# INTELLIGENT ROUTING ALGORITHM IN WIRELESS SENSOR NETWORKS

Submitted in partial fulfillment of the requirements
of the degree of

**B. E. Computer Engineering**

By


| | |
|---|---|
| **Devang Jhaveri** | **60004120045** |
| **Miti Jhaveri** | **60004120046** |
| **Jehan Kandawalla** | **60004120049** |


Guide:


**Aruna Gawade**
Asst. Professor




Department of Computer Engineering
D.J. Sanghvi College of Engineering
Mumbai– 400 056


University of Mumbai
2015-2016

# CERTIFICATE

This is to certify that the project entitled **"Intelligent Routing Algorithm in Wireless Sensor Networks"** is a bonafide work of **Devang Jhaveri (60004120045), Miti Jhaveri (60004120046), and Jehan Kandawalla (60004120049)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of B.E. in Computer Engineering

**(Name and sign)**
**Internal Guide**

**(Name and sign)**
**Head of Department**

**(Name and sign)**
**Principal**

# Declaration

I/We declare that this written submission represents my/our ideas in my/our own words and where others' ideas or words have been included, I/We have adequately cited and referenced the original sources. I/We also declare that I/We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my/our submission. I/We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

| | | |
|---|---|---|
| --- | --- | --- |
| (Signature) | (Signature) | (Signature) |
| Devang Jhaveri<br>(60004120045) | Miti Jhaveri<br>(60004120046) | Jehan Kandawalla<br>(60004120049) |

Date:

# Abstract

A Wireless Sensor Network (WSN) is a network of distributed sensor nodes which are capable of monitoring their environment through sensors. Because of the various types of sensors that are available, WSNs have a wide of applications, each with their own characteristics and requirements. Although WSN platforms are the subject of constant evolution, most sensor nodes remain reliant on a limited capacity power source. This limitation requires WSN software to be economical with available resources, as applications often require sensor nodes to be located in remote locations, making battery replacement expensive or even impossible.

The wide range of available applications for WSNs means that developing algorithms which can run on sensor nodes and which are suitable for multiple applications becomes ever more challenging. However, one constraint sensor nodes are likely to face in most scenarios is a limited power source. Furthermore, because sensor nodes are mostly located in extreme locations, they often form very irregularly connected network topologies. To ensure satisfying network performances a routing algorithm has to be able to cope with these irregular topologies, while conserving energy whenever possible in order to maximize the network's lifetime.

Reinforcement Learning (RL) is a machine learning variation that enables an agent to learn which actions it can take when it is executing a task in order to maximize a long-term reward. The principles from RL have already been applied to the routing problem, resulting in an algorithm that delivers good network performance in wired networks by using the communication delay across the network as a metric for network performance. A RL based routing algorithm requires little information about its environment and it is able to adjust its routing behaviour to dynamical conditions during the network's lifetime. Because of this

RL based routing algorithms are very suitable for WSNs and recent work has applied some of the existing RL routing principles on WSNs by using the energy of sensor nodes as a metric.

In this project we implement Q-routing algorithm for WSNs. The algorithm is able to significantly improve the lifetime of a WSN in certain scenarios by propagating a sensor node's energy information throughout the network. The algorithm also combines both metrics for energy and message delay into a single algorithm, resulting in a routing policy able to balance the network performance and network lifetime.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Description

Wireless Sensor Networks are spatially distributed autonomous sensors to monitor physical and environmental conditions like temperature, pressure, sound, humidity etc. and cooperatively send their aggregated data to a data center (sink node) for further analysis. A typical sensor node in the WSN consists of several sensors capable of sensing valuable information. In today's era, wireless sensor nodes are being deployed for a variety of applications. Applications like environmental monitoring, industrial process control, machine health monitoring, smart city development are just a few examples of areas where WSNs are used. Wireless sensor nodes will be extensively used in the future world of "Internet of Things".

WSN have a typical minuscule architecture consisting of set of sensors, a micro controller unit, wireless antenna, program memory and a battery power source. Their limited hardware enables sensor nodes to be deployed at odd locations with acceptable lifetimes on regular portable batteries. However, their odd locality of deployment makes it very difficult to maintain these sensor nodes. In majority of the cases, it is impossible to recharge their batteries and hence, renders it useless post its lifetime. Therefore, the scope of research in the field of Wireless Sensor Network has shifted towards developing efficient routing algorithms to make optimal use of the available battery and thus, increase the average lifetime of the network.

## 1.2 Problem Formulation

Wireless sensor networks have numerous issues related to network communication. Firstly, majority of the sensor nodes are deployed at dynamically changing locations like underwater. Due to this, their link quality constantly changes. Secondly, these nodes have limited range of wireless signal and thus, packets have to be sent to closer neighboring nodes before they reach the data center or the sink nodes. In other words, traffic mostly travels along multiple hops before it reaches its destination. Energy is one most dynamically changing factor which drains at every operation performed by the sensor node. Moreover, to save their battery lives, sensor nodes go into sleep or inactive state, making themselves unavailable for data forwarding.

Considering all these dynamically changing factors, a static algorithm followed in traditional network topologies is unpropitious to be implemented in WSNs. These dynamic quantities have persuaded researchers to look for Machine Learning solutions to develop efficient routing algorithms. Reinforcement learning is a branch of Machine Learning that allows the agent to learn which actions are most suitable when executing a certain task. The biggest advantage of Reinforcement Learning which makes it most favorable for WSNs is that it can be deployed in an environment without prior information. It does not require any prerequisite training set to perform the operation. The agent learns which action to take by a process of trial and error. Every time the agent performs an action, it gets a rewards or feedback which helps the agent to learn

how good or bad the action was so that it performs better the next time it executes the same action.

## 1.3 Motivation

In relation to the Quality of Service of WSN, there are two major performance evaluation parameters – network latency and average lifetime. It is very difficult to design a single routing algorithm which can achieve an optimal solution for both of these parameters. Network latency is defined as the time lapse between a packet that is sent from its source node to reach its destination node whereas average lifetime of a network is defined as the time lapse before the battery of the first node drains out. Static routing algorithms used in wired communications strive to minimize the network latency by selecting the best routing path from source node to the sink node (destination). However, if the same sets of nodes (which fall under the optimal route) are active to send the packet majority of the time, the battery life of these nodes will drain much faster than the other nodes in the network resulting in overall attenuation of average lifetime of the network. Therefore, it is necessary to design a routing algorithm which gives us acceptable network latency and at the same time enhances the average lifetime of the network by keeping all nodes equally active.

## 1.4 Problem Solution

Our proposed algorithm intends to tackle the issue of limited lifetime of the Wireless Sensor Network with marginal increase in the network latency. Our algorithm takes care of three parameters before making a routing decision:

- **Battery life**: The battery life of the adjacent nodes indicating the degree of life left in those nodes. This helps us to increase the average lifetime of the network.

- **Load**: The number of packets waiting in the queue of the neighbouring nodes. This helps us to increase the average network latency in WSNs.

- **Cost**: The cost associated with sending of a packet to the adjacent nodes.

These three parameters are returned as rewards on the basis of which the routing decision is made. Hence, using Reinforcement Learning our proposed algorithm intends to tackle the above discussed issue.

### 1.4.1 Advantages of Our Algorithm

- **Increase Network Lifetime** – Using Reinforcement Learning, the algorithm makes optimal use of network battery life by incorporating the remaining battery life of the as one of the factors in calculation of the Q-value and hence, influences the routing decision.

- **Minimize Network Latency** – Our proposed algorithm minimizes the network latency by

distributing packets evenly to all the nodes in the network.

- **Increased Network Reliability** – No packets are lost in transmission as they don't have to wait in the waiting queue.

## 1.5 Scope of the project

The proposed algorithm will increase the network lifetime of Wireless Sensor Networks with minimal network latency. The scope of the project can be extended by finding a function which incorporates global load of the network into consideration. In majority the cases load is excessive on node which are closer to the sink than others. Having access to the information about congestion at these nodes will help the source node to make better routing decisions and thus, improve the average latency of the network. We intend to test our proposed algorithm on various homogenous and heterogeneous networks. We will specially focus on bottleneck cases where load multiplication factor plays a crucial role.

# Chapter 2

## Review of Literature

### 2.1. Introduction

Wireless sensor networks raise several issues related to network communication. Because of the limited range of the wireless signal, traffic will mostly have to travel along multiple hops before reaching its destination (a sink node). But in order to increase their lifetime, nodes will spend most of their time in a sleep state, making them unavailable to forward other data. This means that some paths through the network might not be available at all times. The possibilities of a hardware malfunction or a drained battery also increase the odds of a changing network topology. The combination of limited wireless antenna range and uncertainty about a node's availability means that Wireless Sensor Networks tend to have irregularly connected topologies that change dynamically during the network's lifetime.

The multi-hop routing in wireless sensor networks is possible because sensor nodes tend to be densely deployed near an area of interest (also called phenomenon in [3]). The exact positions of nodes however are not always determined before deployment. This allows for rapid deployment of networks in extreme terrains, but it also contributes to the irregular nature of network topologies. The combination of limited information before deployment and irregular topologies also means that sensor networks should possess self-organising abilities. Another issue that needs to be considered is the impact of using the wireless radio on the lifetime of a node. Since using the wireless radio is a big drain on a node's battery life [2], using it too often can result in heavily reduced lifetime. So if one node needs to forward considerably more packets than other nodes its lifetime will be far below the average lifetime of other nodes. This can have a serious impact on general network performance, since a lot of data will now have to be re-routed along other paths.

A big difference in lifetime between nodes can also be undesirable when multiple nodes in critical locations suffer from this problem. This would require more regular maintenance which can be very costly in some scenarios (eg. sensors located in remote or dangerous locations [1]). These issues are recognised by Akyildiz et al. in their survey on wireless sensor networks [3]. In their survey they propose a model for a protocol stack for wireless sensor networks. This stack is shown in figure and consists of five layers which are from top to bottom; the application layer, the transport layer, the network layer, the data link layer and the physical layer. The problem of routing data, which is the subject of this master thesis, is part of the network layer. Readers with experience in data networks might notice that this stack heavily resembles the OSI model protocol stack [4].

It is important however to note that at each layer, protocols need to take into account three domain specific constraints for wireless sensor networks. These constraints concern power management, mobility management and task management. As general guidelines Akyildiz et al. suggested that protocols at the network layer should take the following principles into account for their design:

Power efficiency is always an important consideration. Sensor networks are often data-centric. Data aggregation is only useful when it does not hinder the collaborative behaviour of other nodes. An ideal sensor network has attribute-based addressing and location awareness. The scenario we are looking at is limited purely to the routing of data from sensors nodes to a sink and assumes communication through the network to be one way from nodes to the sink, so we do not need to worry about a data-centric approach [5]. The fourth item from their guideline is somewhat out of reach for our research, as even in today's modern sensor network hardware location awareness almost always comes at a great energy cost. Furthermore our experiments are focussed on non-mobile sensor networks.

## 2.2. Routing in WSNs

In their survey on wireless sensor networks, [3] presented an overview of protocols that were available or under development for different layers of the sensor networks protocol stack. We will use the remainder of this section to discuss some of the protocols that are available for the network layer of the sensor network protocol stack.

- Small minimum energy communication network (SMECN)
  This approach is proposed by [6] and works by computing an energy efficient subnetwork for a given communication network. The algorithm works by taking into account energy costs for different communication links when it attempts to construct a minimum energy graph.
- Flooding
  An old technique that has been used in many networks and that is also applicable on WSNs [7]. Each node that receives a message broadcasts it to all of its neighbours, increasing the messages hop count. This process is continued until the message expires after reaching the maximal amount of hops. Over the years this techniques has shown many shortcomings. In densely connected networks messages can cause implosions of traffic, because two nodes with a lot of common neighbours may receive each other's messages once for every neighbour they broadcasted too, causing for a lot duplicated messages and a lot of wasted resources.
- Gossiping
  This technique is a variation on Flooding that attempts to avoid the implosion of duplicated messages by sending each message only to one randomly selected neighbor instead of broadcasting it to all neighbours [5]. While this approach works to conserve resources, it can also be very slow to deliver messages.
- Sensor protocols for information via negotiation (SPIN)
  In [4] the authors propose a collection of adaptive protocols that are designed to overcome the shortcomings of classic algorithms such as flooding. The algorithms are based on two main principles; Sensor nodes can operate more efficiently by sending information about their sensor data instead of the actual sensor data and sensor nodes should monitor changes in their available resources, e.g., energy level, available bandwidth etc. The SPIN system is based on data-centric routing [7], this means that sensors periodically broadcast an availability of updated data and wait for a request rom a sink before sending any actual data.
- Sequential assignment routing (SAR)

In [8] a set of algorithms are proposed. These algorithms handle the organisation, management

and mobility management operations in a sensor network. Self-organizing MAC for sensor networks (SMACS) is a distributed protocol that enables a collection of sensor nodes to discover their neighbours and establish communication schedules without the need for centralised coordination. The eavesdrop and register (EAR) algorithm is designed to support seamless interconnection of the mobile nodes. The EAR algorithm is based on invitation messages and on the registration of stationary nodes by the mobile nodes. The SAR algorithm creates multiple trees where the root of each tree is an one hop neighbour from the sink. Each tree grows outward from the sink while avoiding nodes with a low performance (i.e., low throughput/high de- lay) and low energy reserves. At the end of this procedure, most nodes belong to multiple trees. This allows a sensor node to choose a tree to relay its information back to the sink. There are two parameters associated with each path, i.e., a tree, back to the sink:

- Energy resources:

The energy resource is estimated by the number of packets, which the sensor node can send, if the sensor node has exclusive use of the path.

- Additive QoS metric: A high additive QoS metric means low QoS.

The SAR algorithm selects the path based on the energy resources and additive QoS metric of each path, and the packet's priority level. As a result, each sensor node selects its path to route the data back to the sink. Also, two more algorithms called single winner election and multi-winner election handle the necessary signaling and data transfer tasks in local cooperative information processing.

- Directed diffusion (DD)

The directed diffusion data dissemination paradigm is proposed in [7]. Directed diffusion is a data-centric routing protocol, this means that it requires the sink in a WSN to broadcast an interest for a certain type of data (this is called a query). This interest is propagated through the network to all sensors. All sensors remember the type of information that is queried before forwarding the query further into the network. When the query is propagated, all nodes also update the query with a timestamp and a gradient, nodes receiving a query store this gradient information in a local cache before updating the message. This way the query actually builds the routing tree, so when a sensor node has information that is of interest to the sink, it knows where to send the information. In order to achieve a good routing behavior the sink needs to refresh and reinforce the interest as soon as it starts to receive data from a source.

- Q - Routing

Q-Learning is one of the reinforcement learning techniques which uses the actions, states and rewards as described in the previous section. Depending upon the requirement of the system, the discount factor $\gamma$ can be introduced. If the value of $\gamma$ is nearer to zero, the system will seek short term future rewards whereas if $\gamma$ is nearer to one, then the system will seek long term future rewards. The value of learning rate $\alpha$ decides the fate of the information that is acquired regularly. If the value of $\alpha$ is nearer to zero, the system will discard new information and not

learn anything new whereas if the value is nearer to one, the system will discard old information and learn the new information. Q-learning is used as an inspiration to develop Q-Routing algorithm [9]. Q-routing was developed to distribute the network load in an irregular topology and also to operate in dynamically changing network environments and topologies.

Let $Q_a(d,b)$ be the time that the node a estimates to send a packet to destination d via a neighboring node b. This estimate includes the delay that the packets may face at node a. When a routing decision needs to be made, each node by default chooses the neighbor with the lowest Q-value to the destination. This value is stored by each node in local tables for each neighbor-destination route available. When a packet is sent to node b, it immediately responds to node a with T which is the estimate calculated by node b for the packet to reach d.

$$T = \min_{c \in N} Q_b(d,c)$$

There are various factors that are considered when the local tables of estimate are updated. They include the variable T, the time taken by the node to receive the message and forward it that is q and the link quality that is s. The update rule is given as:

$$Q_a^{t+1}(d, b) = Q_a^t(d, b) + \Delta Q_a(d, b)$$

$$\text{Where } \Delta Q_a^{t+1}(d, b) = \alpha\,(q + s + T - Q_a^t(\text{d, b}))$$

Initially, the estimated time that is calculated contains errors as the ideal routing paths of the network are unknown. However, as the network learns, the tables are updated and eventually a stable table will emerge on converging. In case of WSNs, the nodes only need to store a single Q-value as the destination is only the sink.

- **Energy Efficient Q- routing**

Energy efficiency is an important criteria to enhance the network lifetime. The network lifetime is the time taken by any node in the network to fail due to exhaustion of energy. For instance, if a particular route is marked as the optimal route, and each time if the same node wishes to send packets to the sink, it will repetitively use the same path. This will cause congestion and in turn delay and also depletion of energy of the same nodes. However, if the energy is also considered as a factor in the cost function while selecting the optimal path, the state of the above two issues can be improved. Whenever a node receives feedback about the estimated time T and the energy level of a neighboring node b containing its energy level E_a^t(b), the node will update the energy weighted Q-value E(Q_a^t(b)) for that neighbor in the following way:

$$\text{E}\,(Q_a^t(b)) = Q_a^t(b) * \text{w}(E_a^t(b))$$

In the above formula, w is the energy weighing function. A balance needs to be struck between the importance of energy consumption and the latency of network. Depending upon the requirement, a linear function or exponential function can be used. In exponential functions, initially the impact is small on energy management; however, it becomes very aggressive when a node reaches the end of its life cycle [10].

# Chapter 3

## System Analysis

In this chapter, we shall discuss what is expected from our system. It is categorized as functional and non – functional requirements. This chapter also specifies the minimum hardware and software need to implement the specified requirements. Later, functional requirements are described as use case diagram.

## 2.3. Functional Requirements

A functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behaviour, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. The functional requirements are captured in use cases. Generally, functional requirements are expressed in the form "system must do <requirement>".

### 3.1.1. Criticality:

In the given software product, the functionality plays a very important role. It tells us how the system reacts to a set of inputs. The criticality of these functions tells us the performance and success of the software product. The product should be tested under heavy load and stress conditions. Under these conditions the functionality of the software should not hinder. Moreover, the failure of functionality will result in failure of the entire software. Thus the product should be such that it works efficiently under different load conditions and give similar results irrespective of the stress.

### 3.1.2. Technical Issues:

Major threat to the project that is possible is that the system might get crashed or maybe a virus would cause a major problem. But, for these problems backup should be taken so that we do not lose the important data as well as we keep a continuous check onto the virus that try to affect the project and the software and eliminate by using an antivirus software preinstalled in the computer.

### 3.1.3. Cost and Schedule:

As far as the cost of the project is concerned it has been very cost efficient. This includes all the costs of hardware, software and communication. General estimation of the cost is done based on the size of our software.

### 3.1.4. Risk:

The various risks like management and the budget risks should be taken care of by the policies that are imposed on the project. Also, we plan to have proper communication with the customer for the maintenance of the software. We are trying to make use of the proactive risk strategy to avoid risks. Also, we'll prepare a risk management plan to avoid and fight the risks which works according to the policies planned.

**3.1.5. Dependency with other requirements:**
The basic requirement needed for the software to be in working condition is the database. The data can be handled using many alternatives present.

## 2.4. Non-Functional Requirements

Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation. Non-functional requirements are "system shall be <requirement>".

### 2.4.1. Reliability:

The R.I. generation will impact the decision of normality and abnormality for a node and will also impact the packet delivery. Hence to make system reliable, the guidelines must be followed very carefully under the guidance of a field expert.

### 2.4.2. Maintainability:

One thing it explains is the Application extendibility. The application should be easy to extend. The code should be written in a way that it favours implementation of new functions. The design should be in such a way that future functions could be implemented easily to the application and databases of new nodes could be loaded easily. Other thing it explains is Application testability. Test environments should be built for the application to allow testing of the applications different functions. It is used for testing the application.

### 2.4.3. Portability:

It explains the Application portability. We are trying to build it in such a way that it can be used on multiple OS.

### 2.4.4. Binary Compatibility:

Binary Compatibility means running of the software not only on one system but also on another. The application is able to run on some other host machine apart from the one on which it is made.

### 2.4.5. Extensibility:

It explains how extensible is the software i.e. if you try to put new functionality or by modifying the existing one. Our software is extensible in terms of adding new database schemas and loading new data. Our system can also include new test parameters for future use.

## 2.5. Specific Requirements

### 2.5.1. Minimum Hardware Requirements

For installation:

1. Any Intel or AMD x86 processor
2. 1 GB for MATLAB only
3. 2 GB RAM; With Simulink Coder, 4 GB is recommended

## 2.6. Use case Diagrams and descriptions

**3.4.1. Scenario:** A node is selected for sending a packet**.**

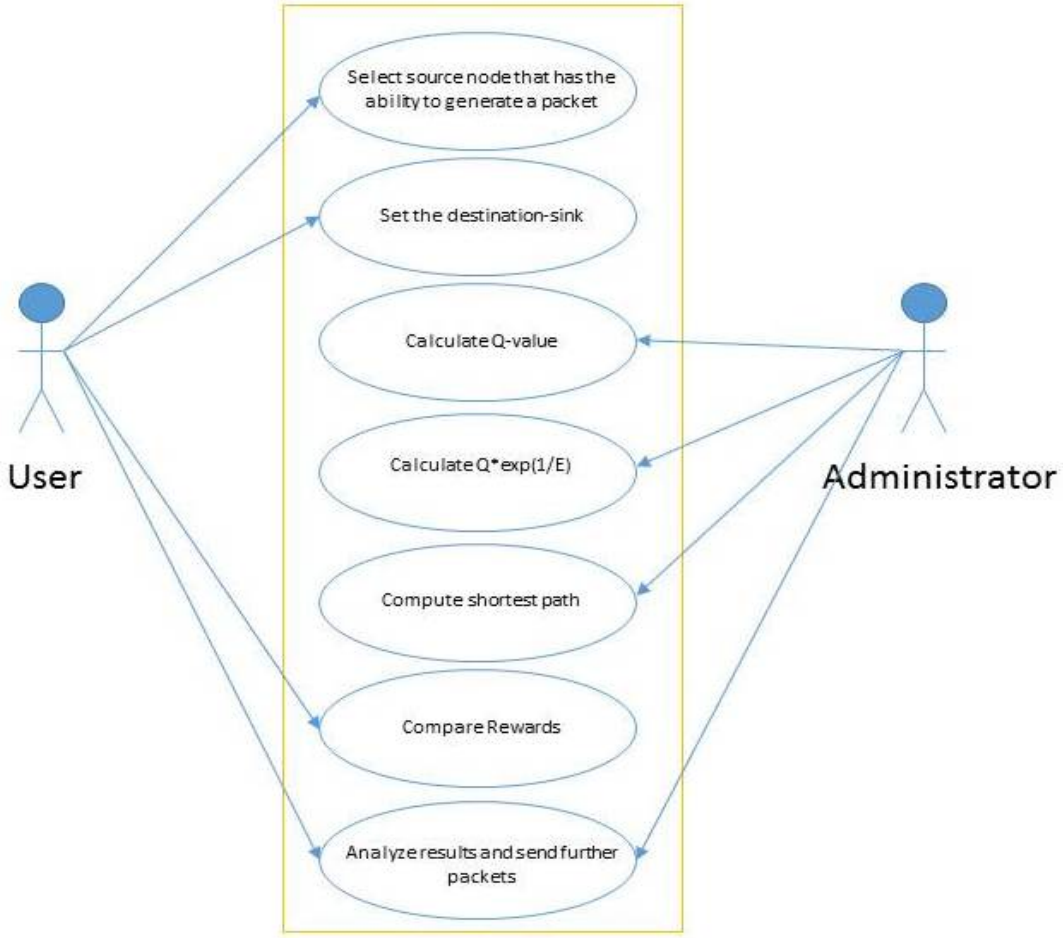


Figure 3.1 Use Case Diagram

# Chapter 4

## Analysis Modelling

### 4.1. Data Modelling

This section examines data objects independently of processing and creates a model at the customer's level of abstraction. The section 4.1.1 indicates how data objects relate to one another.

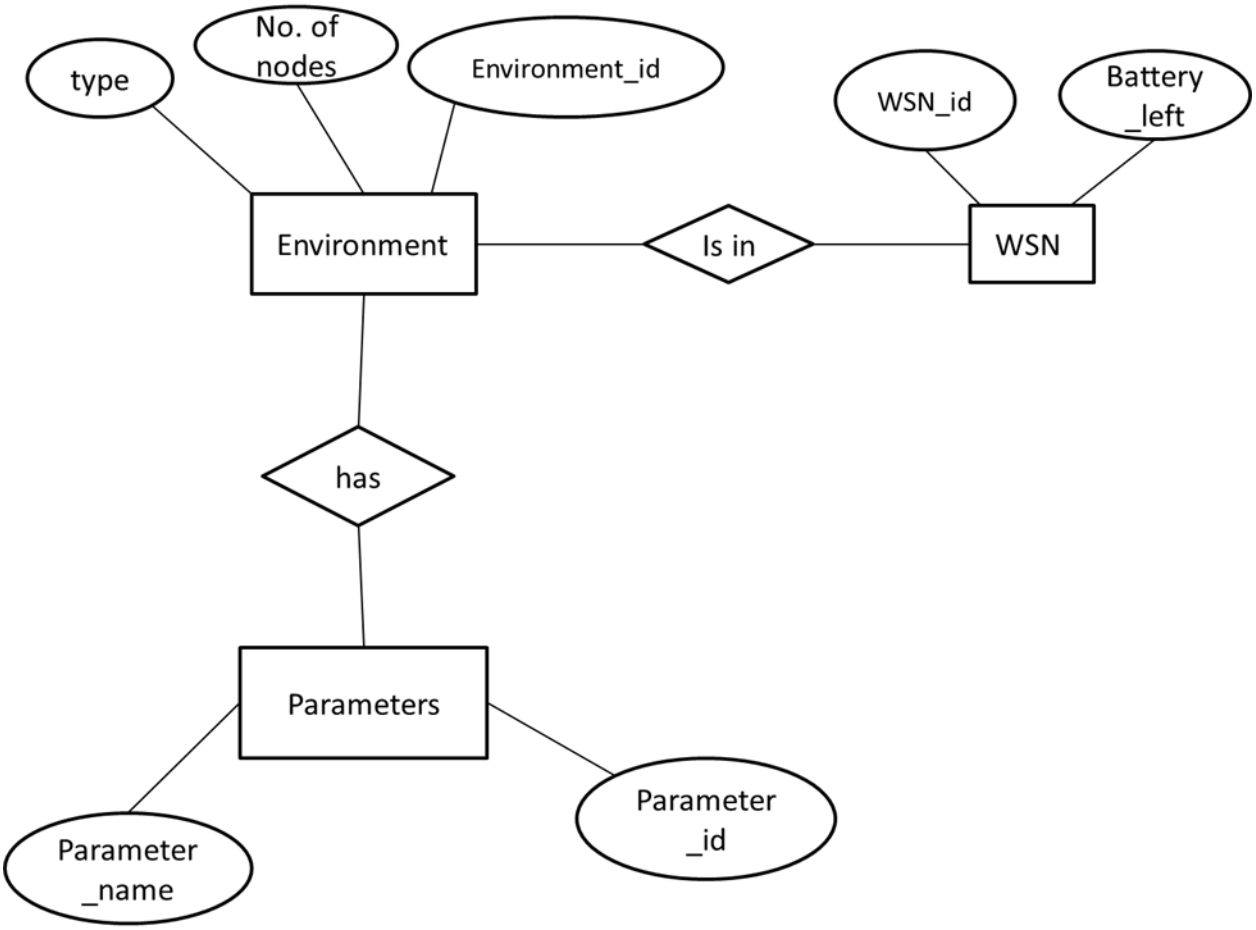


Figure 4.1 E- R Diagram

## 4.2 Activity Diagram

This diagram describes various activities performed in software under given scenario.
**Scenario:** This is the activities taking place from the point of view of each.

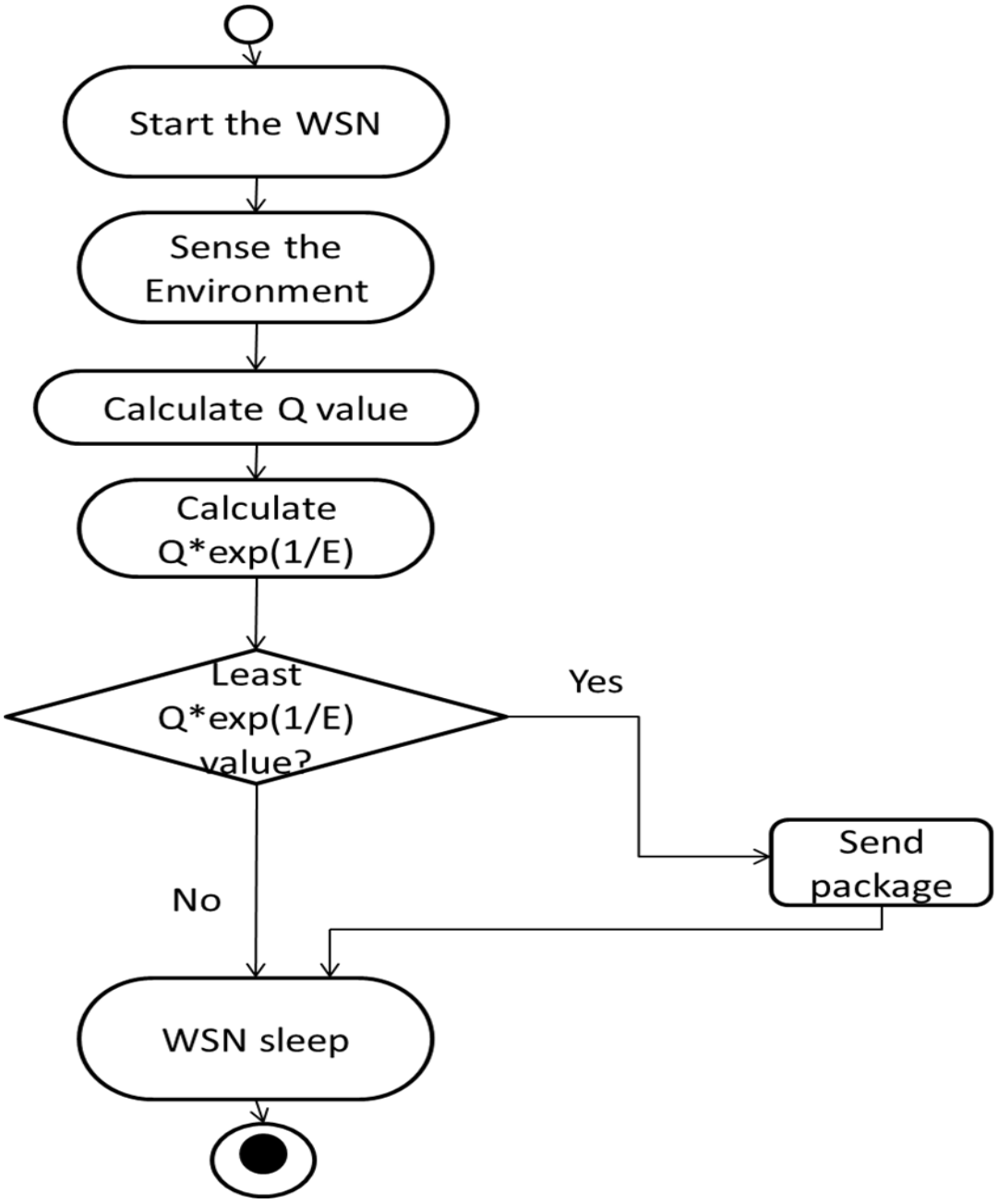


Figure 4.2 Activity Diagram

## 4.3. Functional Model

This section represents how data objects are transformed as they move through the system. Flow-oriented modeling provides a view of the system that is unique—it should be used to supplement other analysis model elements.

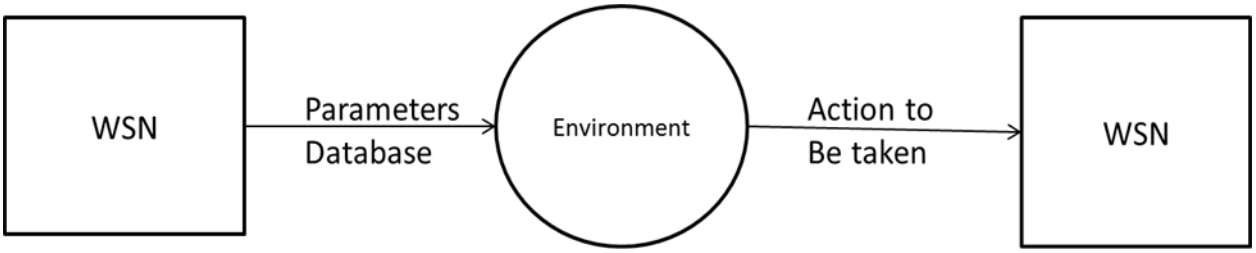


Figure 4.3.1 DFD Diagram Level 0

The WSN sends all its data from the parameter database to the environment and after processing the same, the action to be taken is sent back to the WSN along with the rewards.

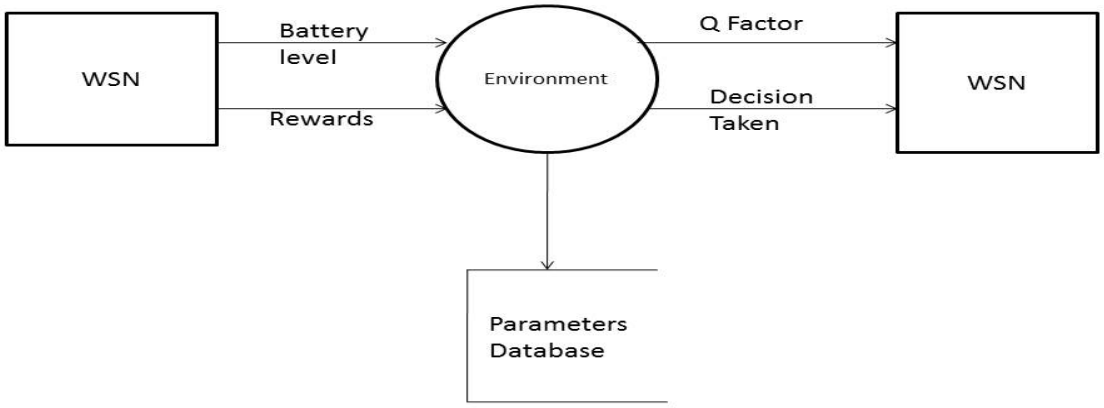


Figure 4.3.2 DFD Diagram Level 1

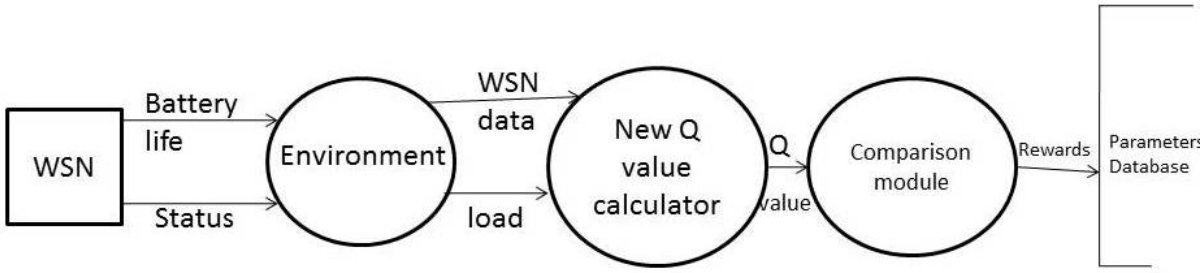


Figure 4.3.3 DFD Diagram Level 2

Each WSN first send its battery life and current status to the environment, after which the environment forwards this data to the New Q value calculator along with the load at each WSN. These Q values are then compared at the comparison module after which required action takes place and the rewards are sent to the parameters database.

## 4.4. Timeline chart

The chart below depicts the expected schedule to be followed for the project.

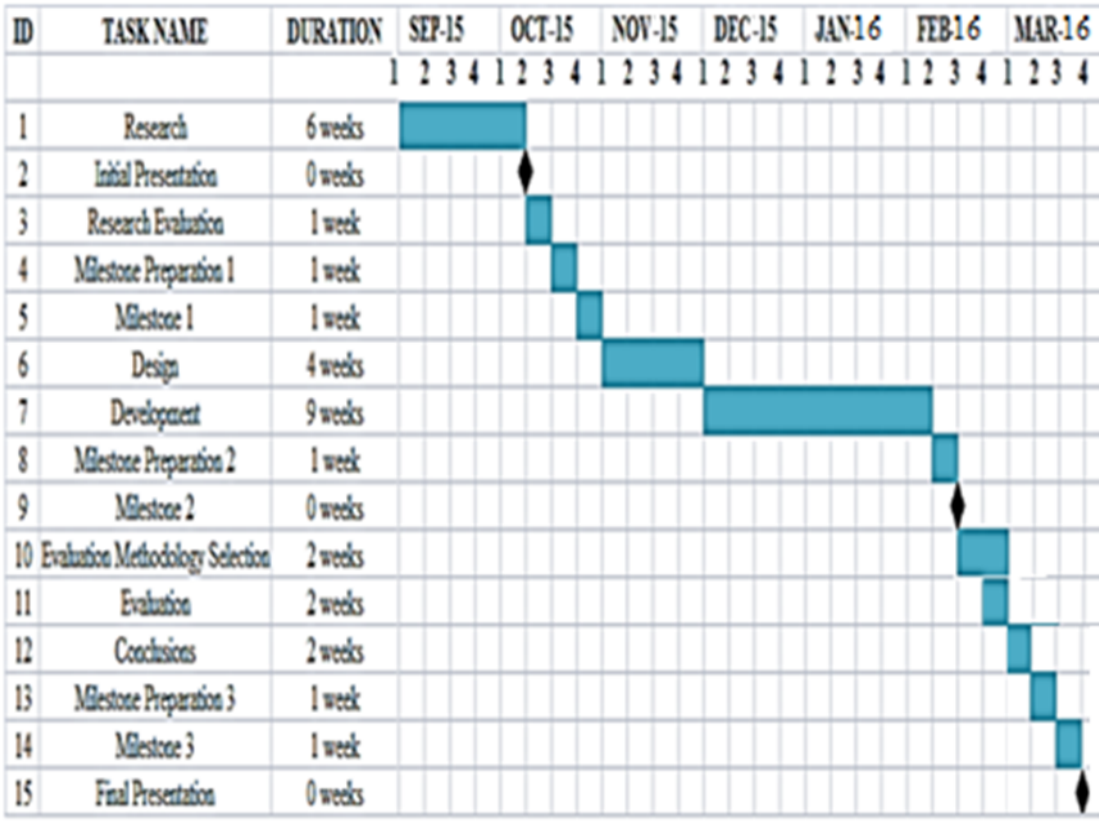


Figure 4.4 Timeline Chart

# Chapter 5

## Design

In this chapter, we illustrate the architecture diagram and the GUI screens for our proposed solution.

### 5.1. Architectural Design

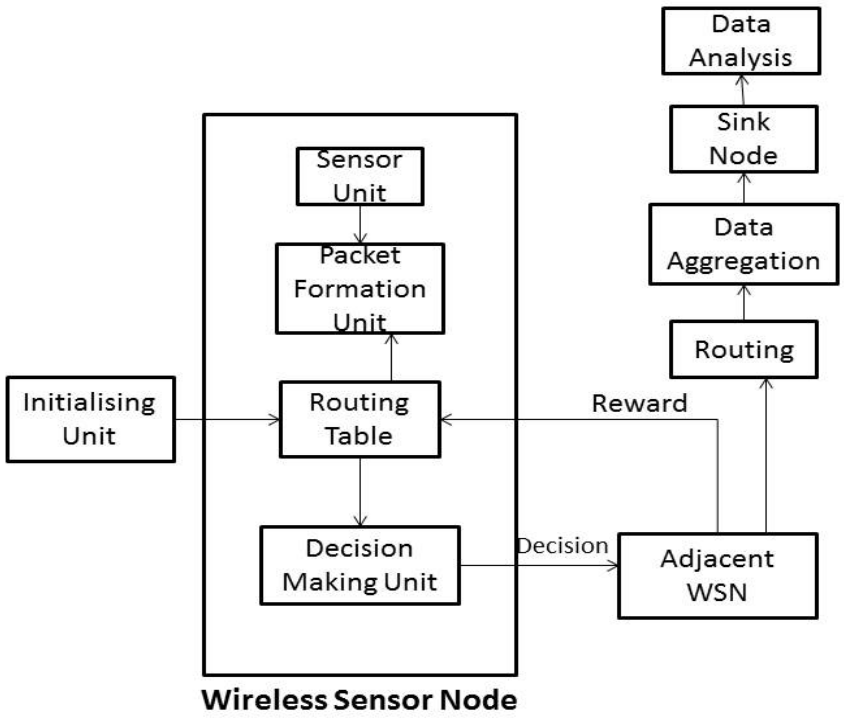


Figure 5.1 System Architecture Diagram

The System Architecture of our proposed algorithm has the following components:

- **Sensor Unit:** The sensor unit has the hardware sensors which are capable of capturing vital information like temperature, pressure, moisture etc.
- **Packet Formation Unit:** This information is then coded into packets which is set to be delivered to the sink node, which will then aggregate all the data to produce various insights from the captured information:
- **Routing Table:** The routing table is initialized using the initialization unit, which is trained with one of the algorithms like Dijktra's, Bellman Ford etc.

- **Decision Making Unit:** The Decision making unit looks up the corresponding routing table and selects the neighboring node which has the least Q – value associated with it.
- **Data Aggregation:** Data aggregation takes place at regular intervals so that large chunk of accumulated data is not sent at once.
- **Sink Node:** Sink Node is a node with infinite amount of energy and is responsible for analyzing the collected data and generating valuable insights. In a Wireless Sensor Network, all nodes sent their packets to the Sink Node.

## 5.2. User Interface Design

Nam is a Tcl/Tk based animation tool that is used to visualize the ns simulations and real world packet trace data. The first step to use nam is to produce a nam trace file. The nam trace file should contain topology information like nodes, links, queues, node connectivity etc as well as packet trace information. In this chapter we shall describe the nam trace format and simple ns commands/APIs that can be used to produce topology configurations and control animation in nam. The underlying design constraints for nam were that it is able to handle large amounts of trace data and that its animation primitives be adaptable so that it may be used in different types of network visualization. As a result, internally nam reads information from a file and keeps only a minimum amount of animation event information in memory. Its animation event has a fairly simple and consistent structure so that it can many different visualization situations.

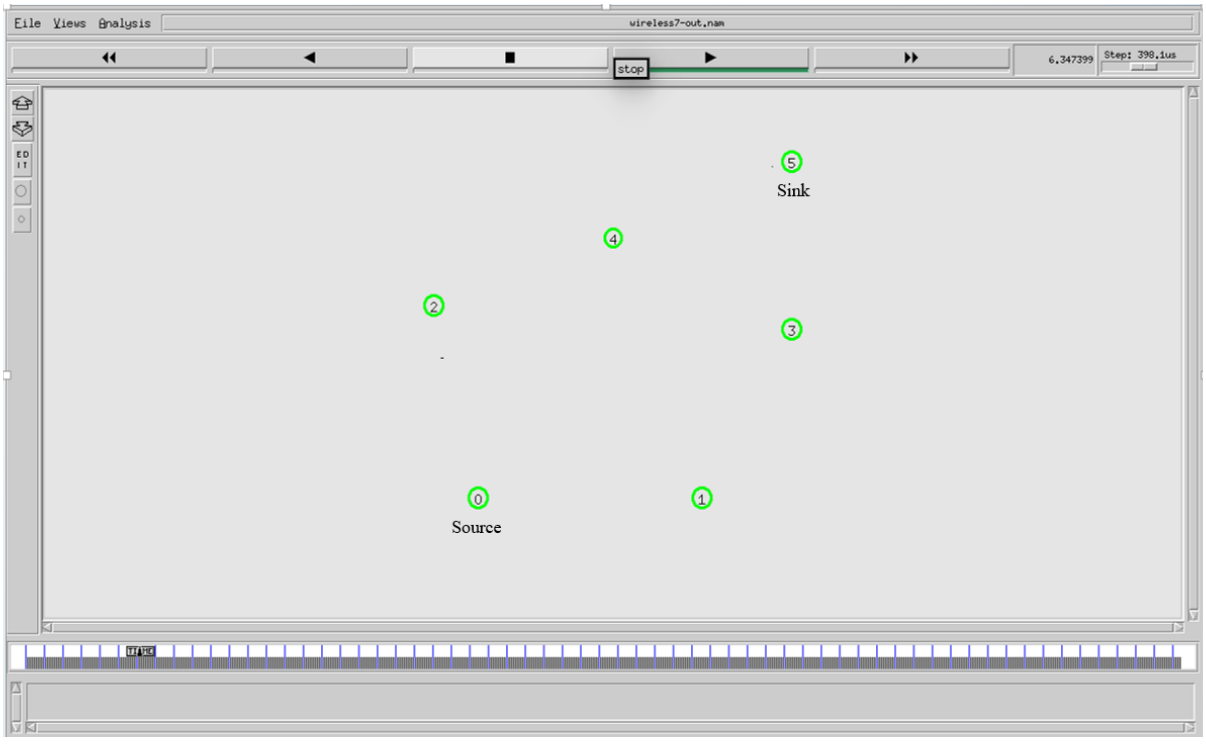


Figure 5.2.1 Initial Topology

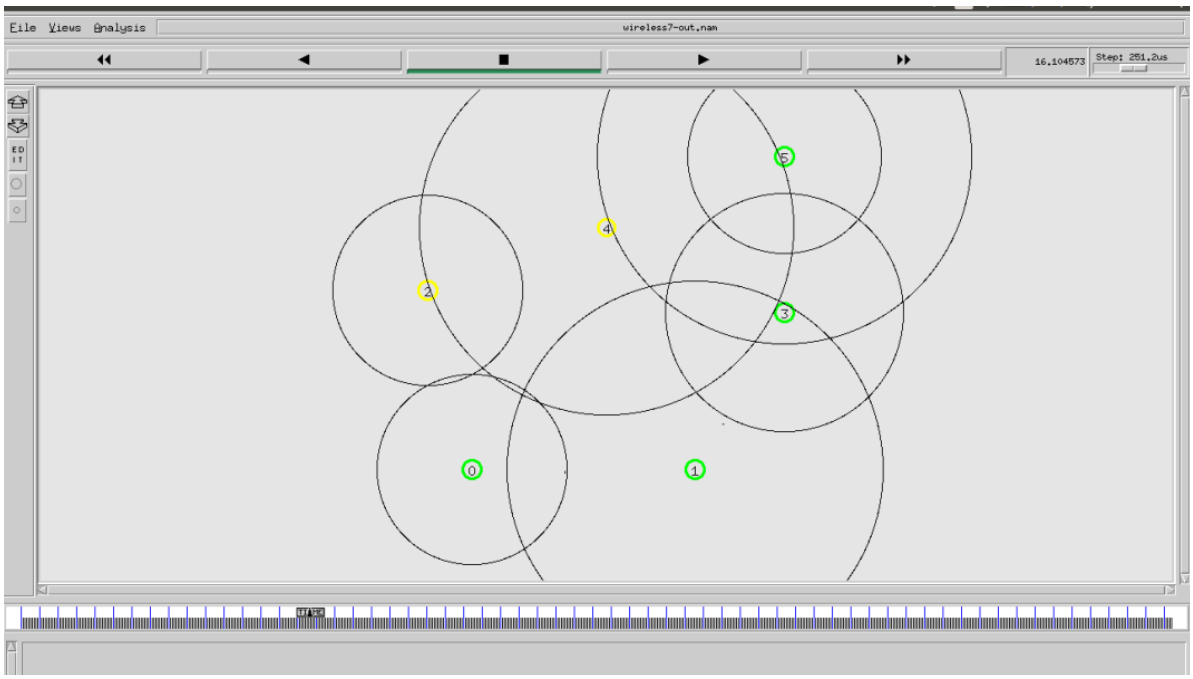


Figure 5.2.2 Intermediate Topology

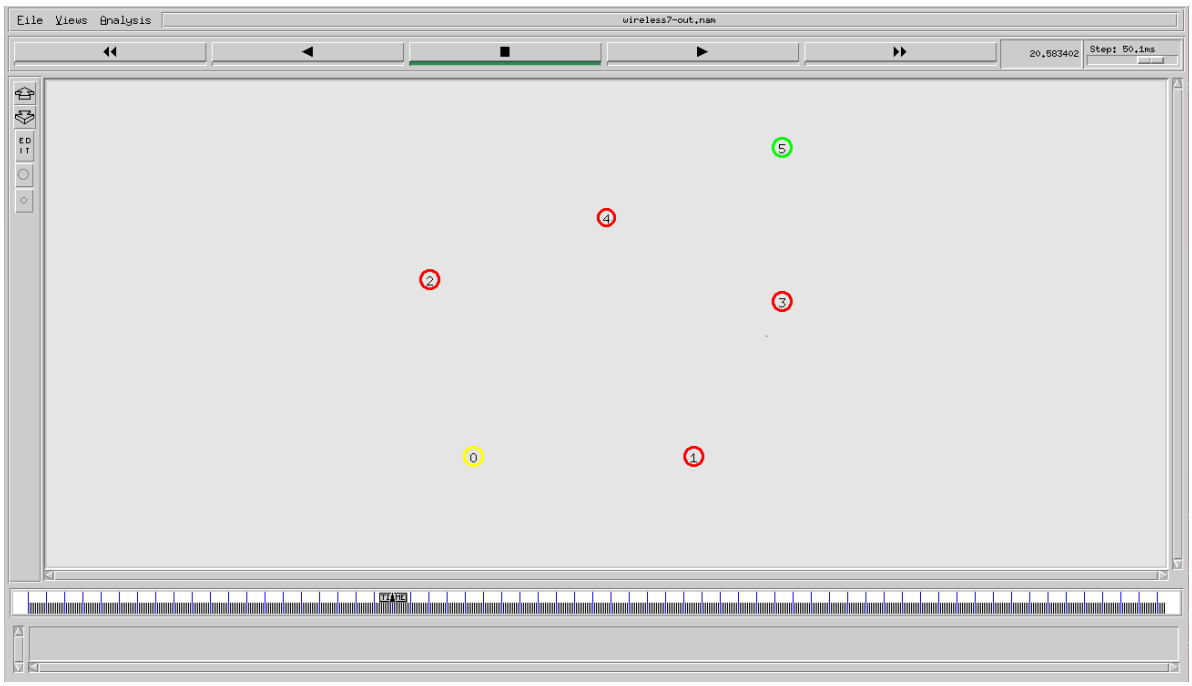


Figure 5.2.3 Final Topology

# Chapter 6

# Implementation

## 6.1. Algorithm

Our model is a fusion of both delay-based Q-routing and FROMS Q-routing, incorporating the congestion in the network into account. In the proposed algorithm, the reward packet consists of three things – the Q-value, the waiting time and the current available energy of the neighboring node. The Q-value and waiting time are same as in delay based Q-routing algorithm [12], while the energy values are adopted from the energy –efficient Q-routing algorithm [13].
All nodes store a pair of values for each neighbor. The pair of values consists of the Q-value inspired from the delay based Q-routing and the E-value denoting the energy level of the neighboring node. The product of these two values, the EQ value determines the decision of the next packet.

$$EQ(i) = Q(j) * \exp(1/E(j))$$

$$\text{Where } Q(j) = Q_{old}(j) + \alpha (t + f(q) - Q_{old}(j))$$

α is the learning rate between 0 and 1

q is the waiting time spent in the queue of neighboring node, denoting the congestion.

i is the current denote which decided to send a packet to one of its neighboring node j.

f (q) is the load multiplication function, which decides how much influence the congestion should have in the decision making process.

E (j) is the available energy of the neighboring node j.

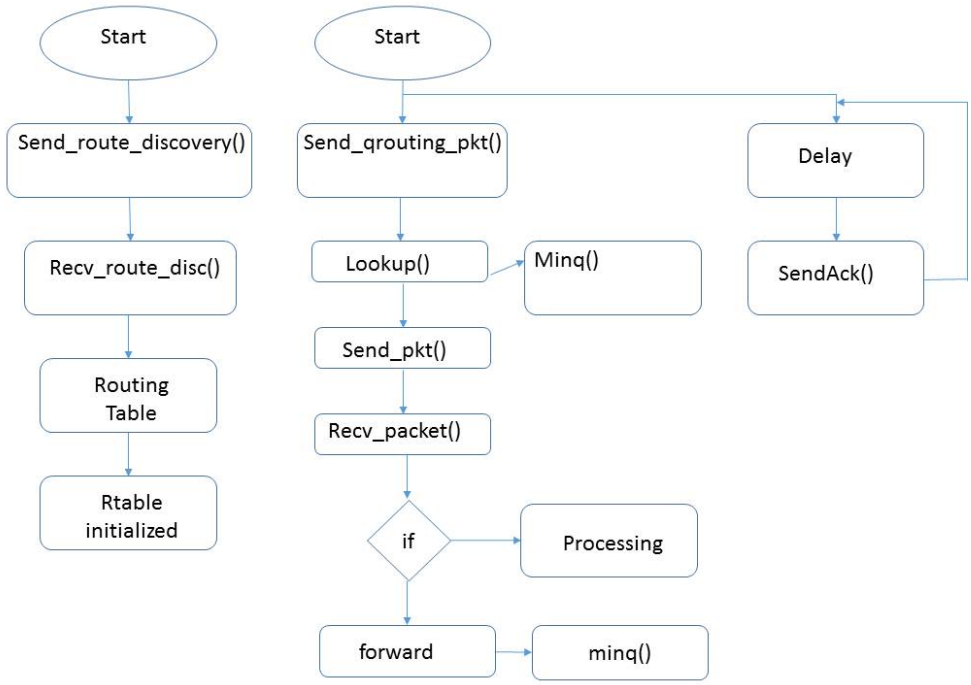


Figure 6.1 Flowchart of Q-Routing Algorithm

## 6.2 Working of the project

- **Routing Table Structure**

```
struct rtable_ent
{
public:
        nsaddr_t n;
        nsaddr_t own;
        double qenergy_;
        double qvalue_;
        double metric_;
};

class qrouting_rtable
{
        friend class rtable_ent;
public:
        qrouting_rtable();
        void add_entry(nsaddr_t,nsaddr_t,double,double,double);
        rtable_ent* lookup(nsaddr_t);
        nsaddr_t minq(nsaddr_t);
        void update_t(nsaddr_t,double);
        void print();
        rtable_ent* rtab;
        int count;
};
```

- **Send Route Discovery**

```
void Qrouting::send_route_discovery()
{

        Packet* p = allocpkt();
        struct hdr_ip* ih = HDR_IP(p);
        struct hdr_cmn* ch = HDR_CMN(p);
        struct hdr_route_discovery_pkt* rd = HDR_ROUTE_DISCOVERY_PKT(p);

        iNode = (MobileNode*)(Node::get_node_by_address(ra_addr_));
```

```
        rd -> qenergy_ = iNode -> energy_model() -> energy();
        rd -> rd_type = QROUTING_ROUTE_DISC;

        ch -> ptype() = PT_QROUTING;
        ch -> direction() = hdr_cmn::DOWN;
        ch -> size() = IP_HDR_LEN + 7;
        ch -> error() = 0;
        ch -> next_hop() = IP_BROADCAST;
        ch -> addr_type() = NS_AF_INET;

        ih -> saddr() = ra_addr_;
        ih -> daddr() = IP_BROADCAST;
        ih -> sport() = RT_PORT;
        ih -> dport() = RT_PORT;
        ih -> ttl() = IP_DEF_TTL;

        Scheduler::instance().schedule(target_,p,JITTER);
}
```

- **Receive Route Discovery Packet**

```
void Qrouting::recv_route(Packet* p)
{
        struct hdr_route_discovery_pkt* rd = HDR_ROUTE_DISCOVERY_PKT(p);
        struct hdr_ip* ih = HDR_IP(p);

        iNode = (MobileNode*)(Node::get_node_by_address(ih->saddr()));

        double x1 = iNode->X();
        double y1 = iNode->Y();

        iNode = (MobileNode*)(Node::get_node_by_address(ra_addr_));

        double x2 = iNode->X();
        double y2 = iNode->Y();

        double dist = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));

        double metric = dist/70.7;
```

```
        if(rd->qenergy_ >= 0.0001)
        {

                double qvalue = metric*exp(1.0/(rd->qenergy_));

                rtable_.add_entry (ra_addr_,ih->saddr(),rd->qenergy_,metric,qvalue);
        }
        Packet::free(p);
}
```

- **Send Q-routing Packet**

```
    void Qrouting::send_qrouting_pkt()
    {
        MobileNode* source =(MobileNode*) Node::get_node_by_address(0);
        nsaddr_t sou = source -> nodeid();

        if(sou == ra_addr_)
        {

            Packet* p = allocpkt();
            struct hdr_qrouting_pkt *qt = HDR_QROUTING_PKT(p);
            struct hdr_cmn *ch = HDR_CMN(p);
            struct hdr_ip* ih = HDR_IP(p);

            qt -> q_type = QROUTING_PKT;

            nsaddr_t dest = rtable_.minq(sou);

            ch -> ptype() = PT_QROUTING;
            ch -> direction() = hdr_cmn::DOWN;
            ch -> size() = IP_HDR_LEN + 7;
            ch -> error() = 0;
            ch -> next_hop() = dest;
            ch -> addr_type() = NS_AF_INET;

            MobileNode* sink =(MobileNode*) Node::get_node_by_address(5);
            nsaddr_t s = sink -> nodeid();

            ih -> saddr() = sou;
            ih -> daddr() = s;
```

```
                ih -> sport() = RT_PORT;
                ih -> dport() = RT_PORT;
                ih -> ttl() = IP_DEF_TTL;

                Scheduler::instance().schedule(target_,p,JITTER);
        }
}
```

- **Send Q-routing ACK**

```
    void Qrouting::send_qrouting_ack()
    {
        Packet* p = allocpkt();
        struct hdr_qrouting_pkt_ack *qa = HDR_QROUTING_PKT_ACK(p);
        struct hdr_cmn *ch = HDR_CMN(p);
        struct hdr_ip *ih = HDR_IP(p);

        iNode = (MobileNode*)(Node::get_node_by_address(ra_addr_));
        qa -> qa_type = QROUTING_ACK;
        qa -> qa_ = iNode->energy_model()->energy();

        ch -> ptype() = PT_QROUTING;
        ch -> direction() = hdr_cmn::DOWN;
        ch -> size() = IP_HDR_LEN + 7;
        ch -> error() = 0;
        ch -> next_hop() = IP_BROADCAST;
        ch -> addr_type() = NS_AF_INET;

        ih -> saddr() = ra_addr_;
        ih -> daddr() = IP_BROADCAST;
        ih -> sport() = RT_PORT;
        ih -> dport() = RT_PORT;
        ih -> ttl() = IP_DEF_TTL;

        Scheduler::instance().schedule(target_,p,JITTER);
}
```

- **Receive Q-routing Packet**

```
    void  Qrouting::recv_qrouting_pkt(Packet* p)
        {
```

```
        struct hdr_qrouting_pkt *qt = HDR_QROUTING_PKT(p);
        struct hdr_ip* ih = HDR_IP(p);

        MobileNode* sink =(MobileNode*) Node::get_node_by_address(5);

        nsaddr_t s = sink -> nodeid();

        if(ra_addr_ == s)
        {
                printf("Processing...");
                Packet::free(p);
        }
        else
        {
                send_qrouting_ack(ih->saddr());
                forward_data(p);
        }

}
```

- **Receive Q-routing ACK**

```
void Qrouting::recv_qrouting_ack(Packet* p)
{
        struct hdr_qrouting_pkt_ack *qa = HDR_QROUTING_PKT_ACK(p);
        struct hdr_ip* ih = HDR_IP(p);

        rtable_ent* rt = rtable_.lookup(ih->saddr());

        double en = qa -> qa_;

        if(en >= 0.0001)
                rtable_.update_t(rt->n,en);

}
```

# Chapter 7

# TESTING

## 7.1 Test cases

### 7.1.1 White-box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing, by seeing the source code) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).
While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

### 7.1.2 Black-box testing

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing. Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

### 7.1.3 Unit Testing

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control

data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

## 7.2 Type of Testing used

We use Unit Testing for our code. We test each module of our C++ code and then test the entire program after aggregation of all the modules. We perform Unit Testing in two phases

- Route Discovery Phase
- ACK Check Phase

- **Route Discovery Phase:**

The Route Discovery Phase involves the identification of neighbors and initialization of the Routing Tables for each node. Each node will broadcast a simple packet to all its neighbors. The neighbor will send an ACK with its node_id, distance and energy value. Each node will receive ACK packets and initialize their respective routing tables. We perform unit testing to check whether the routing tables are being initialized accurately.

```
S_Node = 0      Neig_Node = 2   Energy=10.000000        Metric=2.915916 Qvalue=3
.222586
S_Node = 0      Neig_Node = 1   Energy=10.000000        Metric=3.536068 Qvalue=3
.907959
S_Node = 1      Neig_Node = 0   Energy=15.000000        Metric=3.536068 Qvalue=3
.779841
S_Node = 1      Neig_Node = 3   Energy=10.000000        Metric=2.850869 Qvalue=3
.150698
S_Node = 2      Neig_Node = 0   Energy=15.000000        Metric=2.915916 Qvalue=3
.116937
S_Node = 2      Neig_Node = 4   Energy=10.000000        Metric=2.997117 Qvalue=3
.312327
S_Node = 3      Neig_Node = 4   Energy=10.000000        Metric=3.131767 Qvalue=3
.461137
S_Node = 3      Neig_Node = 1   Energy=10.000000        Metric=2.850869 Qvalue=3
.150698
S_Node = 3      Neig_Node = 5   Energy=1000000.000000   Metric=2.475248 Qvalue=2
.475250
S_Node = 4      Neig_Node = 2   Energy=10.000000        Metric=2.997117 Qvalue=3
.312327
S_Node = 4      Neig_Node = 3   Energy=10.000000        Metric=3.131767 Qvalue=3
.461137
S_Node = 4      Neig_Node = 5   Energy=1000000.000000   Metric=3.046769 Qvalue=3
.046772
S_Node = 5      Neig_Node = 3   Energy=10.000000        Metric=2.475248 Qvalue=2
.735572
S_Node = 5      Neig_Node = 4   Energy=10.000000        Metric=3.046769 Qvalue=3
```

Figure 7.2.1 Testing Routing Route Discovery

- **ACK Check Phase:**

In this phase we check whether the routing tables are being updated correctly. For this, we unit test the function send_qrouting_ack() and recv_qrouting_ack().
We display the routing table at an intermediate timestep to check whether the information is being updated in the Routing Table.

```
S_Node = 0      Neig_Node = 2   Energy=4.599946 Metric=2.915916 Qvalue=4.777247
S_Node = 0      Neig_Node = 1   Energy=6.623941 Metric=3.536068 Qvalue=4.900652
S_Node = 1      Neig_Node = 0   Energy=12.294911        Metric=3.536068 Qvalue=4
.039083
S_Node = 1      Neig_Node = 3   Energy=6.596000 Metric=2.850869 Qvalue=3.875491
S_Node = 2      Neig_Node = 0   Energy=12.294911        Metric=2.915916 Qvalue=3
.330713
S_Node = 2      Neig_Node = 4   Energy=4.996588 Metric=2.997117 Qvalue=4.700254
S_Node = 3      Neig_Node = 4   Energy=7.146152 Metric=3.131767 Qvalue=3.897381
S_Node = 3      Neig_Node = 1   Energy=7.806227 Metric=2.850869 Qvalue=3.505880
S_Node = 3      Neig_Node = 5   Energy=999997.657140    Metric=2.475248 Qvalue=2
.475252
S_Node = 4      Neig_Node = 2   Energy=6.925757 Metric=2.997117 Qvalue=3.746605
S_Node = 4      Neig_Node = 3   Energy=7.806227 Metric=3.131767 Qvalue=3.851316
S_Node = 4      Neig_Node = 5   Energy=999997.657140    Metric=3.046769 Qvalue=3
.046775
S_Node = 5      Neig_Node = 3   Energy=7.806227 Metric=2.475248 Qvalue=3.043956
S_Node = 5      Neig_Node = 4   Energy=7.146152 Metric=3.046769 Qvalue=3.791605
```

Figure 7.2.2 Routing Table after 10 NS2 time steps

# Chapter 8

## Results and Discussions

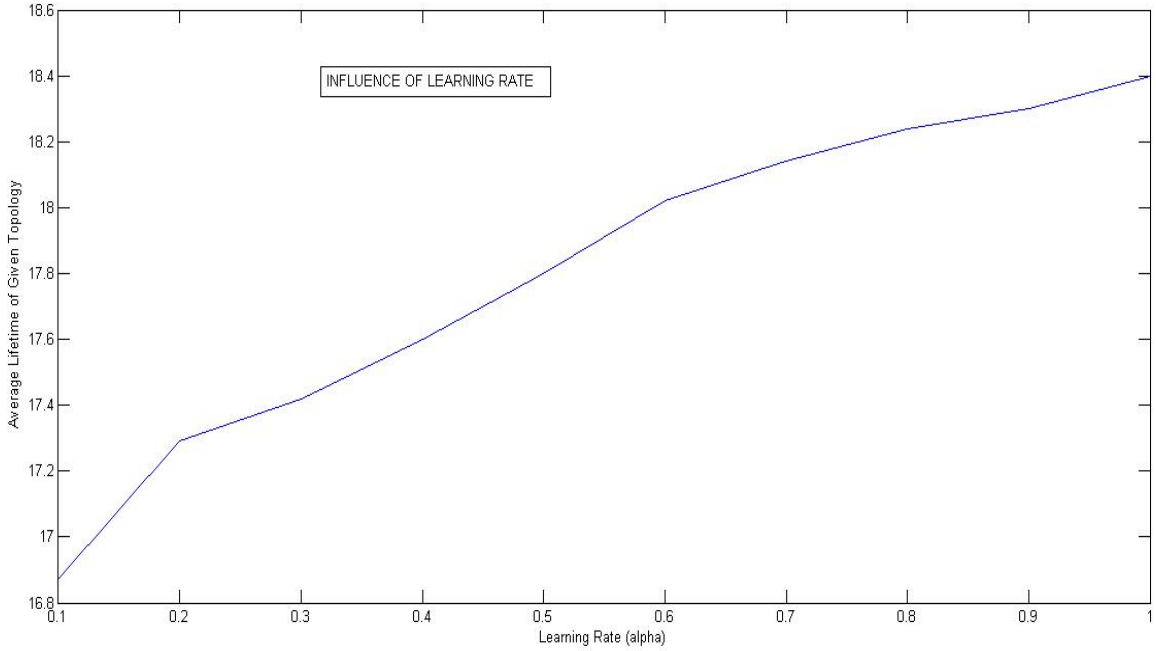


Figure 8.1 Graph of Lifetime of Topology against Learning Rate

Table 9.2 Learning rate versus average lifetime

| α | Average Lifetime |
|---|---|
| 0.1 | 16.87 |
| 0.2 | 17.79 |
| 0.3 | 17.42 |
| 0.4 | 17.60 |
| 0.5 | 17.80 |
| 0.6 | 18.02 |
| 0.7 | 18.14 |
| 0.8 | 18.24 |
| 0.9 | 18.30 |
| 1.0 | 18.40 |

The above graph depicts the influence of the learning rate (alpha) on the average lifetime of the network. Learning rate is the amount of influence the new information has on the old information. As we can see in graph 8.1, the lifetime of the network increases steadily when 0.1 < alpha < 0.6 and then remains almost constant. Hence, the ideal learning rate would be when alpha = 0.6

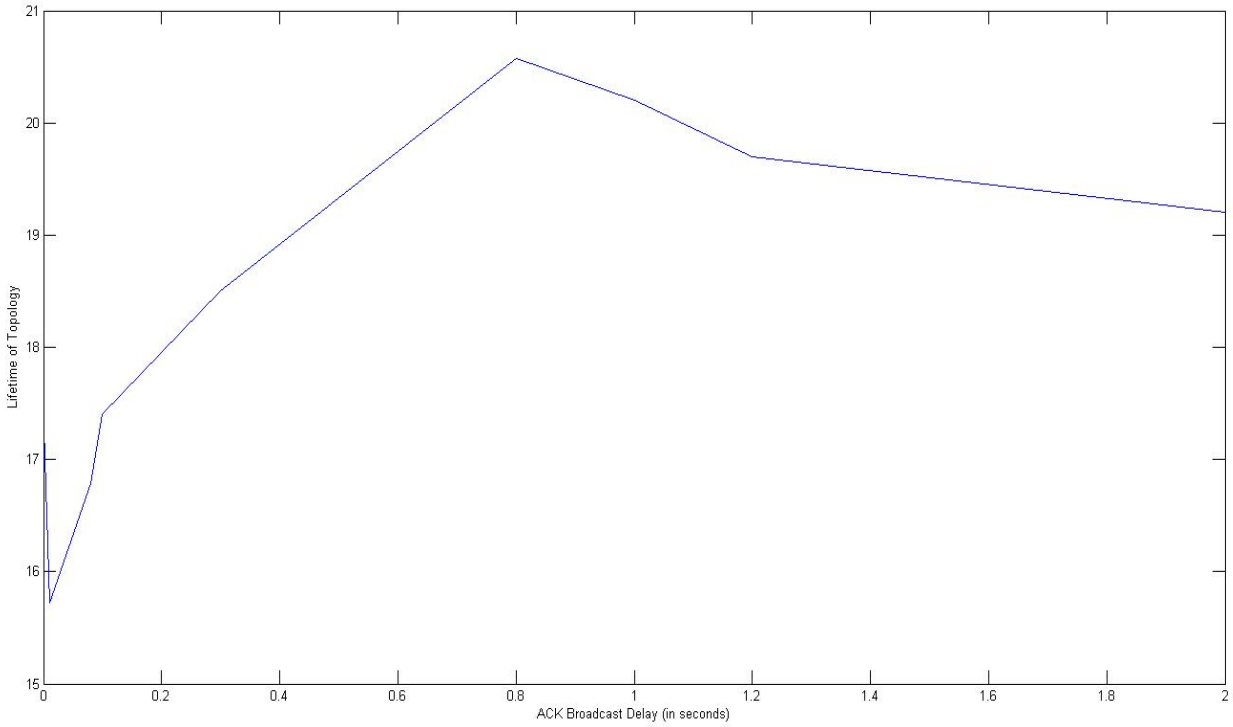


Figure 8.2 Graph of Lifetime of Topology against Delay for an Acknowledge Packet

Table 9.2 Delay for an Acknowledge Packet versus average lifetime

| Delay for an Acknowledge Packet | Average Lifetime |
|---|---|
| 0.001 | 17.27 |
| 0.01 | 15.72 |
| 0.08 | 16.79 |
| 0.1 | 17.40 |
| 0.3 | 18.85 |
| 0.8 | 20.58 |
| 1.0 | 20.20 |
| 1.2 | 19.70 |
| 2.0 | 19.20 |

The above graph depicts the influence of delay enforced on the sending of broadcast packets to the neighboring nodes, so that they can update their information. A very frequent IP Broadcast would consume sensor node's battery life, while on the other hand, a delayed IP Broadcast would mean that the node is using older information while making routing decisions. Hence, a balance between the two is very necessary.

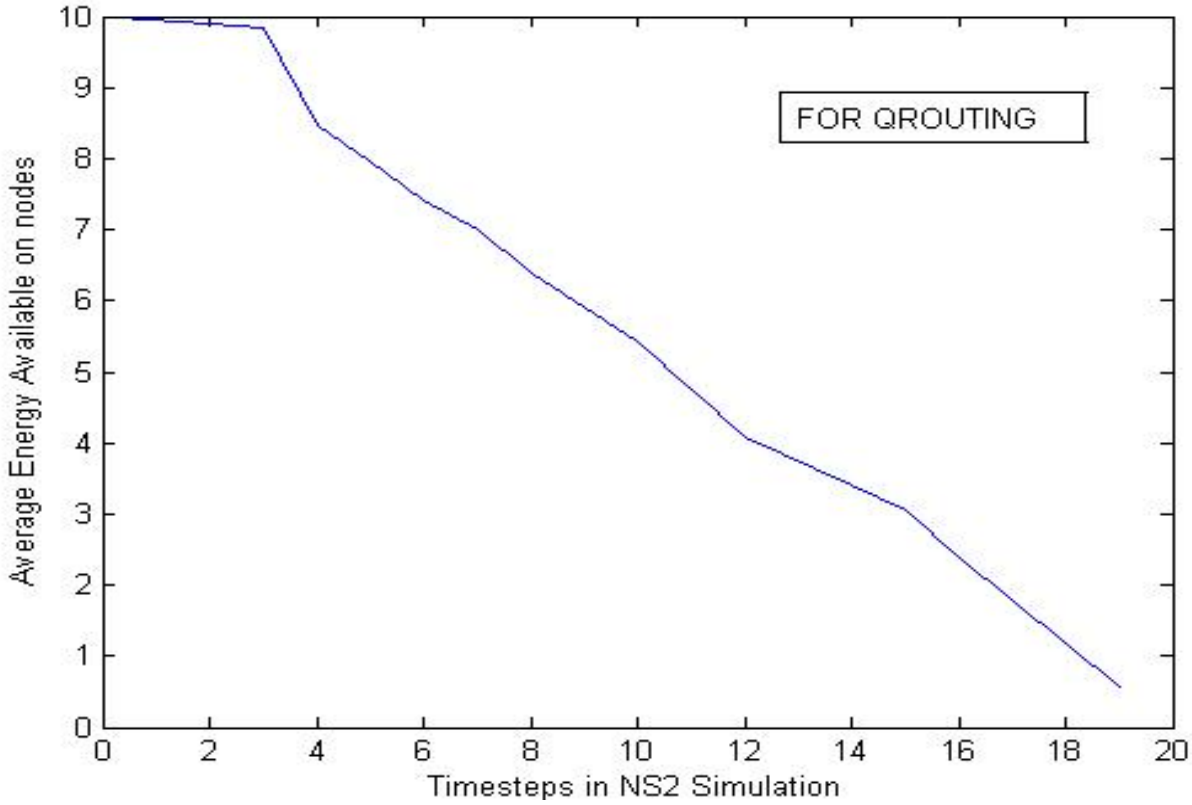


Figure 8.3 Graph of Lifetime of Topology against Energy for Q-Routing

Table 9.3 Energy versus average lifetime for Q-Routing

| Lifetime | Energy |
|---|---|
| 0 | 10.000 |
| 2 | 9.893 |
| 4 | 8.478 |
| 6 | 7.415 |
| 8 | 6.400 |
| 10 | 5.418 |
| 12 | 4.088 |
| 14 | 3.410 |
| 16 | 2.385 |
| 18 | 0.789 |
| 19 | 0.556 |

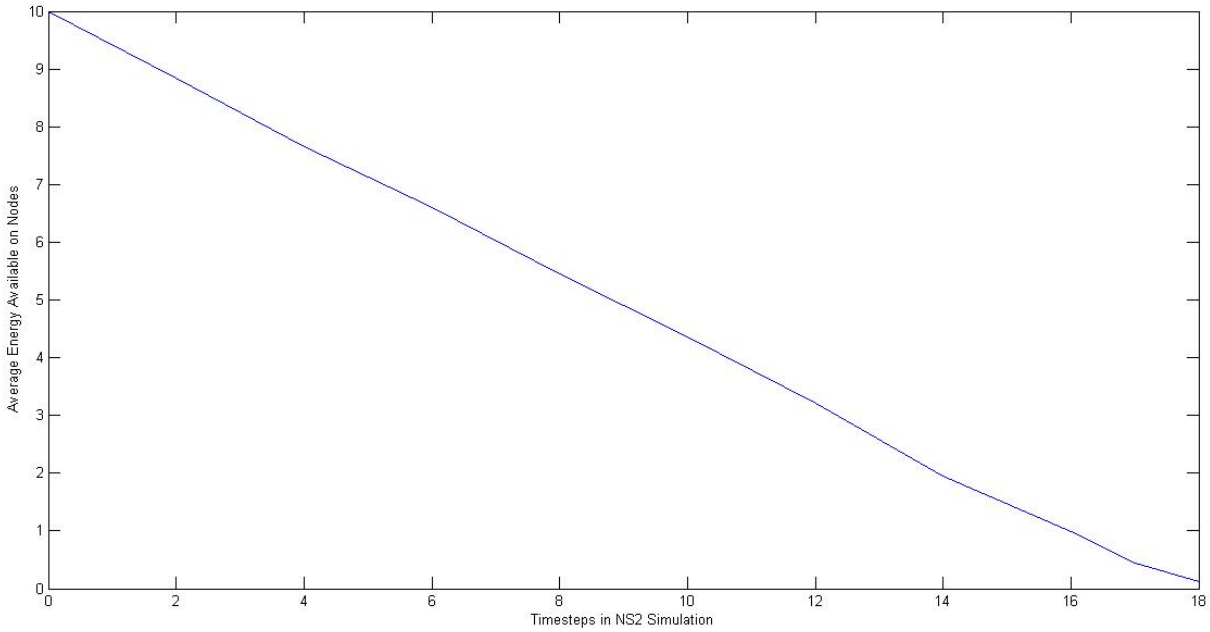


Figure 8.4 Graph of Lifetime of Topology against Energy for AODV

Table 9.4 Energy versus average lifetime for AODV

| Lifetime | Energy |
|---|---|
| 0 | 10.000 |
| 2 | 8.845 |
| 4 | 7.650 |
| 6 | 6.598 |
| 8 | 5.458 |
| 10 | 4.365 |
| 12 | 3.228 |
| 14 | 1.950 |
| 16 | 0.997 |
| 18 | 0.115 |

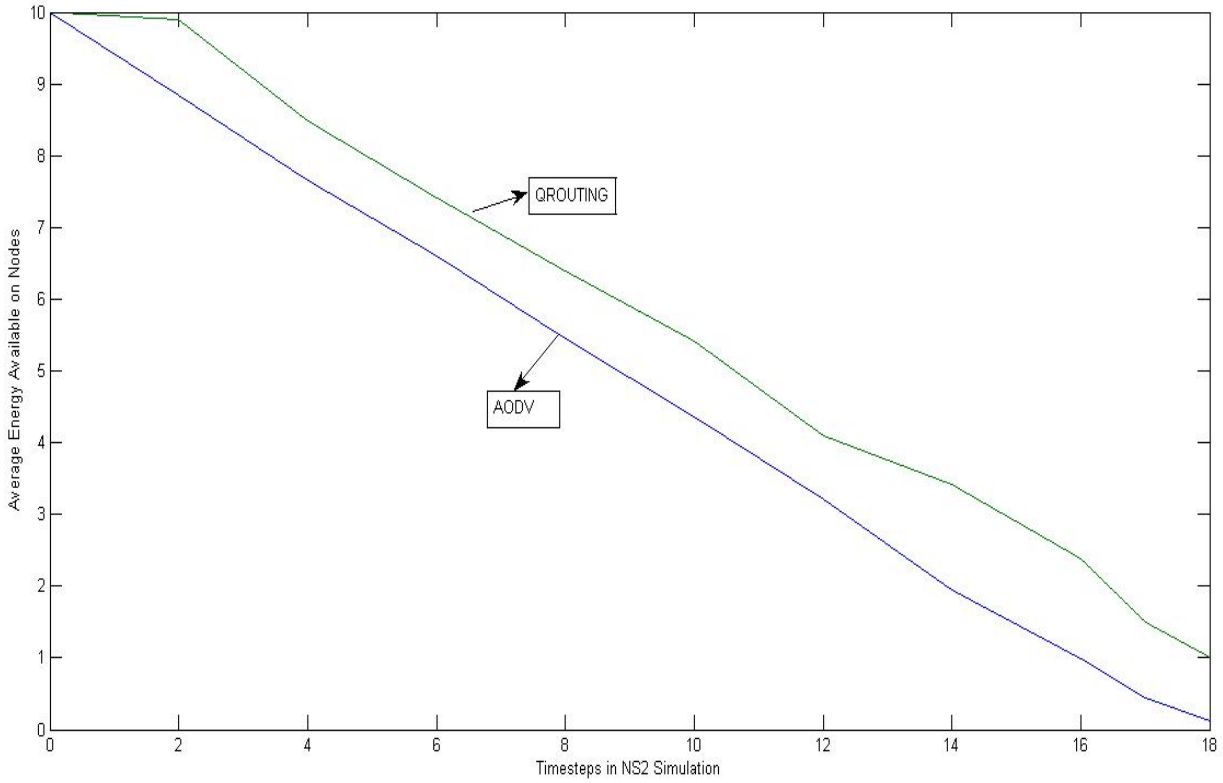


Figure 8.5 Comparison Graph for Lifetime of Topology

The above graph depicts the comparison of AODV protocol and our Q-Routing protocol. As we can see, for the given topology, the average lifetime of the network when AODV routing algorithm in used is 18.345 time steps. On the other hand, when our Q-Routing algorithm is implemented on the same topology, the lifetime of the network increases to 19.876 time steps. This increase in lifetime is achieved due to balancing of load equally to all available neighboring nodes rather than passing packets through the same route. This constantly switching of routes helps the sensor networks to preserve energy when inactive, in turn, increasing the lifetime of the entire network.

# Chapter 9

## Conclusion and Future Scope

In conclusion, we discern that the average lifetime of the network increases considerably when an adaptive algorithm is used. Considering the load of the neighboring nodes before sending packets helps us to reduce network latency. The packet might travel an extra hop or more distance rather than waiting in the queue.

The future work includes finding a function which incorporates global load of the network into consideration. In majority the cases load is excessive on node which are closer to the sink than others. Having access to the information about congestion at these nodes will help the source node to make better routing decisions and thus, improve the average latency of the network. We intend to test our proposed algorithm on various homogenous and heterogeneous networks. We will specially focus on bottleneck cases where load multiplication factor plays a crucial role.

This project can also be extended by finding ways in which Energy can be transferred from one node to another through some medium, for instance, Bluetooth. Energy harvesting techniques like obtaining energy through solar, vibration, wind etc.

# References

[1] T. He, J. Stankovic, C. Lu and T. Abdelzaher, “A Spatiotemporal Communication Protocol for Wireless Sensor Networks”, IEEE Transactions on Parallel and Distributed Systems, 2005.

[2] Tiwari A. Ballal, Lewis F.L, “Energy-efficient wireless sesnor network design and implementation for condition based maintanenance”, ACM Trans. Sen. Netw. 3, 2008.

[3] Mehaseb Ahmed Mehaseb, “WSN Application Traffic Characterization for Integration within the Internet of Things,” IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks, 2013.

[4] The VINT Project. The ns Manual, December 2003 http://www.isi.edu/nsnam/ns/ns-documentation.html.

[5] Farzad Kiani, Ehsan Amiri,Mazdak Zamani, Touraj Khodadadi, Azizah Abdul Manaf, “Efficient Intelligent Energy Routing Protocol in Wireless Sensor Networks”, International Journal of Distributed Sensor Networks, Volume 2015, Article ID 618072.

[6] C. H. B. Wendi Rabiner Heinzelman, "Energy-efficient communication protocol for wireless micro sensor networks," Proc. 33rd Annual Hawaii International Conference on System Sciences, 2000.

[7] Xuxun Liu,"A Survey on Clustering Routing Protocols in Wireless Sensor Networks",sensors ISSN 1424-8220,August 2012.

[8] Stephanie Lindsey, Cauligi S. Raghavendra, “PEGASIS: Power-Efficient GAthering in Sensor Information Systems,” Aerospace Conferance Proceedings, 2002.

[9] Anna Forster, “Machine Learning Techniques Applied to Wireless Ad-hoc Networks: Guide and Survey”, University of Lugano, Switzerland.

[10] Boyan, J. A. and Littman, M. L. “Packet routing in dynamically changing networks: A reinforcement learning approach. In Advances in Neural,” Advances in Neural Information Processing Systems 6, pages 671-678, 1994.

[11] Forster, A. and Murphy, A. L., “Balancing energy expenditure in WSN through reinforcement learning: A study,” 1st International Workshop on Energy in Wireless Sensor Networks WEWSN, page 7, 2008.

# Acknowledgements

We would like to express our sincere gratitude to our guide Prof. Aruna Gawade for her guidance, encouragement and gracious support throughout the course of our work, for her expertise in the field that motivated us to work in this area and for her faith in us in every stage of this research.

We're grateful to Dr. Narendra Shekokar, Head of the Computer Engineering Department, for letting us use the department resources, labs and books.

We're highly indebted to our Principal, Dr. Hari Vasudevan for availing us with library books and relevant materials and also the SVKM management for providing us with popular and resourceful journals like IEEE and other online material that helped us finish our project with ease and perfection.

We would also like to thank all our fellow students and staff of Department of Computer Engineering for their help in the whole process leading to the conceptualization of the project.

Devang Jhaveri

Miti Jhaveri

Jehan Kandawalla