().clock() + Random::uniform();
            printf("Normal node %d is sending back
energy reply to %d which goes to %d...\n", mNodeIndex,
nexthop, dst);
            Scheduler::instance().schedule(mLLTarget,     p,
0.0);
            mEnergyReplySeqno++;
}
//===========================================
================================
//                       LEADER            ELECTION
FUNCTIONS
//===========================================
================================

void EECED::broadcast_leader_election()
{
        /* if replies cache is not empty, get the address of
         * packet with maximum energy and elect the node that sent
         * that packet to be the leader
         */
        mElectionSendTimestamp = Scheduler::instance().clock();
        if(mEnergyRepliesCache.size() > 0){
                //Gets the packet with maximum energy
                nsaddr_t dst = mStats.getPacketAddressWithMaxEnergy(mEnergyRepliesCache);
                //leader_addr = dst;
                //Sends the election broadcast

                send_leader_election(dst);
        }
        /* Otherwise elect yourself as a leader and acknowledge all nodes in your network
         * that you are elected as a leader
         */
        else if(mEnergyRepliesCache.size() == 0)
        {
                mNodeRole = LEADER;
                printf("The elector node %d is chosen as a leader\n", mNodeIndex);
                //send_cluster_advertisement();
                //Packet::free(p);
        }   }
void EECED::send_leader_election(nsaddr_t leader_addr) {
                Packet* p = Packet::alloc();
                struct hdr_cmn* ch = HDR_CMN(p);
                struct hdr_ip* ih = HDR_IP(p);
                struct hdr_eeced_cinf* ah = HDR_EECED_CINF(p);
                ch->ptype() = PT_EECED;
                ch->size() = IP_HDR_LEN + ah->size();
                ch->addr_type() = NS_AF_NONE;
                ch->error() = 0;
                ch->direction() = hdr_cmn::DOWN;
                ch->prev_hop_ = mNodeIndex;
                ch->next_hop() = IP_BROADCAST;

                ih->saddr() = mNodeIndex;
                ih->daddr() = IP_BROADCAST;
                ih->ttl() = 1;
                ah->cinf_pkt_type() = EECEDTYPE_CINF;
                ah->cinf_pkt_bid() = mLeaderElectionSeqno;
                ah->cinf_pkt_hopcount() = 1;
                ah->cinf_pkt_dest() = leader_addr;
                ah->cinf_pkt_timestamp() = Scheduler::instance().clock();
                ah->cinf_node_role() = mNodeRole;

        Scheduler::instance().schedule(mLLTarget, p, 0.0);
                mLeaderElectionSeqno++;
                printf("Elector node %d is broadcasting leader election....\n", mNodeIndex);
                }
//==========================================================================================
//                                        CLUSTER HEAD ADVERTISEMENT AND JOIN REQUEST FUNCTIONS
//==========================================================================================
void EECED::send_cluster_advertisement()
{
        Packet* p = Packet::alloc();
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_eeced_cadv* ah = HDR_EECED_CADV(p);
        ch->ptype() = PT_EECED;
        ch->size() = IP_HDR_LEN + ah->size();
        ch->error() = 0;
        ch->addr_type() = NS_AF_NONE;
        ch->direction() = hdr_cmn::DOWN;
        ch->prev_hop_ =  mNodeIndex;
        ch->next_hop() = IP_BROADCAST;
        ih->saddr() =  mNodeIndex;
        ih->daddr() = IP_BROADCAST;
        ih->ttl() = 1;
        ah->cadv_pkt_type() = EECEDTYPE_CADV;
        ah->cadv_pkt_bid() = mClusterAdvertisementSeqno;
        ah->cadv_pkt_src() = mNodeIndex;
        ah->cadv_pkt_hopcount() = 1;
        ah->cadv_pkt_timestamp() = Scheduler::instance().clock();
        ah->cadv_node_role() = mNodeRole;
        Scheduler::instance().schedule(mLLTarget, p, 0.0);
        mClusterAdvertisementSeqno++;

        printf("The leader node %d is broadcasting a cluster head message...", mNodeIndex);
        }
void EECED::send_join_request(int cadv_pkt_bid,

                         nsaddr_t nexthop,

                         nsaddr_t dst)
{

                Packet* p = Packet::alloc();
                struct hdr_cmn* ch = HDR_CMN(p);
                struct hdr_ip* ih = HDR_IP(p);
                struct hdr_eeced_jreq* ah = HDR_EECED_JREQ(p);
                ch->ptype() = PT_EECED;
                ch->size() = IP_HDR_LEN + ah->size();
                ch->addr_type() = NS_AF_INET;
                ch->error() = 0;
                ch->direction() = hdr_cmn::DOWN;
                ch->prev_hop_ = mNodeIndex;
                ch->next_hop() = nexthop;
                //setup IP header
                ih->saddr() =  mNodeIndex;
                ih->daddr() = nexthop;
```

```
			ih->ttl() = 1;
			//Packet initialisation
			ah->jreq_pkt_type()			=
EECEDTYPE_JREQ;
			ah->jreq_pkt_bid() = cadv_pkt_bid;
			ah->jreq_pkt_src() = mNodeIndex;
			ah->jreq_pkt_dst() = dst;
			ah->jreq_pkt_hopcount() = 1;
			ah->jreq_pkt_timestamp()		=
Scheduler::instance().clock();
			ah->jreq_node_role() = mNodeRole;

		Scheduler::instance().schedule(mLLTarget,	p,
0.0);
				printf("Normal node %d is
reply with a join request to leader node
%d",mNodeIndex,dst);
		 }
//===========================================
==============================
//
			HELPER FUNTIONS
//===========================================
==============================
//Checks whether a duplicate packet has been
received(To be revised)

//===========================================
==============================
//
		GENERIC PACKET RECEPTION FUNCTION
//===========================================
==============================
void EECED::recv(Packet* p, Handler*) {
		struct hdr_cmn* ch = HDR_CMN(p);
		struct hdr_ip* ih = HDR_IP(p);
		//EECED Packets
		if(ch->ptype() == PT_EECED){
			//Drop if time to live has expired
			if(ih->ttl() == 0){
				drop(p, DROP_RTR_TTL);
				return;
			} else
			{
				//Decrement the ttl and
send to the appropriate receive method
				ih->ttl()--;
				recv_eeced(p);
				return;
			}	}
		else
		//Any Packet type
		//  Must be a packet I'm originating
		if((ih->saddr() == mNodeIndex) && (ch-
>num_forwards() == 0))
		{
			// Add the IP Header. TCP adds the
IP header too, so to avoid setting it twice,
			// we check if  this packet is not a
TCP or ACK segment.
			if (ch->ptype() != PT_TCP && ch-
>ptype() != PT_ACK) {
				ch->size() += IP_HDR_LEN;
			}	}
		// I received a packet that I sent.  Probably
routing loop.
		else if(ih->saddr() == mNodeIndex) {
			drop(p, DROP_RTR_ROUTE_LOOP);
			return;
		}
		//Packet I'm forwarding...
		else{
			if(--ih->ttl_ == 0) {
				drop(p, DROP_RTR_TTL);
				return;
			}		}
		// Forward the packet
		//forward(p, ih->daddr(), 0);
}
//===========================================
==============================
//			HANDLE RECEPTION OF DIFFERENT
EECED PACKET TYPES
//===========================================
==============================
//perfect
void EECED::recv_eeced(Packet* p){
		struct hdr_eeced* ah = HDR_EECED(p);
			switch(ah->pkt_type()){
			case EECEDTYPE_EREQ:
				recv_energy_request(p);
				break;
	case EECEDTYPE_EREP:
				recv_energy_reply(p);
				break;
	case EECEDTYPE_CINF:
		recv_leader_election(p);
		break;
			case EECEDTYPE_CADV:

		recv_cluster_advertisement(p);
				break;
			case EECEDTYPE_JREQ:
				recv_join_request(p);
				break;
			default:
				//Do nothing
				break;
		}	}
//Receives energy request
void EECED::recv_energy_request(Packet *p) {
		struct hdr_ip* ih =  HDR_IP(p);
		struct	hdr_eeced_ereq*	ah	=
HDR_EECED_EREQ(p);
		int ereq_pkt_bid = ah->ereq_pkt_bid();
		nsaddr_t ereq_pkt_src = ah->ereq_pkt_src();
		int	ereq_pkt_hopcount	=	ah-
>ereq_pkt_hopcount();
		double	ereq_pkt_timestamp	=	ah-
>ereq_pkt_timestamp();
		int ereq_node_role = ah->ereq_node_role();
printf("Normal node %d is receiving energy request
packet from node %d which originates from elector
node %d...\n",
		mNodeIndex, ih->saddr(), ereq_pkt_src);
		//I am originating this packet: Drop it
```

```
        if(ereq_pkt_src == mNodeIndex)
        {
        printf("Elector node dropping a packet that it
sent...\n", mNodeIndex);
        Packet::free(p);
                return;
        }
        if(mNodeRole == NORMAL)
        {
        //Packet forwarding
        RouteEntry *rt = rt_lookup(ereq_pkt_bid,
ereq_pkt_src);
        if(rt == NULL)
                {
                //active_rx_count++;
                //printf("active rx count: %d\n",
active_rx_count);
                //Inserts a new route
                rt_insert(ereq_pkt_bid,
                 ereq_pkt_src,
                 ih->saddr(),
                 mNodeEnergy,
                ereq_pkt_hopcount);
                //Increment the hopcount

                //ereq_pkt_hopcount++;
                //Forwards the energy request
packet
                //printf("Route cache size after
forwarding: %d\n", mRouteCache.size());
                }else
                {
                        //Drop the packet it is a
duplicate
                        printf("Duplicate packet
dropped..\n" );
                        Packet::free(p);
                }
                //We are sure that the packet has
been inserted : Perform second lookup
                rt = rt_lookup(ereq_pkt_bid,
ereq_pkt_src);

                //Checks whether there is a valid
route
                if(rt != NULL){

                        if(ereq_node_role ==
ELECTOR){
                        mElectorNode =
(MobileNode*)(Node::get_node_by_address(ereq_pkt_s
rc));
                        double elector_energy =
mElectorNode->energy_model()->energy();
                        if(mNodeEnergy >
elector_energy){

send_energy_reply(ereq_pkt_bid,
                         rt->rt_nexthop,
                         ereq_pkt_src);
                        Packet::free(p);
                        //Clear the cache
```

```
                              rt_remove(ereq_pkt_bid,
ereq_pkt_src);
                              printf("Routibg Table
size%d\n", mRouteCache.size());
                                        }  }      }
                        //printf("Route cache size after
sending reply: %d\n", mRouteCache.size());
                        //printf("Route cache entry: %d\n",
mRouteCache.back()->rt_nexthop);
                }   }
//Handles energy reply packets received
void EECED::recv_energy_reply(Packet *p) {
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_eeced_erep* ah =
HDR_EECED_EREP(p);
        int erep_pkt_bid = ah->erep_pkt_bid();
        int erep_pkt_rep_id = ah->erep_pkt_rep_id();
        nsaddr_t erep_pkt_src =  ah->erep_pkt_src();
        nsaddr_t erep_pkt_dst =  ah->erep_pkt_dst();
        int erep_pkt_hopcount = ah-
>erep_pkt_hopcount();
        //Record the timestamp when energy reply
was received.
         ah->erep_pkt_timestamp() =
Scheduler::instance().clock();
         printf("Packet reply timepstamp:%.6f\n", ah-
>erep_pkt_timestamp());
        int erep_node_role = ah->erep_node_role();
        double erep_node_energy = ah-
>erep_node_energy();
        //Normals nodes forwards the energy reply
packet
        if(mNodeRole ==  NORMAL){
        }
        else if(mNodeRole == ELECTOR){
                printf("Node %d is receiving an
energy reply from %d\n", mNodeIndex, erep_pkt_src);
                //mTimestamps.push_back(ah-
>erep_pkt_timestamp());
                mEnergyRepliesCache.push_back(p);
                printf("Replies cache size %d\n",
mEnergyRepliesCache.size());
                //Packet::free(p);
        }   }
// Handles received leader selection message
void EECED::recv_leader_election(Packet* p){
        printf("Node %d received a leader selection
message....\n", mNodeIndex);
        struct hdr_eeced_cinf* ah =
HDR_EECED_CINF(p);
        int node_dest_addr =  ah->cinf_pkt_dest();
        int src_role = ah->cinf_node_role();
        if(mNodeRole == NORMAL){
                //Checks whether current node
index match the one contained in the received packet
                if(src_role == ELECTOR &&
node_dest_addr == mNodeIndex){

        mElectionReceiveTimestamp =
Scheduler::instance().clock();
                        mNodeRole = LEADER;
                        printf("I node %d is
selected as a leader\n", mNodeIndex);
```

```
                                //
send_cluster_advertisement();
                                Packet::free(p);
                        }else
                        {
                                //Do nothing for now

            //forward(p,IP_BROADCAST,0.0);
                        }               } }
//Handle received cluster leader broadcast packet //
target: normal nodes
void EECED::recv_cluster_advertisement(Packet *p){
            printf("Inside receive cluster advertisement");
            struct          hdr_eeced_cadv*         ah          =
HDR_EECED_CADV(p);
   int bid =  ah->cadv_pkt_bid();
   //Initiate a join request after receiving a cluster head
advertisement
   if(mNodeRole == NORMAL){
               //send_join_request(routeCache, bid);
               Packet::free(p);
            }
   //Drop the packet if it comes back to me as the sender
or received by elector node
   else  if(mNodeRole  ==  LEADER  ||  mNodeRole  ==
ELECTOR)
   {
            Packet::free(p);
   }  }

//Handle received join request packet   // target: Leader
node
void EECED::recv_join_request(Packet *p){
            printf("Inside receive join request");
            struct          hdr_eeced_jreq*         ah          =
HDR_EECED_JREQ(p);
            if(mNodeRole == NORMAL || mNodeRole ==
ELECTOR){
            }
            else if(mNodeRole ==  LEADER){
                        mJoinRequestCount++;
            } }
//============================================
=============================================
//                              ROUTING    MANAGEMENT
FUNCTIONS
//============================================
=============================================
//Inserts a new route
void EECED::rt_insert( int id,

nsaddr_t dst,

nsaddr_t nexthop,

double energy,
                                                        int
hopcount){

            RouteEntry* rt =  new RouteEntry(id, dst);
            rt->rt_nexthop = nexthop;
            rt->rt_energy = energy;
            rt->rt_hopcount = hopcount;
            mRouteCache.push_back(rt);
            }
//Removes an existing route
void EECED::rt_remove(int id, nsaddr_t dst){
            std::vector<RouteEntry*>::iterator it;
            if(mRouteCache.size() > 0) {
                        for(it  =  mRouteCache.begin();  it  <
mRouteCache.end(); it++){
                                    if(         id     ==        (*it)-
>rt_seqno &&
                                                dst      ==      (*it)-
>rt_dst){

            mRouteCache.erase(it);
                                    } } }  }
//Retrieves a valid route
RouteEntry*  EECED::rt_lookup(int id, nsaddr_t dst){
            std::vector<RouteEntry*>::iterator it;
            if(mRouteCache.size() > 0)
            {
                        for(it  =  mRouteCache.begin();  it  <
mRouteCache.end(); it++){
                                    if( id == (*it)->rt_seqno &&
dst == (*it)->rt_dst){
                                                return *it;
                                    }  }  }
            return NULL;
}
//Checks whether we have received a duplicate packet
bool EECED::is_packet_duplicate(int bid, nsaddr_t bsrc){
            std::vector<RouteEntry*>::iterator it;
            for(it     =     mRouteCache.begin();     it     !=
mRouteCache.end(); it++){
                        if(bid == (*it)->rt_seqno && bsrc ==
(*it)->rt_dst)
                                    return true;
            }
            return false;
}
```

APPENDIX C: ECDBC TCL

```
#=============================================
===============
# eeced.tcl -  a script to start the simulation
#=============================================
===============
#Node options
set val(chan)    Channel/WirelessChannel      ; # Channel
type
set val(prop)      Propagation/TwoRayGround ; # radio-
propagation model
set val(ant)     Antenna/OmniAntenna          ; # Antenna
type
set val(netif)   Phy/WirelessPhy              ; # Network
interface type
set val(mac)     Mac/802_11                  ; # Mac type
set val(ifq)      Queue/DropTail/PriQueue   ; # Interface
queue type
set val(ifqlen)  50                       ; # Max interface queue
length
set val(ll)      LL                     ; # Link layer type
set val(nn)       200                      ; # number of mobile
nodes
set val(rp)        AODV                      ; # ad-hoc routing
protocol
set val(x)       400               ;  #  x  dimension  of  the
topography
set val(y)        400                     ; # y dimension of the
topography
set val(energymodel)  EnergyModel          ;   #    Energy
Model
set val(initialenergy) 5.0                    ; # Remaining
node energy in joules
set val(seed)   0.0                          ;  #  simulation
seed
set val(stop)    1000                        ; # simulation
time in seconds
#set val(rxPw)         1.0                  ;     #     Node
receiving power in Watts
#set val(txPw)    2.0                      ;     #     Node
transmission power in Watts
#set val(idlePw)  0.5                     ;  #  Node  idle
power in Watts
#set val(sleepPw) 0.005                            ; # Node
sleeping power in Watts
# Initialize the SharedMedia interface with parameters
to make
# it work like the 914MHz Lucent WaveLAN DSSS radio
interface
#Phy/WirelessPhy set CPThresh_ 10.0
#Phy/WirelessPhy set CSThresh_ 1.559e-11
#Phy/WirelessPhy set RXThresh_ 3.652e-10
#Phy/WirelessPhy set Rb_ 2*1e6
#Phy/WirelessPhy set Pt_ 0.3 ;#0.2818
#Phy/WirelessPhy set freq_ 914e+6
#Phy/WirelessPhy set L_ 1.0
 #Create the event scheduler
set ns [new Simulator]
#$ns use-scheduler Heap
#Open trace files
set tracefd [open edbc-out.tr w]
set namtracefd [open edbc-out.nam w]
#Write to trace files
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtracefd $val(x) $val(y)
# set up topography object
set topo   [new Topography]
$topo load_flatgrid $val(x) $val(y)
set god_ [create-god $val(nn)]
#Node configuratio
$ns node-config -adhocRouting $val(rp)\
                                        -llType $val(ll) \
                                        -macType
$val(mac) \
                                        -ifqType  $val(ifq)
\
                                        -ifqLen
$val(ifqlen) \
                                        -antType
$val(ant) \
                                        -propType
$val(prop) \
                                        -phyType
$val(netif) \
                                        -channel      [new
$val(chan)]\
                                        -topoInstance
$topo \
                                        -agentTrace ON \
                                        -routerTrace   ON
\
                                        -macTrace OFF \
                                        -movementTrace
OFF \
                                        -energyModel
$val(energymodel)\
                                        -rxPower 0.3 \
                                        -txPower 0.6 \
                                        -idlePower 0.2 \
         -sleepPower 0.05 \
                                        -sleepTime 2 \
                    -transitionPower 0.2 \
                    -transitionTime 0.005 \
                                        -initialEnergy
$val(initialenergy)
#Create new nodes and disable random motion
puts "Creating mobile nodes..."
for { set i 0 } { $i < $val(nn) } { incr i } {
   set mnode($i) [$ns node]  ;#create a new mobile node
   $mnode($i)  random-motion  0  ;#  disable  random
motion
            $god_ new_node $mnode($i)
}
#Position the nodes randomly across the topography
puts "Setting random positions for mobile nodes..."
for { set i 0 } { $i < $val(nn) } { incr i } {
  $mnode($i) set X_ [expr {int(rand()*$val(x))}]
  $mnode($i) set Y_  [expr {int(rand()*$val(y))}]
  $mnode($i) set Z_ = 0.0
}
#set the initial position of the nodes in nam
puts "Setting initial positions for NAM.."
for { set i 0 } { $i < $val(nn) } { incr i } {
```