Resource Aware Task Scheduling in Wireless Sensor Networks

By

Maksura Mahjabeen

ID: 13101297

Under the Supervision of

Dr. Md .Muhidul Islam Khan

Inspiring Excellence

Department of Computer Science & Engineering

School of Engineering & Computer Science

BRAC University

Resource Aware Task Scheduling in Wireless Sensor Networks

Bachelor of Science

In

Computer Science and Engineering

Under the Supervision of

Dr.Md.Muhidul Islam Khan

By

Maksura Mahjabeen

ID: 13101297

April, 2016

DECLARATION

We do hereby declare that the thesis titled " Machine Learning Based Resource Aware Task Scheduling in Wireless Sensor Networks" is submitted to the Department of Computer Science and Engineering of BRAC University in partial fulfillment of the completion of Bachelors of Science in Computer Science and Engineering. We hereby declare that this thesis is based on results obtained from our own work. Due acknowledgement has been made in the text to all other material used. This thesis, neither in whole nor in part has been previously submitted to any University or Institute for the award of any degree or diploma. The materials of work found by other researchers and sources are properly acknowledged and mentioned by reference.

Dated: 20th April, 2016

Signature of Supervisor

Signature of Authors

_____

Dr. Md. Muhidul Islam Khan

Assistant professor

Department of Computer Science and Engineering

BRAC University

Dhaka,Bangladesh

_____

Maksura Mahjabeen

ID: 13101297

FINAL READING APPROVAL

Thesis Title:

Machine Learning Based Resource Aware Task Scheduling in Wireless Sensor Networks

Date of Submission: 20th April, 2016

The final report is satisfactory and it's all materials are also acceptable and ready for the submission to the Department of Computer Science and Engineering,BRAC University.

Signature of Supervisor

Dr.Md.Muhidul Islam Khan

Assistant professor

Department of Computer Science and Engineering

BRAC University

PREFACE

I am very thankful to my thesis coordinator Dr.Md.Muhidul Islam Khan , Assistant Professor, Department of Computer Science and Engineering, BRAC University for guiding me throughout my thesis work.

.

Date:20th December 2015

Maksura Mahjabeen

ABSTRACT

Wireless Sensor Networks (WSNs) consist of so many sensor nodes capable of sensing, processing and transmitting sensed information to a remote station. Among the application of wireless sensor networks target tracking is considered as the most significant and pre-eminent application. While tracking multiple objects, it is very important to schedule the task in an efficient manner such that we can get optimal result by maintaining energy constraint. Here, we develop a method that would be able to find out the best possible result in accordance with the prominent trade-off between these two factors. Furthermore, tracking multiple objects is more formidable than single target tracking as the speed, position and movement of targets can be different. Also, there are issues of connectivity failure and high power consumption that could lead us to data loss in a system. In our paper, we consider the sensor nodes as intelligent agents that will adapt the next task by observed application behavior by using cooperative reinforcement learning where we introduce a reward function based on our specific condition that includes energy efficiency and performance. Simulation results show that our proposed methods provide better trade-off between power consumption and performance comparing with the existing methods

Keywords: Wireless Sensor Networks, task scheduling, reinforcement learning, trade off, resource aware, reward function.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter One

Introduction

In research area wireless sensor network has occupied tremendous interest recently. There are logical reasons behind this like advancement in wireless communication and micro-electromechanical systems (MEMSs) have enabled the development of low-cost, low power, multi-functional, tiny sensor nodes that can sense the environment, perform data processing, and communicate with each other over short distances [3]. A typical wireless sensor network consists of thousands of sensor nodes, deployed either randomly or according to some predefined statistical distribution, over a geographic region of interest. In a wireless sensor network (WSN), the usages of resources are usually highly related to the execution of tasks which consume a certain amount of computing and communication bandwidth. Parallel processing among sensors is a promising solution to provide the demanded computation capacity in WSNs, and task allocation and scheduling play an essential role in parallel processing. Therefore, how to assign a task to its most appropriate sensor node and simultaneously balance the network load in the context of the uncertain and dynamic network environments represents an important and urgent issue in WSN studies.

1.1 Scope and Challenges

A sensor node by itself has severe resource constraint, such as low battery power, limited signal processing, limited computation and communication capabilities, and a small amount of memory; hence it can sense only a limited portion of the environment. However, when a group of sensor nodes collaborate with each other, they can accomplish a much bigger task efficiently. One of the primary advantages of deploying a wireless sensor network is its low deployment cost and freedom from requiring a messy wired communication backbone, which is often infeasible or economically inconvenient.

Wireless sensor networks ensure a wide range of applications, starting from security surveillance in military and battlefields, monitoring previously unobserved environmental phenomena, smart homes and offices, improved healthcare, industrial diagnosis, and many more. For instance, a sensor network can be deployed in a remote island for monitoring wildlife habitat and animal behavior or near the crater of a volcano to measure temperature, pressure, and seismic activities.

In many of these applications the environment can be hostile where human intervention is not possible and hence, the sensor nodes will be deployed randomly or sprinkled from air and will remain unattended for months or years without any battery replacement. Therefore, energy consumption or, in general, resource management is of critical importance to these networks. With the limited resource, our goal is to get optimum performance that is assigned to the sensor nodes by using minimum energy. In this paper, we propose a co-operative reinforcement learning (RL) method for task scheduling. The proposed a method helps to learn the best task scheduling strategy based on the previously observed behavior and is further able to adapt the changes in environment. A key step here is to exploit cooperation among neighboring nodes, i.e., the exchange of information about the current local view on the application state [10]. Such cooperation helps to improve the trade-off between energy consumption and performance. In our simulation we compare our cooperative with non-cooperative methods in terms of energy efficiency and application quality. We observe the energy/performance trade-off considering different balancing factors of the reward function, different network sizes and different target mobility.

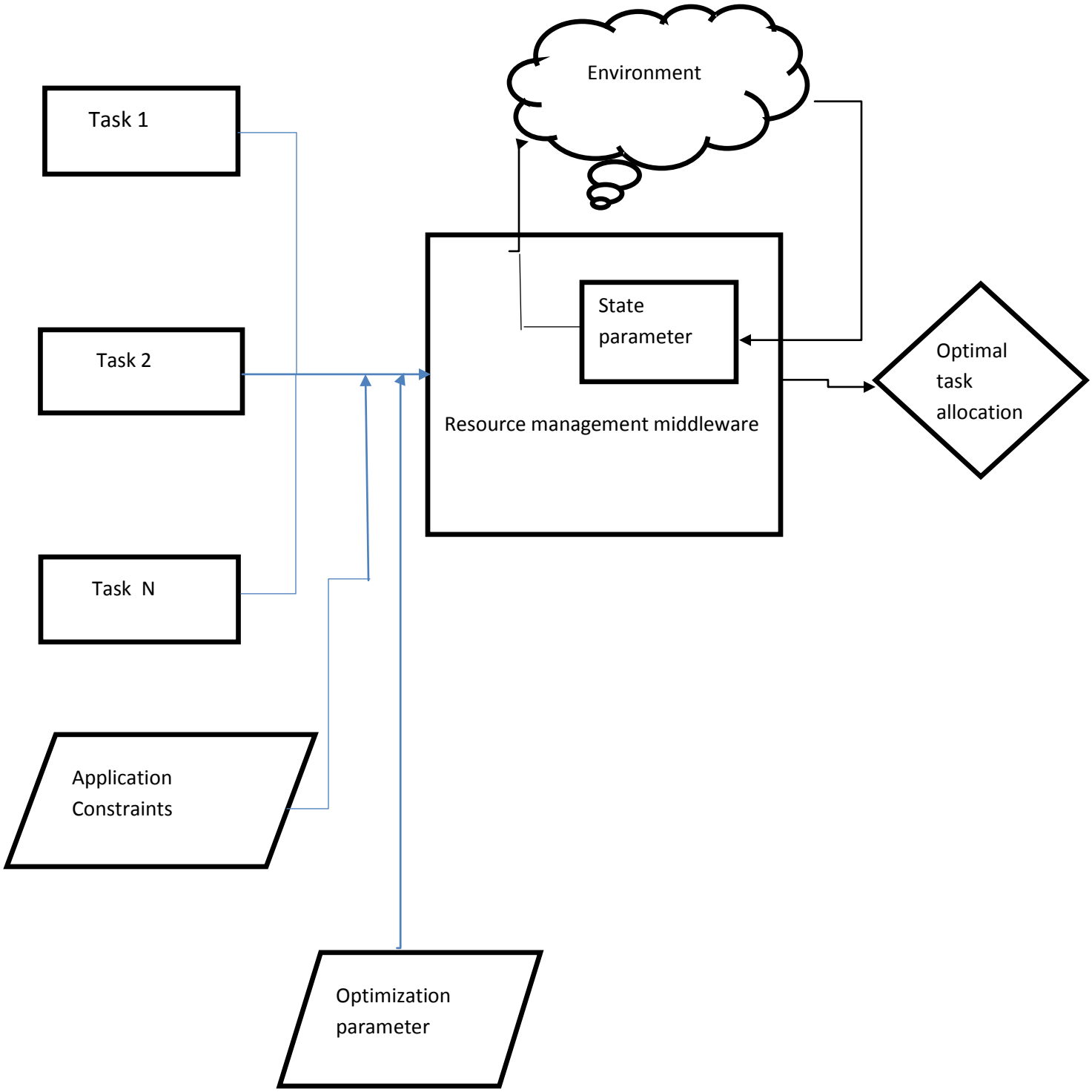


Figure 1 : Resource management problem in WSNs.

1.2 Goals

- Collaboration and coordination among untethered sensor nodes.
- Communicate with other nodes with only by local information.
- Task allocation and runtime adaptation of resources.
- Scalability and robustness should be maintained for resources.
- Increasing the lifetime of the network**.**
- Considering only local information, it will achieve the global perspective.

1.3 Thesis Layout

The rest of the paper is organized as follows.

Chapter two discusses related work. Chapter three is about application of wireless sensor networks and task scheduling. In chapter four we have discussed about artificial intelligence in WSN. Chapter five explains our system model. In chapter six we present our proposed method. Chapter seven is about experimental set up and results. Chapter eight concludes this paper with a brief summary and future work.

Chapter Two

2. Related works:

Most of the WSNs are energy constrained and also there is issue related cost effectiveness. So, it is very important to organize the task in such a manner so that we can achieve maximum output in accordance with energy consumption limit. For that, the best way is to create an environment which allow the sensor nodes to communicate with the neighboring nodes with cooperative algorithms when it requires to exchange information or any other preset work. It would optimize the energy usage and would maintain the scale of performance too. The works which exist in this area are following static task allocation where we are thinking of online cooperative method which can bring out the best maintaining all the constraints.

Guo et al. [6] proposed a self-adaptive task allocation scheduling strategy in WSN. They assume that the WSN is composed of a number of sensor nodes and a set of independent tasks which compete for the sensors. They neither consider distributed tasks scheduling nor the trade-off among energy consumption and performance. Giannecchini et al. [16] proposed an online task scheduling mechanism called collaborative resource allocation (CoRAl) to allocate the network resources between the tasks of periodic applications in WSNs. CoRAl neither addresses mapping of tasks to sensor nodes nor discusses explicitly energy consumption. Shah et al. introduced a task scheduling approach for WSN based on an independent reinforcement learning algorithm for online tasks scheduling. Their approach relies on a simple and fixed network topology consisting of three nodes and a static value for the reward function. They further consider neither any cooperation among neighbors nor the energy/performance trade-off. Our approach has some similarity with but is much more general and flexible since we support general WSN topologies, a more complex reward function for expressing the trade-off between energy consumption and performance, and cooperation among neighbors. F. Tirkawi et al. [14] proposed an adaptive task balancing scheme in which enhances the fairness of distributed sensing by switching priority between sensing and network tasks. There are sensing and network tasks. In sensing tasks the activities about signal processing and encoding is done. In network tasks the transmission of packet that involves the sending of packet and receiving of packets. The priority of the packet is based on the level of the node. Our approach has some similarity with Shat et al. [3] but it is

much more general and flexible since we support general WSN topologies, a more complex reward function for expressing the trade-off between energy consumption and performance, and cooperation among neighbors.

Chapter Three

3.1 Wireless sensor networks

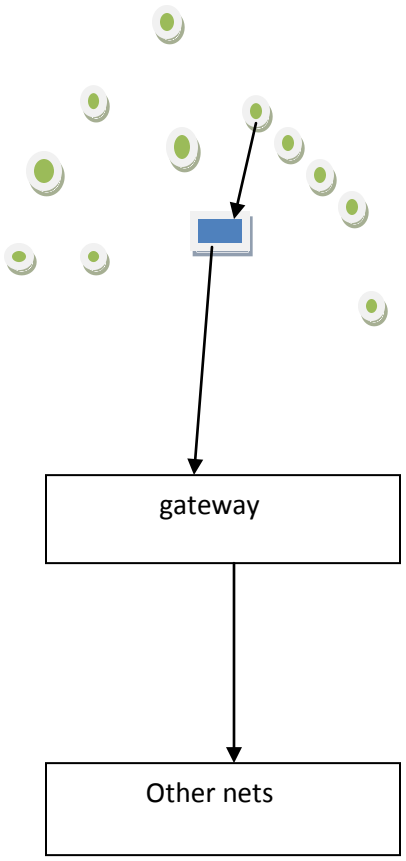
Since the start of the third Millennium, wireless sensor networks (WSNs) generated an increasing interest from industrial and research perspectives. A WSN can be generally described as a network of nodes that cooperatively sense and may control the environment enabling interaction between persons or computers and the surrounding environment. On one hand, WSNs enable new applications and thus new possible markets, on the other hand, the design is affected by several constraints that call for new paradigms. In fact, the activity of sensing, processing, and communication under limited amount of energy, ignites a cross-layer design approach typically requiring the joint consideration of distributed signal/data processing, medium access control, and communication protocols. WSNs have several common aspects with wireless ad hoc network and in many cases they are simply considered as a special case of them. This could be lead to erroneous conclusions, especially when protocols and algorithms designed for ad hoc networks are used in WSN. Applications, on top of the stack, set requirements that drive the selection of protocols and transmission techniques; at the other end, the wireless channel poses constraints to the communication capabilities and performance. Based on the requirements set by applications and the constraints posed by the wireless channel, the communication protocols and techniques are selected. Generally, when a node is in transmit mode, the transceiver drains much more current from the battery than the microprocessor in active state or the sensors and the memory chip. The ratio between the energy needed for transmitting and for processing a bit of information is usually assumed to be much larger than one (more than one hundred or one thousand in most commercial platforms) [9]. For this reason, the communication protocols need to be designed according to paradigms of energy efficiency, while this constraint is less restrictive for processing tasks. Then, the design of energy efficient communication protocols is a very peculiar issue of WSNs, without significant precedent in wireless network history. Most of the literature on WSNs deals with the design of energy efficient protocols, neglecting the role of the energy consumed when processing data inside the node, and conclude that the transceiver is the part responsible for the consumption of most energy. On the other hand, data processing in WSNs may require consuming tasks to be performed at the microprocessor, much longer than

the actual length of time a transceiver spends in transmit mode. This can cause a significant energy consumption by the microprocessor, even comparable to the energy consumed during transmission, or reception, by the transceiver. Thus, the general rule that the design of communication protocol design is much more important than that of the processing task scheduling is not always true.

The process of standardization in the field of WSNs is very active in the last years and an important outcome is represented by IEEE 802.15.4 which is a short-range communication system intended to provide applications with relaxed throughput and latency requirements in Wireless Personal Area.

A WSN can be defined as a network of devices, denoted as *nodes*, which can sense the environment and communicate the information gathered from the monitored field (e.g., an area or volume) through wireless links [1–9]. The data is forwarded, possibly via multiple hops, to a *sink* (sometimes denoted as *controller* or *monitor*) that can use it locally or is connected to other networks (e.g., the Internet) through a *gateway*. The nodes can be stationary or moving. They can be aware of their location or not. They can be homogeneous or not. This is a traditional single-sink WSN (see Figure 1, left part). Almost all scientific papers in the literature deal with such a definition. This single-sink scenario suffers from the lack of scalability: by increasing the number of nodes, the amount of data gathered by the sink increases and once its capacity is reached, the network size cannot be augmented. Moreover, for reasons related to MAC and routing aspects, network performance cannot be considered independent from the network size. A more general scenario includes multiple sinks in the network. Given a level of node density, a larger number of sinks will decrease the probability of isolated clusters of nodes that cannot deliver their data owing to unfortunate signal propagation conditions. In principle, a multiple-sink WSN can be scalable (i.e., the same performance can be achieved even by increasing the number of nodes), while this is clearly not true for a single-sink network. However, a multi-sink WSN does not represent a trivial extension of a single-sink case for the network engineer. In many cases nodes send the data collected to one of the sinks, selected among many, which forward the data to the gateway, toward the final user. From the protocol viewpoint, this means that a selection can be done, based on a suitable criteria that could be, for example, minimum

delay, maximum throughput, minimum number of hops, etc. Therefore, the presence of multiple sinks ensures better network performance with respect to the single-sink case.



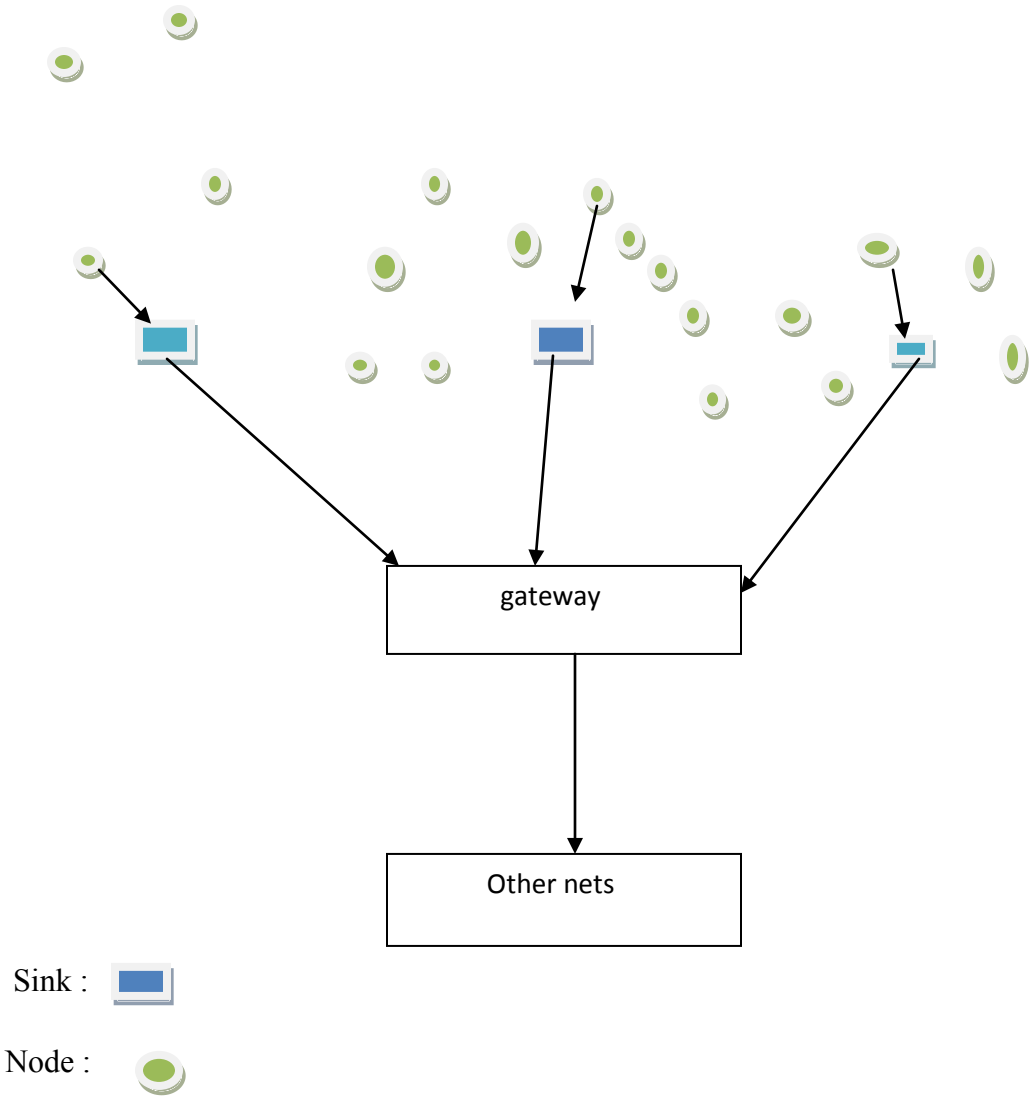


Figure 2 : Up_single sink WSN, Down_multisink WSN

3.2 Task Scheduling in WSN

Types of task scheduling:

First Come First Serve*:* First come first serve is a scheme that is based on the arrival time of the packet. If the packet comes first so it should be served firstly. For real time communication this scheme was used. The packets that come late, that should be scheduled at last. So that packet

requires more time to reach to the destination. The First Come First Serve is the simplest method but it is time consuming scheme for packets that comes late. The Execution of the FCFS policy is simply managed with a First In First Out (FIFO) queue. When the process is ready it enters the ready queue, its process control block is linked on the tail of the queue.

Earliest Deadline First*:* Earliest Deadline First is the dynamic scheduling algorithm used for the real time applications. Every packet having its deadline and before its deadline the packet should be transmitted . In this scheme the data packet which is having earlier deadline that packet scheduled first. This algorithm is considered as efficient algorithm than the FCFS that proposes a real-time communication architecture for large-scale sensor networks, whereby they use a priority based scheduler.

Pre-emptive scheduling*:* The Pre-emptive scheduling technique is one in which the packets that are having higher priority that should be scheduled or processed first. The packets having lower priority that should be processed at last.

Non Pre-emptive scheduling*:* In non pre-emptive scheduling the running packet should be goes on even if the new packet that is having the higher priority comes. The packet that should have to wait up to running packet scheduled.

Real time packet scheduling*:* There is highest priority for the real time packets & there is lower priority for the non real time packets. So the real time packets should be scheduled before the non real time packets. So the real time packets will reach their destination earlier than the non real time packets.

Non Real Time packet scheduling*:* In Non Real time packet scheduling the time is not so important factor. Hence the non real time packets have the lower priority.

Single Queue*:* Each node in the network having separate single queue. All the data packets are reside in that queue & that are scheduled on the basis of their type or on basis of size and also on the priority that packet have.

Multi level queue*:* In Multi level queue there are two or more no of queues for one node. Here the incoming packets are placed on the basis of priorities of that packet. The no of queues for the node is based on the level of the node. If the node is at lower level then the node will have minimum no of queues because it will require lot of time to transmit. The upper nodes should have more no of queues than the lower nodes. So the Energy consumption is balanced & reduces the delay in data transmission.

F. Tirkawi et al. proposed an adaptive task balancing scheme which enhances the fairness of distributed sensing by switching priority between sensing and network tasks. There are sensing and network tasks. In sensing tasks the activities about signal processing and encoding is done. In network tasks the transmission of packet that involves the sending of packet and receiving of packets. The priority of the packet is based on the level of the node. If packet p1 is at lower node and the packet p2 is at upper node so the packet p1 should have higher priority than the packet p2.

In this scheme, every sensor node, monitors its current depth in the network and if it moves close to base station (i.e. to depth two or one), then its sensing tasks are switched to higher priority for just X% of its duty cycle. Within this time, nodes can complete sensing and processing their local tasks and pass local data to base station.

3.3 Reinforcement Learning

Reinforcement learning is learning what to do and how to map situations to actions; so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics--trial-and-error search and delayed reward--are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem. Any method that is well suited to solving that problem, we consider to be a reinforcement learning method. A full specification of the reinforcement learning problem in terms of optimal control of Markov decision processes, the basic idea is simply to capture the most important aspects of the real problem facing a learning agent interacting with its environment to achieve a goal [3]. Clearly, such an agent must be able to sense the state of the environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment. The formulation is intended to include just these three aspects--sensation, action, and goal in their simplest possible forms without trivializing any of them.

Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in machine learning, statistical pattern recognition, and artificial neural networks. Supervised learning is learning from examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory--where one would expect learning to be most beneficial--an agent must be able to learn from its own experience [1].

One of the challenges that arise in reinforcement learning and not in other kinds of learning is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in

producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate its expected reward. The exploration-exploitation dilemma has been intensively studied by mathematicians for many decades. For now, we simply note that the entire issue of balancing exploration and exploitation does not even arise in supervised learning as it is usually defined.

Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. This is in contrast with many approaches that consider sub-problems without addressing how they might fit into a larger picture. For example, we have mentioned that much of machine learning research is concerned with supervised learning without explicitly specifying how such an ability would finally be useful. Other researchers have developed theories of planning with general goals, but without considering planning's role in real-time decision-making, or the question of where the predictive models necessary for planning would come from. Although these approaches have yielded many useful results, their focus on isolated sub-problems is a significant limitation.

Reinforcement learning takes the opposite tack, starting with a complete, interactive, goal-seeking agent. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. Moreover, it is usually assumed from the beginning that the agent has to operate despite significant uncertainty about the environment it faces. When reinforcement learning involves planning, it has to address the interplay between planning and real-time action selection, as well as the question of how environmental models are acquired and improved. When reinforcement learning involves supervised learning, it does so for specific reasons that determine which capabilities are critical and which are not. For learning research to make progress, important sub-problems have to be isolated and studied, but they should be sub-problems that play clear roles in complete, interactive, goal-seeking agents, even if all the details of the complete agent cannot yet be filled in.

One of the larger trends of which reinforcement learning is a part is that toward greater contact between artificial intelligence and other engineering disciplines. Not all that long ago, artificial intelligence was viewed as almost entirely separate from control theory and statistics. It had to do with logic and symbols, not numbers. Artificial intelligence was large LISP programs, not linear algebra, differential equations, or statistics. Over the last decades this view has gradually eroded. Modern artificial intelligence researchers accept statistical and control algorithms, for example, as relevant competing methods or simply as tools of their trade. The previously ignored areas lying between artificial intelligence and conventional engineering are now among the most active, including new fields such as neural networks, intelligent control, and our topic, reinforcement learning. In reinforcement learning we extend ideas from optimal control theory and stochastic approximation to address the broader and more ambitious goals of artificial intelligence.

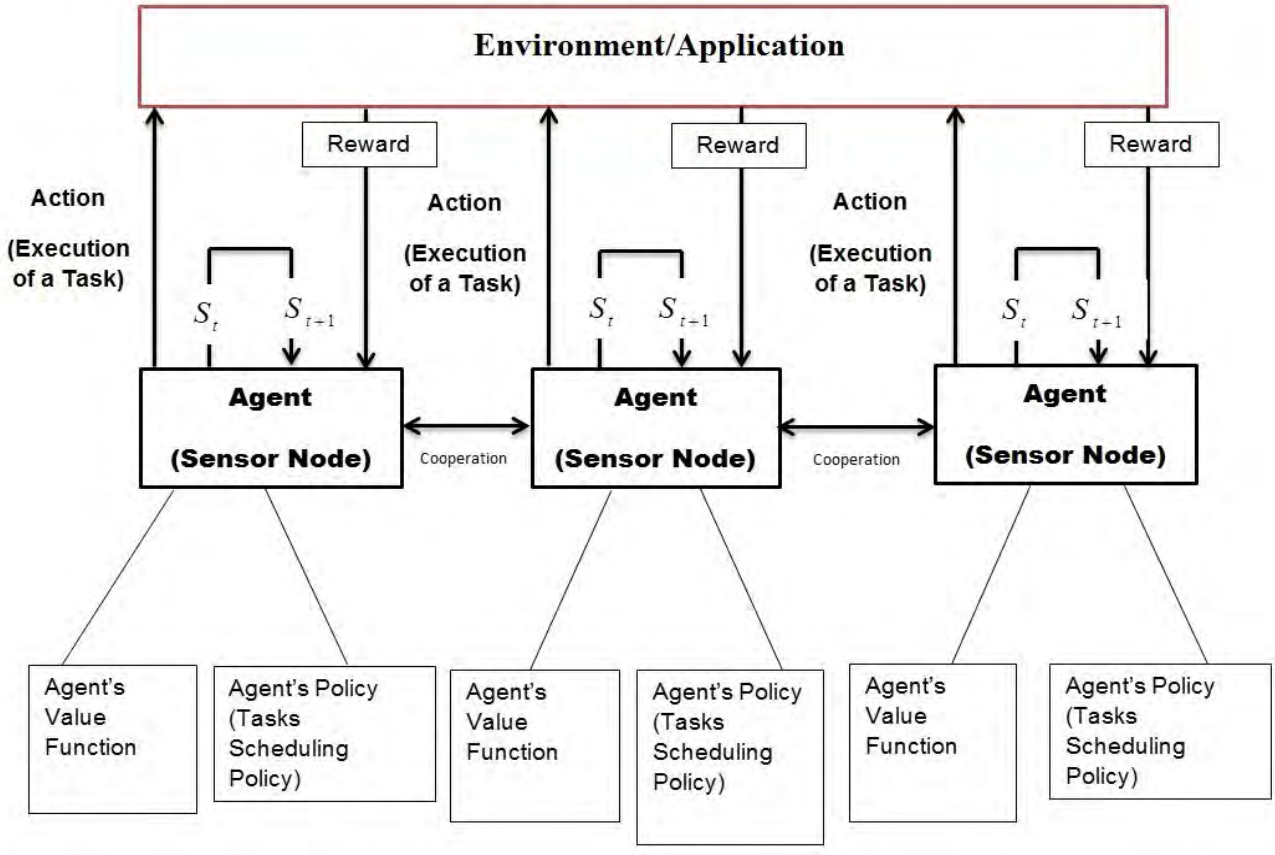


Figure 3 : Reinforcement Learning

The basic reinforcement learning model consists of:

1. A set of environment states S;
2. A set of actions A;
3. Rules of transitioning between states;
4. Rules that determine the scalar immediate reward of a transition; and
5. Rules that describe what the agent observes.

The algorithms we would be using in our proposed methods are mainly based on reinforcement learning. In this section, we would be describing about the basics of Q-learning algorithm and Sarsa lambda algorithm.

Q learning algorithm:

The goal of reinforcement learning is to figure out how to choose actions in response to states so that reinforcement is maximized. That is, the agent is learning a policy, a mapping from states to actions. We will divide the agent's policy into two components, how good the agent thinks an action is for a given state and how the agent uses what it knows to choose an action for a given state.

There are several ways to implement the learning process. We will focus on just one and that is, reinforcement learning in which the agent learns to assign values to state-action pairs [5]. We need first to make a distinction between what is true of the world and what the agent thinks is true of the world. First let's consider what's true of the world. If an agent is in a particular state and takes a particular action, we are interested in any immediate reinforcement that's received but also in future reinforcements that result from ending up in a new state where further actions can be taken, actions that follow a particular policy. Given a particular action in a particular state followed by behavior that follows a particular policy, the agent will receive a particular set of reinforcements. This is a fact about the world. In the simplest case, the Q value for a state-action pair is the sum of all of these reinforcements, and the Q value functions the function that maps from state-action pairs to values. But the sum of all future reinforcements may be infinite when there is no terminal state, and besides, we may want to weight the future less than the here-and-

now, so instead a discounted cumulative reinforcement is normally used: future reinforcements are weights by a value *gamma* between 0 and 1 (see below for mathematical details). A higher value of gamma means that the future matters more for the Q-value of a given action in a given state.

If the agent knew the Q-values of every state-action pair, it could use this information to select an action for each state. The problem is that the agent initially has no idea what the Q-values of any state-action pairs are. The agent's goal, then, is to settle on an optimal Q-value function, one which that assigns the appropriate values for all state/action pairs. But Q-values depend on future reinforcements, as well as current ones. It learns using these two principles, which are the essence of reinforcement learning:

- If an action in a given state causes something bad to happen, learn not to do that action in that situation. If an action in a given state causes something good to happen, learn to do that action in that situation.
- If all actions in a given state cause something bad to happen, learn to avoid that state. That is, don't take actions in other states that would lead you to be in that bad state. If any action in a given state causes something good to happen, learn to like that state.

The second principle is the one that makes the reinforcement learning magic happen. It permits the agent to learn high or low values for particular actions from a particular state, even when there is no immediate reinforcement associated with those actions. For example, in our mosquito-mango world, the agent receives a reward when it reaches the east end from the cell just to the west of it. It now knows that *that* cell is a good one to go to because you can get rewarded in only one move from it.

First, consider the optimal Q-value function, the one that represents what's true of the world.

$$Q(x_t, u_t,) = \Gamma(x_t, u_t,) + \gamma \max Q(x_{t+1}, u_{t+1})$$

That is, the optimal Q-value of for a particular action in a particular state is the sum of the reinforcement received when that action is taken and the discounted best Q-value for the state that is reached by taking that action. The agent would like to approach this value for each state-action pair. At any given time during learning, the agent stores a particular Q-value for each

state-action pair. At the beginning of learning, this value is random or set at some default. Learning should move it closer to its optimal value. In order to do this, the agent repeatedly takes actions in particular states and notes the reinforcements that it receives. It then updates the stored Q-value for that state-action pair using the reinforcement received and the stored Q-values for the next state. Assuming the Q-values are stored in a lookup table, the agent could use one of these update equations:

$$Q(x_t, u_t,) = \Gamma(x_t, u_t,) + \gamma \max Q(x_{t+1}, u_{t+1})$$

$$Q^{new}(x_t, u_t) = (1 - \eta)\, Q^{old}(x_t, u_t,) + \eta\, (\Gamma(x_t, u_t,) + \gamma \max Q(x_{t+1}, u_{t+1}))$$

The first equation sets the Q-value to be the sum of the reinforcement received and the discounted best Q-value for the next state. But this is usually a bad idea because the information just received may be faulty for one reason or another. It is better to update more gradually, to use the new information to move in a particular direction, but not to make too strong a commitment. The second update equation reflects this strategy. There is learning rate, *eta*, which controls the learning step size, that is, how fast learning takes place. The new Q-value for the state and action is the weighted combination of the old Q-value for that state and action and what the new information would lead us to believe. Later we will see how a neural network can replace the lookup table for storing Q-values.

SARSA (lambda) algorithm:

Actually, Q learning algorithm and SARSA algorithm are almost same except the fact that Sarsa only looks up the next policy Value, while Q learning looks up the next maximum policy value.

On-policy SARSA learns action values relative to the policy it follows, while off-policy Q-Learning does it relative to the greedy policy. Under some common conditions, they both converge to the real value function, but at different rates. Q-Learning tends to converge a little slower, but has the capability to continue learning while changing policies. Also, Q-Learning is not guaranteed to converge when combined with linear approximation.

In practical terms, under the ε-greedy policy, Q-Learning computes the difference between Q(s,a) and the maximum action value, while SARSA computes the difference between Q(s,a) and the weighted sum of the average action value and the maximum:

Q-Learning: $Q(s_{t+1},a_{t+1}) = \max_a Q(s_{t+1},a)$

SARSA: $Q(s_{t+1},a_{t+1}) = \varepsilon.\text{mean}_a Q(s_{t+1},a) + (1-\varepsilon) \max_a Q(s_{t+1},a)$

Chapter 4 : System Model

The primary objective of a task scheduler is to pick the best task as well as performing that in accordance with given knowledge. In our approach the WSN is composed by N nodes represented by the set N = {1,2,……..,N}. Each node has a known position (x,y) and a given sensing coverage range which is simply modeled by circle with radius R. All nodes within the communication range R can directly communicate with N and are referred to as neighbors. The available energy of node N is modeled by a scalar E_i. The WSN application is composed by A tasks (or actions) represented by the set A = {a_1,…….,a_n}. Once a task is started at a specific node, it executes for a specific (short)period of time and terminates afterwards. Each task execution on a specific node N requires some energy E_j and contributes to the overall application performance P. Thus, the execution of task a_j on node n_i is only feasible if $E_i \geq E_j$. The overall performance P is represented by an application specific metric. On each node, an online task scheduling takes place which selects the next task to execute among the A independent tasks. The task execution time is abstracted as fixed period. Thus, scheduling is required at the end of each period which is represented as time instant t_i. .The ultimate objective for our problem is to determine the order of tasks on each node such that the overall performance is maximized while the energy consumption is minimized. To achieve our aim at first we have some fixed states of the sensor nodes, some specific actions. Later on, we had applied the algorithms which are Q learning and SARSA(lambda). As we have discussed earlier that these two algorithms are almost same except the policy factor in SARSA(lambda), we need not make any changes in our system model for different algorithms and we would be using same reward

function for both of the cases. The specific application of our system is tracking multiple objects in a certain FOV.

The sate transition between the sensors entirely depends on previous state also the action. The policy we have following determines the next task at certain time. The policy is built on reward function which determines the quality of tracking. How we want to trade off between the factors completely depends on the weight of the reward function. RL maps the states of the environment to the actions that an agent should take in those states to get maximum reward while functioning.

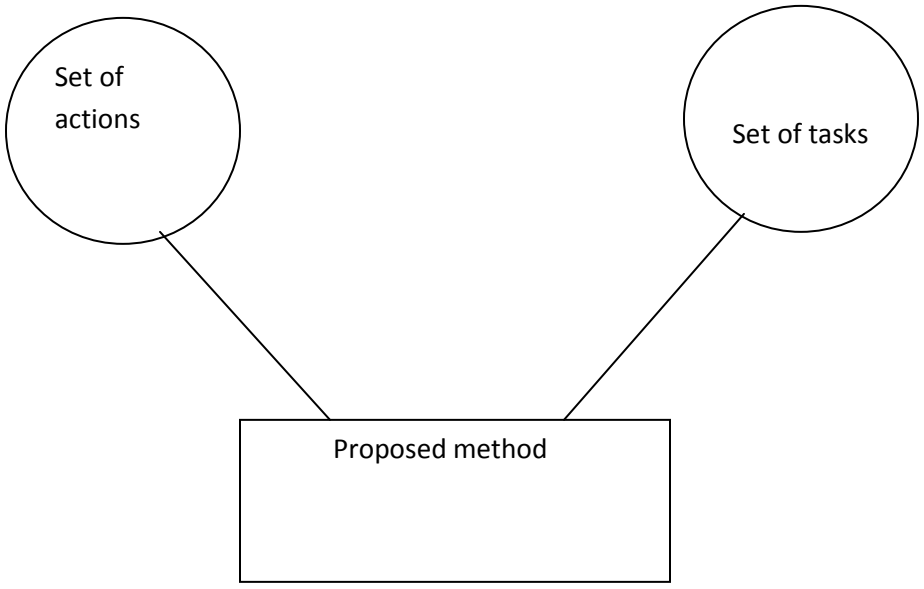


Figure 4 : Task mapping through proposed method

Chapter 5 : Proposed Method

In present time the applications of WSN has become tremendously popular. We can do disaster relief operations by drop nodes from aircraft over a wild fire, biodiversity mapping is also done by WSN. There are other applications too like monitoring stress after earthquake, reduce energy wastage, machine surveillance, precision agriculture, post operative or intensive care and long-term surveillance of chronically ill patients. But for applying our proposed method and to review the result we need an ideal yet simplified application that is, multiple object tracking. Object tracking is one the most popular application of wireless sensor networks. We consider a sensor network which may consists of any number of nodes. The sensing region of each node is called the field of view (FOV). Every node aims to detect and track all targets in the FOV. If the sensor nodes would perform tracking all the time then this would result in the best tracking performance. But executing target tracking is a process where we need energy all the time. So, task should only be executed when necessary and sufficient for tracking performance. Sensor nodes can cooperate with each other by informing neighboring nodes about moving targets. Neighboring nodes can therefore become aware of approaching targets. Here, we propose a cooperative RL method for scheduling the tasks in well organized way.

5.1: Set of Actions

As we have decided that we will keep our system model simple so that less computational error could take place, we have enlisted following actions to perform by the sensor nodes.

a. Sense Target: In our system the sensors track the object the moment they come under the FOV region. The FOV region is set by the required application which varies.
b. Track Target : While the target is moving in between in FOV region, the sensors would track the path of the target.
c. Send Messages: This is the function which passes the messages to the neighboring nodes about the velocity of the target, the position of the target and the duration of the target in FOV.

d. Store information: By this function the sensor can store limited information in the buffer of sensors.

e. Velocity : From this function we get to calculate the velocity of the object. Sometimes, in military bases while tracking object in a certain area it is very important to monitor the velocity of the objects. So that, they can detect early threat and take necessary steps. For calculating the velocity, we take two most recent target positions, i.e., $(x_{t,}y_t)$ at the time t_i and $(x_{t-1,}y_{t-1})$ at the time t_{i-1} .

Velocity = $\sqrt{\{(x_t-x_{t-1})^2 + (y_t-y_{t-1})^2\}} / (t_1-t_{t-1})$

f. Sleep : When there is no object in the region then it would be a wastage of resource to keep using it. So, after scanning the FOV without any object the sensors get back to the sleep mode.

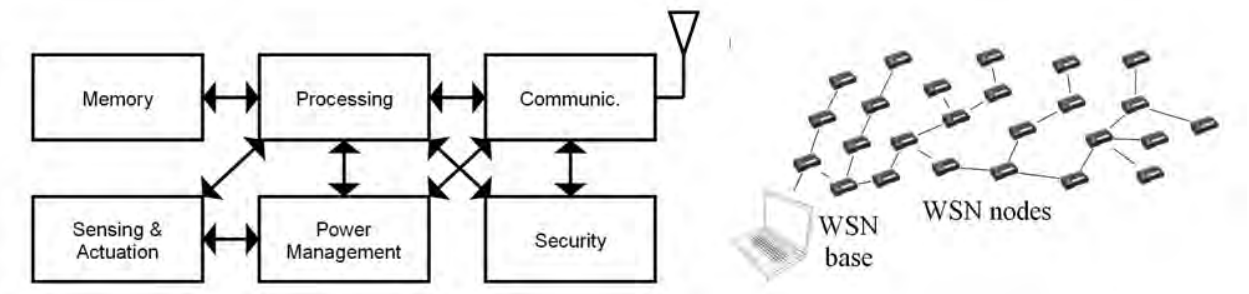


Figure 5 : Object tracking in WSN

5.2: Set of States:

There are exactly three sates for each of the sensors.

a. Sleep: This sates illustrate the situation where no objects in the field of view. Here, we need not to operate the sense_target as the target is far away. So, we could save some energy. To know the trigger point of going into the sleep state we calculate the local time and the expected arrival time. If the difference is more than five unit we take the decision to be in sleep state.
b. Ready: There is currently also no detected target in the node's FOV in this state. The node has received some relevant trajectory information and the expected arrival time of at least one target is in less than five clock ticks. The threshold for the time difference between the expected arrival time and the local clock is set to five based on our simulation studies. In this state, sensor nodes perform Sense Targets more frequently, since at least one target is expected to enter the FOV.
c. Active: In this sate we assume that there is at least one object in the FOV. So, the sensor keeps tracking it and pass the information to the neighboring node and store the information that is needed. The sensors would keep tracking until the object leaves the field of view.

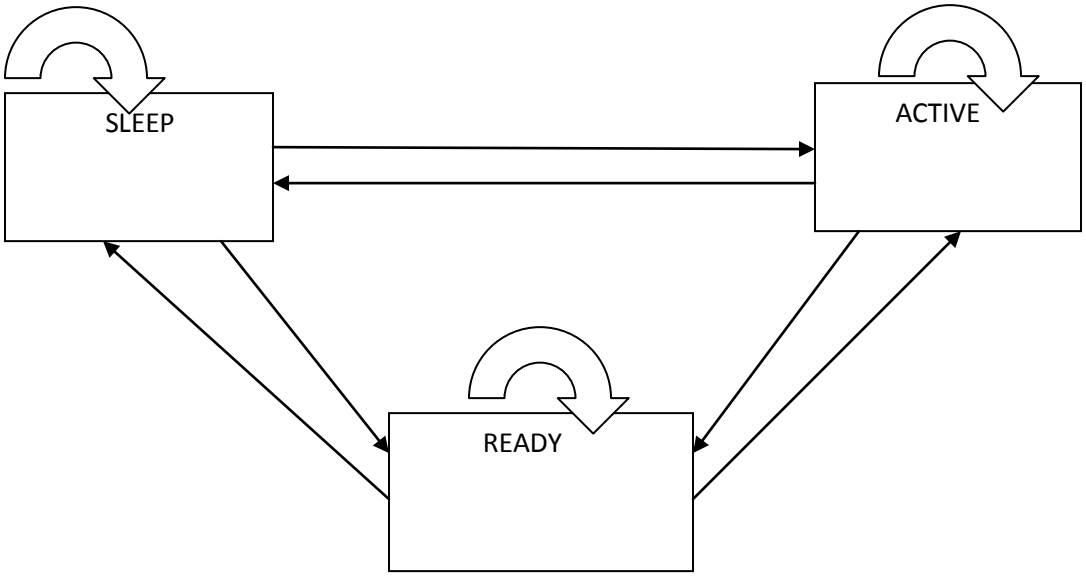


Figure 6 : States of the sensors

While the transition between the sate of sensors occurs there is some energy that is spent. So, we have assumed some constant energy for each of the transition. For example, if a sensor does not change its state it would not cost any energy. So, the cost is zero for this particular event. However, switching to another state there is certainly fact of spending energy. The following table shows the assumption of spending energy for the transitions.

Current State	Next State	Required Energy(unit)
Sleep	Sleep	0
Seep	Active	5
Sleep	Ready	3
Active	Active	0
Active	Sleep	2
Active	Ready	3
Ready	Ready	0
Ready	Sleep	2
Ready	Active	3

Table 1: Energy spent for each transition.

We assume there are three variables : FOV, DTT(data to transmit), RL (resource level) for state transitions:

1. FOV = 0 and RL >= Minimum energy required for Transmission and DTT = 0
2. FOV = 0 and RL>= Minimum energy required for Transmission and DTT = 1
3. FOV = 0 and RL< Minimum energy required for Transmission and DTT = 1
4. FOV = 1 and RL >= Minimum energy required for Transmission and DTT = 0
5. FOV = 1 and RL < Minimum energy required for Transmission and DTT = 0

6. FOV = 1 and RL < Minimum energy required for Transmission and DTT = 1
7. FOV = 1 and RL >= Minimum energy required for Transmission and DTT = 1
8. FOV = 1 and RL < Minimum energy required for Transmission and DTT = 0

5.3: Energy Model and Reward function :

We consider an agent to be energy e_cient when it minimizes most of the major sources of energy waste in WSN communication are idle listening, overhearing and unsuccessful transmissions, while quickly forwarding any packets in its queue to ensure low network latency. Formally, the energy efficiency for agent i in frame f is:

$$E = (1-\alpha) + (1-\beta) + (1-\gamma) + (1-\delta) + \varepsilon \qquad (1)$$

Where,

α = duration of idle listening of agent i within f frame

β = duration of overhearing of agent I within f frame

γ = amount of unsuccessful transmission

δ = sum of duration that each packet spent in queue

ε = battery life of agent

Our belief is that if each agent cares about others that will improve the performance of the whole system. To achieve that, we introduce the concept of an Effect Set (ES) of anode, which is the subset of that node's neighborhood, with which it communicates within a frame window. In

other words, the ES of agent i is the set Ni of nodes, who messages agent i (over) hears within a frame window. Thus, the energy efficiency of agent i is directly dependent on the actions of all agents in Ni and vice versa. As a result of the influence of agents on each other's performance, we form our hypothesis. We believe that if each agent seeks to increase not only its own efficiency, but also the efficiency of its ES, this will lead to higher energy efficiency of the whole system. For this reason, we set the reward signal of each agent to be equal to its mean Effect Set Energy Efficiency (R) over a frame window of size F. We define the R of agent i in the frame window F. Now, we have to merge our reward function with the performance factor to get the reward function.

$$R = \alpha \left(\sum_{i=0}^{n} \frac{E\ (current\ \ node\ ) + E(next\ \ node\ )}{N+1} \right) + (1-\alpha)\ \frac{t}{T} \qquad (2)$$

Here, α =balancing factor,

t = number of tracked target in FOV,

T= total number of tracked target

E indicates the energy model.

Algorithm 1:

The Q learning algorithm :

1. Initialize Q(s,a) arbitrarily
2. Repeat for each episode
3. Initialize S
4. Repeat for each step of episode
5. Chose a from S using policy derived from Q
6. Take action a, observe r (from equation 1), s`
7. $Q^{new}(x_t, u_t) = (1- \eta)\ Q^{old}(x_t, u_t,) + \eta\ (\Gamma(x_t, u_t,) + \gamma \max Q^*(x_{t+1}, u_{t+1})\ )$
8. S ← a s`

9. Update learning rate
   Until s is terminal

Algorithm 2 :

The SARSA algorithm:

1. Initialize Q (s,a) arbitrarily and e (s,a) = 0, for all s,a
2. Repeat for each episode
3. Initialize s, a
4. Repeat for each step of episode
5. Chose a` from s` using policy derived from Q
6. δ <= R + γQ (s`,a`) – Q (s,a)
7. e(s,a) <= e(s,a) +1
8. For all s,a: Q(s,a) <= Q(s,a) + $\alpha\delta$e(s,a)
   e(s,a) <= $\gamma\lambda$e(s,a)
   s<= s` : a <=a`
   until s is a terminal.

The learning rate is updated after each of the iteration.

Chapter 6 : Experiment Set Up and Results

6.1 The simulator

As we have demonstrated earlier that our system consist of many sensors within particular networks and when some objects come near to the territory the sensor nodes start tracking it. For getting the real life experience we have designed sample environment which consist every possible things which is needed for tracking like network, sensor nodes, object with specific attributes. We have used c# as the laguage for coding and the platform is microsoft visual studio.

There is a screenshot of the simulator we have designed in the next page. Now, we are going to describe the options located in the simulator as buttons.

Deploy Network: By pressing this button we originate the network in the system.

Network Size: from this slide bar we can increase or decrease the network size.

Sensor Radius: From this slide bar we can increase of decrease the sensing area.

Sensor period: we can manipulate the sensor's living period from this slide bar.

Sensor cost: sensor cost can be increased and decrised from here.

Transmission area : Transmission area of the sensor is directed from this slide bar.

Transmit cost: tranmission cost of the sensor can be increased or decreased from here.

Receive cost: from this slide bar we can set up the receive cost.

Start simulation: after setting all the parameters if we press the start simulation button, the simulation will be started.

Again, we can choose the roution method from the left down option and aslo can fix the system's energy.

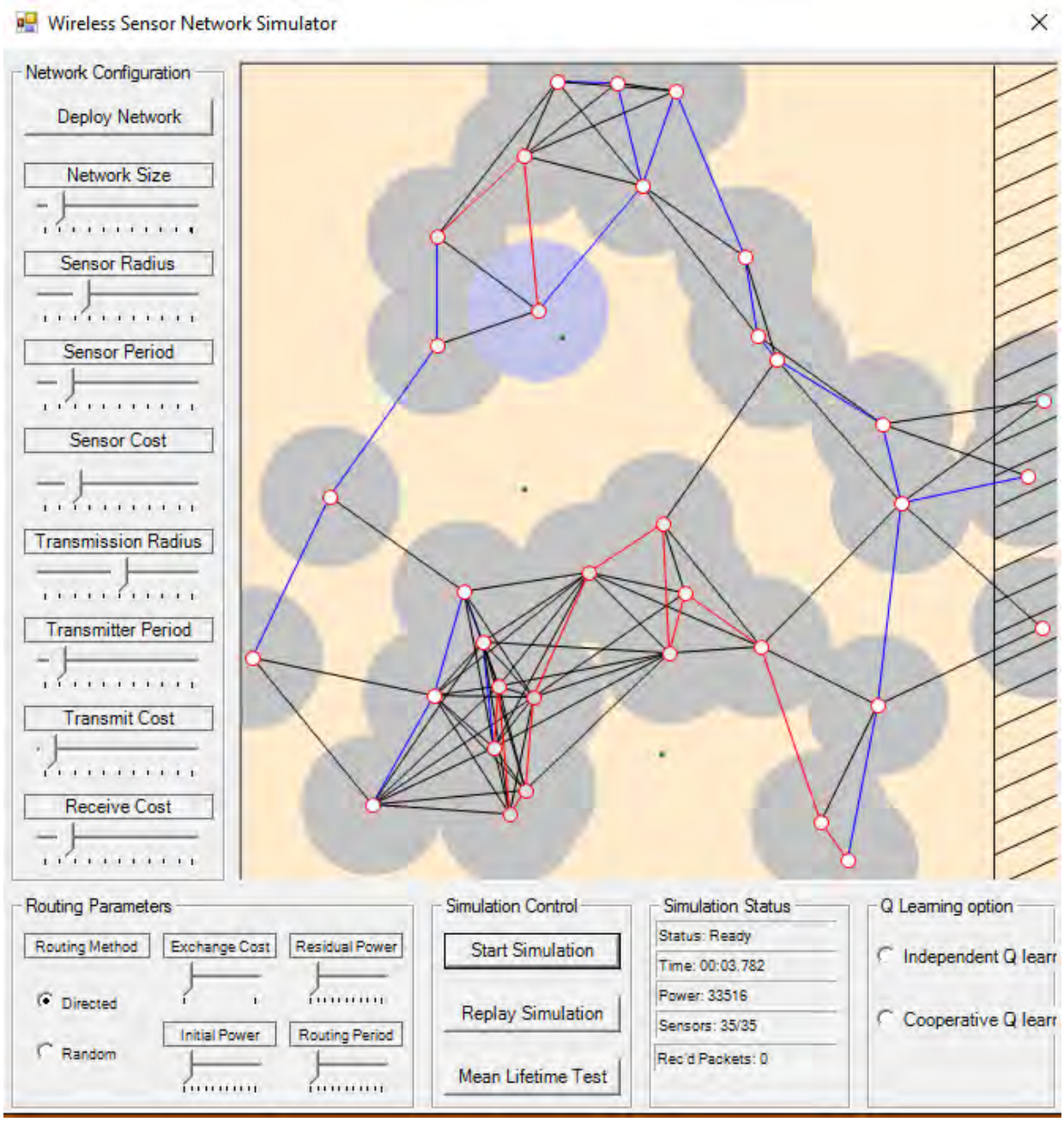


Figure 7: The Simulator

Right after we hit the start simulation we get another pop-up window which gives us the results those are remaining energy, DTT, FOV, reward, energy used for each sensing.

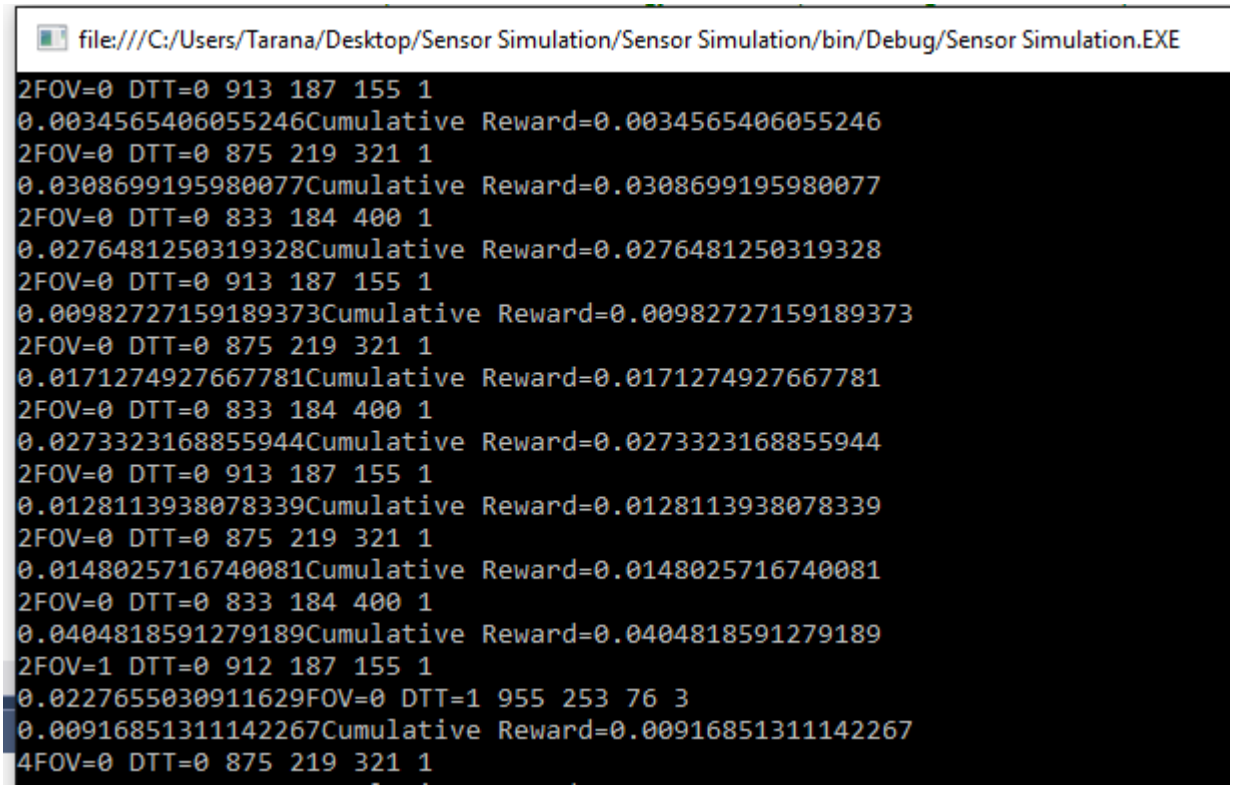


Figure 8: The output window

6.2: The random walk theory

As our targeted objects are always moving we have used random walk algorithm to track down their movement in a well organized way.

Consider a weighted network – either directed or undirected – with n nodes denoted by j=1, 2 …, n; and a random walk process on this network with a transition matrix M. The m_{jk}element of M describes the probability of the random walker that has reached node i, proceeds directly to node j. These probabilities are defined in the following way.

$$M(i,j)= \frac{a_{ij}}{\sum_{j=1}^{n} a_{ij}} \qquad (3)$$

where a_{ij} is the (i,j) th element of the weighting matrix A of the network. When there is no edge between two nodes, the corresponding element of the A matrix is zero.

The random walk closeness centrality of a node i is the inverse of the average mean first passage time to that node:

$$C_i^{RwC} = \frac{n}{\sum_{j=1}^{n} H(j,i)} \qquad (4)$$

Mean first passage time:

The mean first passage time from node i to node j is the expected number of steps it takes for the process to reach node j from node i for the first time:

$$H(i,j) = \sum_{r=1}^{\infty} rP(i,j,r) \qquad (5)$$

where P(i,j,r) denotes the probability that it takes exactly r steps to reach j from i for the first time. To calculate these probabilities of reaching a node for the first time in r steps, it is useful to regard the target node as an absorbing one, and introduce a transformation of M by deleting its j-th row and column and denoting it by M_{-j}^{r-1}. As the probability of a process starting at i and being in k after r-1 steps is simply given by the (i,k)th element of M_{-j}^{r-1}, P(i,j,r) can be expressed as

$$H(i,j) = \sum_{r=1}^{\infty} r \sum_{k \neq j} ((M_{-j}^{r-1}))_{ik} m_{kj} \qquad (6)$$

Substituting this into the expression for mean first passage time yields

$$H(i,j) = \Sigma_{k \neq j} ((I - M_{-j})^{-2})_{ik} m_{kj} \qquad (7)$$

Using the formula for the summation of geometric series for matrices yields

$$H(i,j)=\Sigma_{k\neq j}\left((I-M_{-j})^{-2}\right)_{ik}m_{kj} \qquad (8)$$

where I is the n-1 dimensional identity matrix.

For computational convenience, this expression can be in vector form as

$$H(i,j)=(I-M_{-j})^{-1}e \qquad (9)$$

Where, H(i,j) is the vector for first passage times for a walk ending at node j, and e is an n-1 dimensional vector of ones. Mean first passage time is not symmetric, even for undirected graphs.

6.3 Experiment Results :

We have assumed the energy consumption against each of the action is fixed and it remained the same in both approaches.

Action	Energy Consumption
Sense target	3 units
Track target	7 units
Send messages	5 units
Store information	2 units
Velocity	6 units
Sleep	1 unit

Table 2: : Energy consumption against each of the actions.

The symbol alpha (α) represent the learning rate, set between 0 and 1. Setting it to 0 means that the Q-values are never updated, hence nothing is learned. Setting a high value such as 0.9 means that learning can occur quickly. We have varied the value of α in the range for five values which are 0.15, 0.35, 0.55, 0.75 and 0.90.

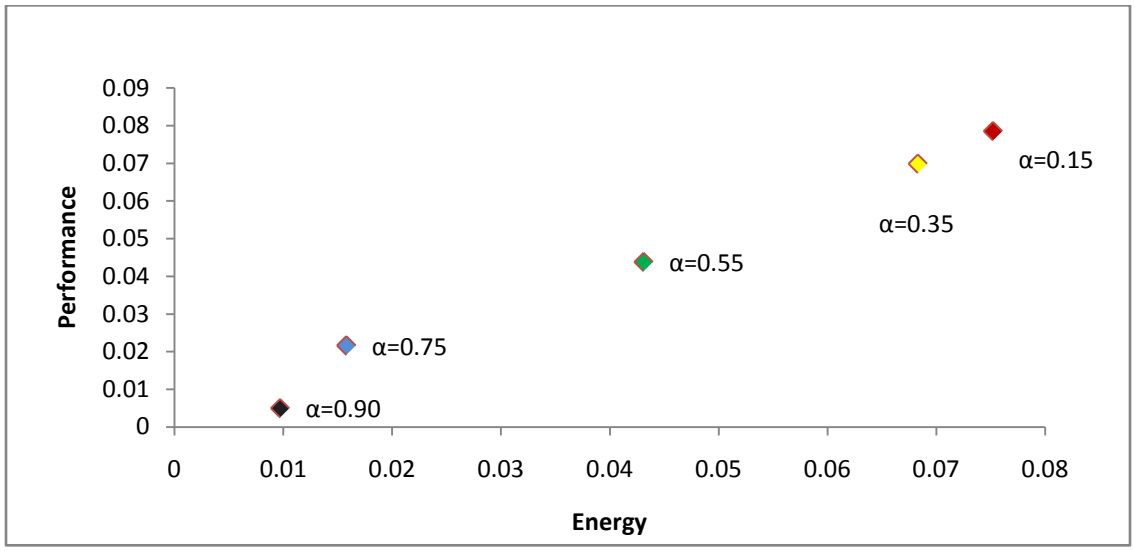


Figure 9: Performance Vs Energy varying the learning rate

We can see it from the graph that the value of learning rate when comes near to 1 the energy consumption become the lowest. For the smaller value of α gives us better performance. We get optimum value of both performance and energy when α remains in middle that is 0.55 . Therefore, we can set the value for the learning rate in accordance with our demand like if we are more focused on good performance or we want to save energy or relevant cost.

The symbol β represents the discount factor in reward function in sarsa lambda algorithm. We have varied the β as 0.15, 0.35,0.55, 0.75, 0.90 and got the trade of between tracking performance and energy consumed.

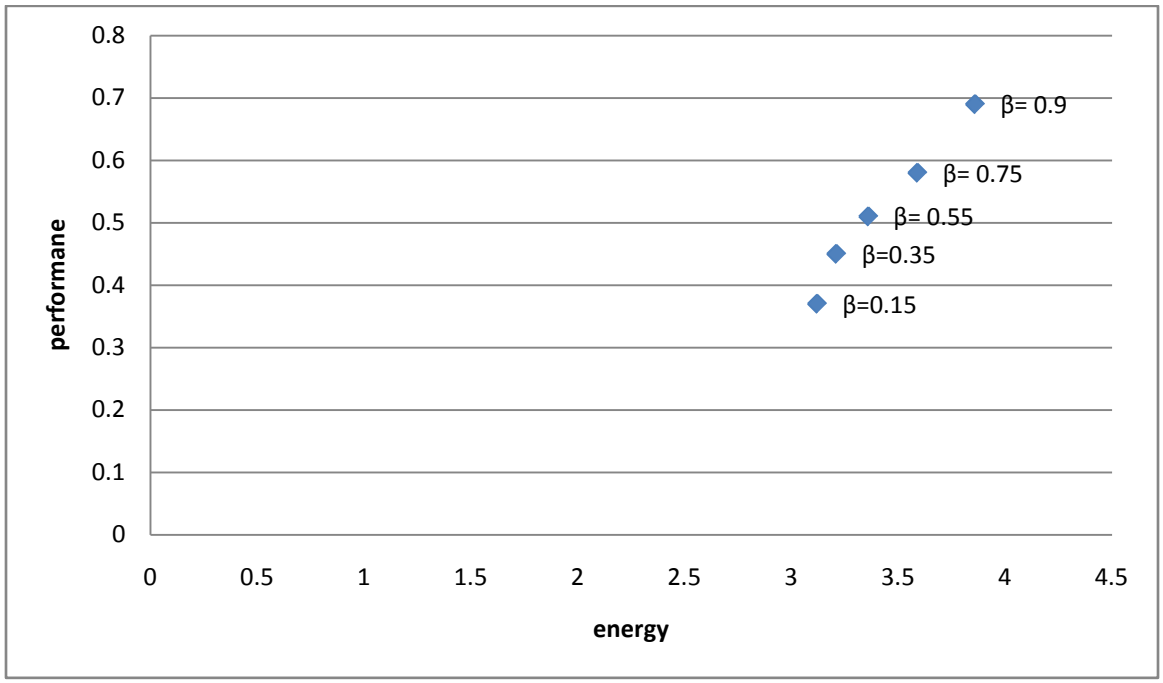


Figure 10 : Performance vs energy varying the balancing factor

We also varied the network size of the system for both of the cases and found out the trade-off. For N= (10,20,30) we took ten value for each of the cases and calculated their average. Hence, we found out that sarsa give us the better results.

Network Size	Q learning		SARSA	
	performance	energy	performance	energy
N=10	0.73	5.27	0.82	6.23
N=20	0.84	6.1	0.91	7.4
N=30	0.94	7.4	0.98	8.21

Table 3 : Trade of between energy and performance varying the network size

Chapter Seven : Conclusion and Future work

8.1 Conclusion

In the world of computer science WSN has taken over huge research interests as the application of WSN has becoming more popular and effective. The work load is not always simple in terms of computational complexity as well as the capacity of whole system. So, scheduling of tasks is very important to achieve our goal. For our proposed method we had set of tasks and set of actions for the sensor nodes over any network. The task mapping is done by the reinforcement learning based proposed method. We have designed energy model as well as the reward function.

Here, we have tried to demonstrate that task scheduling is very effective if we use online reinforcement learning based method where each of the sensor nodes works as intelligent agent. That merges two very important sectors of computer studies one is complex network system and another is Artificial Intelligence as machine learning is a vital part of it. We have worked with two algorithms in here but in future we are looking forward to work with other variant of reinforcement learning with more complex system design. We have found that SARSA (lambda) algorithm provides better tracking performance than Q learning. On the other hand, Q learning consumes less energy comparing with the SARSA (lambda) learning.

8.2 Future Work:

We have so many scopes to broad up the area of this research. We mainly kept our system model simple to examine our RL based proposed method. We have applied two algorithms here but there are many algorithms which follow reinforcement learning. In near future, we are interested to work with existing algorithms and also will try to develop a new one. We will like to cluster based task scheduling in near future and see which method is better.

Chapter Eight : References

1. M.I. Khan and B. Rinner, “Energy-aware Task Scheduling in Wireless Sensor Networks based on Cooperative Reinforcement Learning” , iccw, 2014
2. M. I. Khan and B. Rinner, “Resource Coordination in Wireless Sensor Networks by Cooperative Reinforcement Learning,” in Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, 2012, pp. 895 – 900.
3. K.Shah, “Reinforcement leaning based strategies for adaptive wireless sensor network management”, UTA, 2010
4. M. Li and Y. Liu. “Underground Structure Monitoring with WirelessSensor Networks”. In Proc. of ACM/IEEE IPSN, 2007
5. C. Frank and K. Romer, “Algorithms for Generic Role Assignments in Wireless Sensor Networks,” in Proceedings of the ACM Conference on Embedded Networked Sensor Systems, 2005.
6. W. Guo, N. Xiong, H.-C. Chao, S. Hussain, and G. Chen, “Design andAnalysis of Self Adapted Task Scheduling Strategies in WSN,” *Sensors*,vol. 11, pp. 6533–6554, 2011.
7. D. Wolpert and K. Tumer, “Collective intelligence, data routing and braess’ paradox,” J. Artif. Intell. Res. (JAIR), vol. 16, pp. 359–387, 2002.
8. L. Dai, H.K. Xu, T. Chen, C. Qian, L.J. Xie, “ A multi objective optimization algorithm of task scheduling in WSN.
9. U. A. Khan and B. Rinner, “Dynamic Power Management for Portable,Multi-Camera Traffic Monitoring,” in Proceedings of the IEEE Real-
   Time and Embedded Technology and

   Applications Symposium, 2012.

10. M.H.A Awadalla, “Task Mapping and Scheduling in Sensor Networks” IAENG International Journal of Computer Science, 40:4, IJCS_40_4_05
11. A. A. Rahman, I.U. Mollah, M. Naznin, “Multiple target tracking using kinetics in wireless sensor networks” Wireless Sensor Network, 2011, 3, 263-274 doi:10.4236/wsn.2011.38027 Published Online August 2011

12. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo, "Adopt: asynchronous distributed constraint optimization with quality guarantees," Artificial Intelligence,vol. 161, no. 1-2, pp. 149–180, 2005.

13. C. Buratti, A. Conti, D. DArdari "An Overview on Wireless Sensor Network And it's Evolution. Sensors 2009, 9, 6869-6896; doi:10.3390/s90906869

14. A. sirisha, Dr.K.Shreelakshmi, F.K. Tiwari Multilevel Priority based scheduling scheme for sensor network operating system‖ in Proc. 2014 International journal of advanced scientific and technical research, issue 4 vol.3, PP. 35-43.

15. N. Nasser, L. Karim, and T.Taleb, ―Dynamic Multilevel Priority Packet Scheduling Scheme for Wireless Sensor Network‖, in IEEE Transactions on wireless communications Vol 12, No 4, April 2013

16. S. Giannecchini, M.Caccamo and C.Shih, "Collaborative Resource Allocation in Wireless Sensor Networtks," in Proceedings of the Euromicro Conference on Real-Time Systems, 2004.

17. E. Yoneki and J. Bacon, "A Survey of Wireless Sensor Network Technologies: Research Trends and Middle-ware's Role," Ph.D. Dissertation, University of Cam-bridge, Cambridge, 2005.

18. C. Sharp, S. Schaffet, A. Woo, N. Sastri, C. Karlof, S. Sastry and D. Culler, "Design and Implementation of a Sensor Network and Autonomous Interception," Pro-ceedings of the 2nd European Workshop on Wireless Sensor Networks, Berkeley, 2005.

19. J. Jeong, T. Hwang, T. He and D. Du, "(MCTA) Target Tracking Algorithm based on Minimal Contour in Wire-less Sensor Networks," Technical Report, University of Minnesota, Twin Cities, 2007

20. Younis, M.; Munshi, P.; Al-Shaer, E. Architecture for Efficient Monitoring and Management of Sensor Networks. In Lecture Notes in Computer Science 2839; Springer-Verlag: Berlin, Germany, 2003; pp. 488-502.

21. D. Jung, T. Teixeira, and A. Savvides, "Sensor node lifetime analysis: Models and tools," ACM Transactions on Sensor Networks, vol. 5, no. 1, Feb 2009.

22. D. H. Wolpert and K. Tumer. An introduction tocollective intelligence. Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center, 2008.

23. J. Carle and D. Simplot-Ryl. Energy-efficient area monitoring for sensor networks. IEEE Computer Society, 47(2):40{46, 2004.
24. W.-C. Feng, E. Kaiser, W. C. Feng, and M. L. Baillif, "Panoptes: Scalable low-power video sensor networking technologies," ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 1, no. 2, pp. 151–167, May 2005.
25. P. Basu, W. Ke, and T. D. C. Little, "Dynamic task-based anycasting in mobile ad hoc networks," Mobile Networks and Applications, vol. 8, no. 5, pp. 593–612, Oct. 2003.