Md. Muhidul Islam Khan, M.Sc.

Resource-Aware Task Scheduling in Wireless Sensor Networks

DISSERTATION

to gain the Joint Doctoral Degree
Doctor of Philosophy (PhD)

Alpen-Adria-Universität Klagenfurt
Fakultät für Technische Wissenschaften

in accordance with

The Erasmus Mundus Joint Doctorate in Interactive and Cognitive Environments

Alpen-Adria-Universität Klagenfurt and Universitá degli Studi di Genova

1. Begutachter: Univ.–Prof. Dr. techn. Bernhard Rinner
Institut für Vernetzte und Eingebettete Systeme,
Alpen-Adria-Universität Klagenfurt

2. Begutachter: Prof. Dr. Carlo S. Regazzoni
Dipartimento di Ingegneria Biofisica ed Elettronica,
Universitá degli Studi di Genova

Klagenfurt, July 2014

Acknowledgments

This PhD Thesis has been developed in the framework of, and according to, the rules of the Erasmus Mundus Joint Doctorate on Interactive and Cognitive Environments EMJD ICE [FPA n° 2010-0012] with the cooperation of the following Universities:

Alpen-Adria-Universität Klagenfurt – AAU

Queen Mary, University of London – QMUL

Technische Universiteit Eindhoven – TU/e

Università degli Studi di Genova – UNIGE

Universitat Politècnica de Catalunya – UPC

According to ICE regulations, the Italian PhD title has also been awarded by the Università degli Studi di Genova.

First Reviewer

Univ.-Prof. Dr. Bernhard Rinner

Institute für Informatik-Systeme

Alpen-Adria-Universität Klagenfurt, Austria

Second Reviewer

Prof. Dr. Carlo S. Regazzoni

Dipartimento di Ingegneria Biofisica ed Elettronica

Universitá degli Studi di Genova

Declaration of Honor

I hereby confirm on my honor that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisors have been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

I am aware that a false declaration will have legal consequences.

Md. Muhidul Islam Khan Klagenfurt, July 25th, 2014

Abstract

Wireless sensor networks (WSN) are an attractive platform for various pervasive computing applications. A typical WSN application is composed of different tasks which need to be scheduled on each sensor node. However, the severe resource limitations pose a particular challenge for developing WSN applications, and the scheduling of tasks has typically a strong influence on the achievable performance and energy consumption. In this thesis we propose different methods for scheduling the tasks where each node determines the next task based on the observed application behavior. We propose a framework where we can trade the application performance and the required energy consumption by a weighted reward function and can therefore achieve different energy/performance results of the overall application. By exchanging data among neighboring nodes we can further improve this energy/performance trade-off. We evaluate our approaches in a target tracking application. Our simulations show that cooperative approaches are superior to non-cooperative approaches for this kind of applications.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In recent years, wireless sensor networks (WSN) have become an attractive platform for observing real world phenomena. A WSN consists of hundreds or thousands of tiny sensor nodes capable of sensing, communicating wirelessly with their neighbors and performing on-board computation. Sensor nodes are small devices equipped with one or more sensors, one or more transceivers, a processor, a storage device and actuators. Sensor nodes are typically powered by batteries which pose strong limitations on energy but also on computation, storage and communication capabilities [50, 3].

Sensor networks perform a wide range of applications like area monitoring, military applications, target tracking, health and security, etc. Basically WSNs can be applied to monitoring and computation-based applications [17]. Monitoring application includes fire detection, traffic monitoring, and wildlife habitat monitoring. Computation-based applications include target detection, tracking, and acoustic signal processing which may require image processing and filtering operations.

WSNs applications require various kinds of independent tasks like sensing, transmitting, receiving, processing, etc., and various resources like processors, bandwidth, memory, battery power, etc. Each sensor node is responsible for performing a particular task at each time step. Task execution consumes energy from the fixed energy budget of the sensor nodes. The scheduling of the individual tasks has a strong influence on achievable performance and energy consumption [60].

One of the important characteristics of a WSN is that the network changes dynamically over time. The network dynamics may happen because of node mobility, or because of node failure due to energy depletion. Energy consumption includes the energy cost arising due to computation and communication tasks at each sensor node. In order to accommodate the dynamic nature of WSN, the need for adaptive and autonomous task scheduling is well recognized [41].

In this thesis we focus on resource-aware task scheduling in WSN. Resource-aware, effective task scheduling is very important so that the WSN can know the best task to execute during upcoming time slots. We propose different methods for online scheduling of tasks so that a better resource consumption/performance

trade-off is achieved.

1.1 Motivation

Recent technological advances have enabled the deployment of tiny, energy sensitive sensor nodes capable of local processing and wireless communication. Each battery-operated sensor node is able to perform limited processing. When sensor nodes communicate with neighboring nodes and form a network they then have the ability to measure a given physical environment in detail. Thus, a sensor network can be described as a collection of sensor nodes that communicate with each other to perform application-specific actions [9].

Typically a sensor node has four basic components. Sensing components are responsible for gathering data from the environment, processing components for data processing, wireless communication components for data transmission and energy supply components for power supply to the sensor nodes [63]. Sensor nodes should be chosen for an application depending on the physical phenomena to be monitored like temperature, pressure, light, humidity, etc. Table 1.1 shows the classification and examples of some sensors.

Type	**Examples**
Temperature	Thermistors, thermocouples
Pressure	Pressure gauges, barometers, ionization gauges
Optical	Photodiodes, phototransistors, infrared sensors, CCD sensors
Acoustic	Piezoelectric resonators, microphones
Mechanical	Strain gauges, tactile sensors, capcitive diaphragms
Motion, vibration	Accelerometers, gyroscopes, photo sensors
Flow	Anemometers, mass air flow sensors
Position	GPS, ultrasound-based sensors, infrared-based sensors
Electromagnetic	Hall-effect sensors, magnetometers
Chemical	pH sensors, electromechanical sensors, infrared gas sensors
Humidity	Capacitive and resistive sensors, hygrometers
Radiation	Inonization detectors, geiger-Mueller counters.

Table 1.1: Classification and examples of sensors [18].

In some applications of WSNs, sensor nodes communicate directly with a centralized controller or a base station. A WSN could be a collection of autonomous nodes that communicate with each other by forming multi-hop radio networks and maintaining connectivity in an ad hoc manner. Such WSNs could change their topology dynamically, when the connectivity changes either to the mobility of nodes or to the failure of nodes due to energy depletion [91].

A WSN monitors the environment by measuring the physical parameters like pressure, temperature, and humidity. Constant monitoring, detection of specific

events, military battlefield surveillance, object tracking, in-network data aggregation, forest fire detection, flood detection, habitat exploration of animals, patient monitoring, home appliances are some of the common applications in sensor networks [2].

The number of nodes in a typical WSN is much higher than in a typical ad hoc network and dense deployment of these nodes is desired in a WSN to ensure the coverage and connectivity. Sensor nodes must be cheap in order to make their huge numbers. Sensor network hardware should be power-efficient and reliable in order to maximize the lifetime of the network.

The lifetime of a WSN is extremely crucial for most applications. The lifetime of the network depends on the energy consumed by the sensor nodes during the performance of the different tasks required by the application. Typically sensor nodes perform packet transmission, sensing, processing, and idle mode operation. Transmitting packets is the most energy-consuming task. Sensing and even in idle mode operation consume a consistent amount of power as well [30]. The lifetime of the network can be extended through energy efficient hardware and protocol design. Protocol helps to operate the sensor nodes based on the application demands.

For example, in a surveillance application, sensors (e.g., acoustic, video, seismic) are distributed throughout an area. The application will have some quality of service (QoS) or performance requirements like minimum percentage sensor coverage, minimum probability of missed detection of an event [79]. At the same time, the application needs to continue this performance as long as is possible with limited battery power, bandwidth, memory, etc. A careful design of the network protocol is required in order to maintain the trade-off between performance and resource consumption.

WSNs face some challenges and limitations in performing the application. Sensor networks operate in an unattended fashion in remote geographic locations. Nodes may be deployed in harsh and hostile environments. Wireless communication links in a WSN operate in the radio, infrared, or optical range. To facilitate the global operation of WSNs, the selected transmission channel must be available worldwide. Deploying and managing a high number of sensor nodes in close proximity is a challenge. Any time after deployment, topology changes due to changes in sensor node position, power availability, dropouts, malfunctioning, jamming and so on. A sensor node may need to fit into a tight module on the order of $2cm \times 5cm \times 1cm$ or even as small as a $1cm \times 1cm \times 1cm$; a hardware constraint for developing a WSN applications [74].

Wireless communications pose challenges to WSN design. An increasing distance between a sensor node and the base station increases the required transmission power which is very energy consuming. Therefore, the distance is divided into several shorter distances with multi-hop communication for sensor nodes protocols. Multi-hop communication requires cooperation among sensor nodes to identify efficient routes and to serve as relays. Many sensor nodes use power-aware protocols where radios are turned off when they are not in use [69].

The large scale and the energy constraints of WSNs make centralized algorithms infeasible. For example, in routing application, a base station can collect information from all sensor nodes and inform each node of its route. On the other hand, in decentralized algorithms, sensor nodes cooperate with the neighboring nodes to make localized decisions.

Sensor nodes typically exhibit a strong dependency on battery life. Generally, sensor nodes use AA alkaline cells or one $Li - AA$ cell. Energy consumption can be allocated to three functional domains: sensing, communication and data processing.

A performance metric is required to determine the performance of a particular application. Since energy-efficiency is the key characteristic of a WSN, the performance metric is typically associated with the trade-off between energy consumption and performance of the application. For example, in a typical surveillance system, it may be required that one sensor node remains active in every sub-region of the network. In this case, the performance metric can be determined by the percentage of the environment that is actually covered by the active sensors. In a typical tracking application, the performance metric can be considered as the accuracy of target location estimation provided by the network [61].

Generally a WSN node can perform the following tasks: transmit, receive, sleep, sense, data processing, etc. Each task requires a specific energy consumption level. Energy consumption in sense task is relevant to a specific application, while energy consumption in other tasks are related to work process of a node [34]. In addition, energy consumption for the radio component in a node is much greater than that of other components in the majority of WSN applications. Task scheduling helps to find the best allocation of tasks to resources for the specific application by performing some triggering activities. This triggering can be performed in several ways such as offline, periodic, online or can be issued by changes in network. It also helps in learning the usefulness of tasks in any given state to maximize the total amount of reward over time [5].

For determining the next task to execute, the sensor nodes need to consider the impact of each available task on energy budget and the application's performance. There is a trade-off between application performance and resource consumption, and the task scheduler of the node should be able to adapt to changes in the environment. For example, in a target tracking application, sensor nodes should frequently execute the tracking task when objects are within the field of view (FOV). Since tracking is very resource consuming, this task should be avoided when no object to track is nearby. Thus, task scheduling is an important issue to improve the energy/performance trade-off, and we investigate scheduling methods which are able to learn effective scheduling strategies in dynamic environments. Since resource-awareness is an important aspect we consider energy consumption for task scheduling and aim for low resource consumption of the scheduling algorithms. The ultimate goal is to achieve a high application performance while keeping the resource consumption low.

In this thesis we propose different methods of task scheduling. The proposed

methods help to learn the best task scheduling strategy based on previously observed behavior and is further able to adapt to changes in the environment. A key step here is to exploit cooperation among neighboring nodes, i.e., the exchange of information about the current local view of the application's state. Such cooperation helps to improve the trade-off between resource consumption and performance.

1.2 Challenges

As a WSN is a resource constrained network, there are challenges associated with task scheduling. Figure 1.1 shows the basic task scheduling mechanism in WSN. Here we observe that there is an available resource infrastructure for the WSN application. Basically battery power, memory, and processing functionality form the resource infrastructure. For performing the application, sensor nodes execute some tasks. Task scheduling methods help to schedule the tasks in a way that the resource is optimized with the goal of maximizing the lifetime of the network. Sensor nodes consume some resources from the resource budget for each executed task. Scheduling can be performed online, offline or periodically.

Some of the challenges and design issues for task scheduling are as follows:

- Which task scheduling method to use?
  - To select a suitable task scheduling method is a challenge. There are several methods for task scheduling (cp. Section 2.4 for more details). Due to the dynamic nature of WSNs, adaptive, and online task scheduling methods should be chosen.
- How to monitor a resource assignment?
  - Battery power, memory and processing functionality are the main resources of WSNs. We need to monitor the resources after executing the tasks. For example, a task execution consumes some energy from the energy budget of the sensor node which has also an effect on the performance of the application. The goal is to maintain a better trade-off between energy consumption and application performance.
- How to model the WSN and application?
  - To model the WSN with some number of nodes having different parameters such as sensing radius, transmission radius, and resource consumption model is a challenge. These parameters provide an impact on task scheduling. Tasks scheduling should meet the application demands. For that, careful consideration should be given to model the WSN and the application.

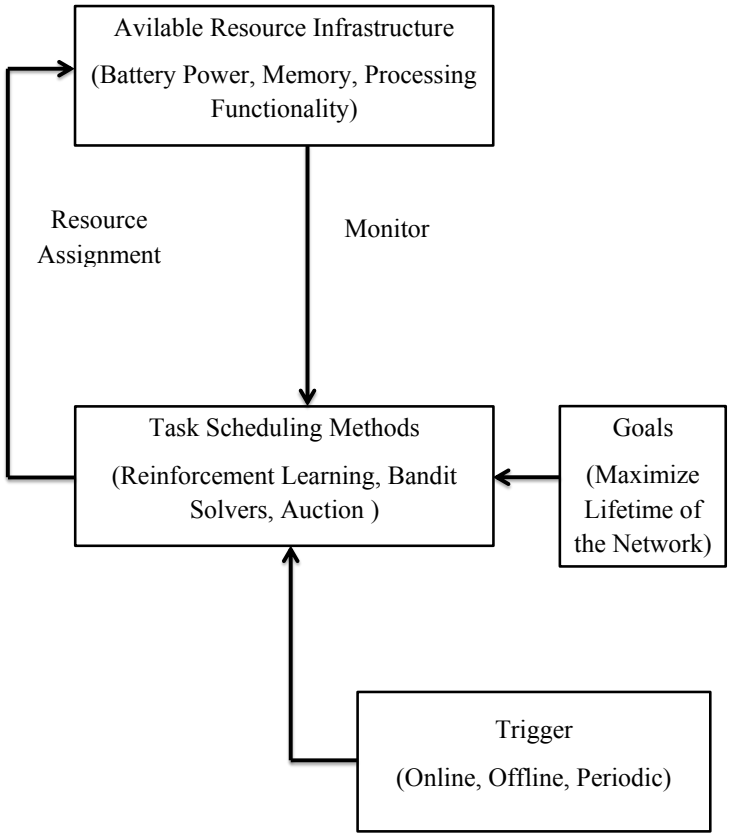


Figure 1.1: Task scheduling in WSN

- How to learn the best scheduling?
  - There is a wide range of learning methods in computer science like supervised learning, unsupervised learning, reinforcement learning, etc. The choice of a particular learning method is however a challenge since there is no a priori information or a very little information exists for a dynamic system like WSNs.
- Which type of triggering should be used?
  - Triggering for task scheduling can be performed online, offline or periodically. Appropriate triggering depends on the application and task scheduling methods.

1.3 Contributions

The main contributions of this thesis are following:

- Perform research on state-of-the-art.
  - We perform state-of-the art research on resource-aware task scheduling methods in WSN. We find that most of the existing approaches do not provide online scheduling of tasks. Further, they do not consider cooperation among neighboring nodes for task scheduling.
- Apply online learning methods for task scheduling.
  - We apply online learning methods for task scheduling. We further consider the cooperation among neighboring nodes for task scheduling.
  - We propose an online task scheduling method with cooperation among neighboring nodes. First, we propose a cooperative Q learning, a variant of the reinforcement learning, for task scheduling. The main contribution here is cooperation among neighboring nodes. This work is published in [41]. This is our initial work to apply reinforcement learning. We consider a set of actions, a set of states, and scalar values for the reward. We consider a fixed topology with only three nodes in this work.
  - We propose a market-based method for resource-aware task scheduling in WSN. We apply a combinatorial auction based method for task scheduling. We compare our proposed approach with static and random task scheduling methods. Our proposed method outperforms the others, in terms of residual energy, comparing with the existing static and random task scheduling. This work is published in [45].
  - We update the work [41] with different topologies consisting of a different number of nodes, weighted reward function, and exchanging data among neighboring nodes. We propose a cooperative reinforcement learning (CRL), state-action-reward-state-action ($SARSA(\lambda)$) algorithm for task scheduling. We compare our proposed method with independent reinforcement learning (RL) [67]. This work is published in [42].
  - We propose a method using bandit solvers for task scheduling. We use the classical adversarial algorithm, Exp3 (Exponential-weight algorithm for exploration and exploitation) for task scheduling. Exp3 and CRL achieve similar results in terms of performance/resource consumption trade-off. The paper based on this study is currently under review [43].
- Evaluate/compare based on simulation.

– We evaluate RL, CRL, and Exp3 in terms of the performance/resource consumption trade-off. The paper based on this study is currently under review [44].

1.4 Thesis outline

The remainder of this thesis is organized as follows:

Chapter 2 offers an overview of the literature published in this area. In this chapter, we provide background on wireless sensor networks (WSN), task scheduling, and online learning. We then review related works based on task scheduling in a WSN. We conclude this chapter with a discussion of the novelty of our work.

Chapter 3 provides the description of the system model. In this chapter, we describe the system model of our task scheduling methods. First, we delineate the basic system model for our task scheduling followed by the system model for combinatorial auction method. Afterward, we describe our considered set of actions, set of states, and the reward function.

Chapter 4 describes the proposed methods for the task scheduling in a WSN and the task scheduling for the target tracking application. We describe the proposed resource-aware task scheduling methods for the WSN. Here, we consider the system model, the set of tasks, the set of states, and the reward function described in Chapter 3. First, we describe the combinatorial auction method. Then we describe the cooperative reinforcement learning methods (Q learning, $SARSA(\lambda)$) and bandit solvers method. We conclude this chapter with a description of our proposed method for task scheduling in an object tracking application.

Chapter 5 shows the results and discussions. We describe our simulation environment first. We apply reinforcement learning (RL), cooperative reinforcement learning (CRL), exponential weight for exploration and exploitation (Exp3) bandit solvers, and combinatorial auction for the task scheduling in a WSN. We consider an object tracking application and apply our methods for task scheduling. We evaluate our applied methods. We conclude this chapter with the discussion of simulation results.

Chapter 6 concludes the thesis, summarizing the goals, contributions, results, and future work.

CHAPTER

2 Background and related work

In this chapter, we offer an overview of wireless sensor network (WSN), task scheduling, and online learning. We then proceed with related works based on tasks scheduling in WSN. We conclude this chapter with a discussion of the novelty of our work.

2.1 Wireless sensor networks

The recent promotion of micro-electro-mechanical systems (MEMS), digital electronics and wireless communication enable the evolution of low power, multifunctional sensor nodes that are small in size and communicate with their neighbors. Wireless sensor networks (WSN) consist of hundreds or thousands of small, battery-powered, and wireless devices called sensor nodes with on-board processing, communication, and sensing capabilities. Each node consists of one or more processors, multiple types of memory, a wireless transceiver, a power source (e.g., batteries, solar cells), and diverse sensors [4].

Figure 2.1 shows the components of a general sensor node. Sensing elements are a crucial part of a sensor node. Sensors collect information about physical phenomena. Selecting the sensors for a particular application is a challenge. For example, if we propose to calculate distance using audio sensors, we will also need to measure humidity and temperature. Because the velocity of the sound depends heavily on both temperature and humidity of the environment. The sensors have a particular range to sense or detect the targets or events. This range is called sensing coverage [91].

One of the components of a sensor node is the storage device. Depending on the overall WSN structure, the requirement for storage devices at each node should be different. For instance, if the WSN follows the structure that all sensor nodes should send data to the central node then there is less importance for local storage on each single sensor node. But if the goal is to reduce the amount of communication in the network, local storage will be more significant. Flash storage, micro disks and nano-electronics based magnetoresistive random-access memory are some of the common

storage devices in WSN [82].

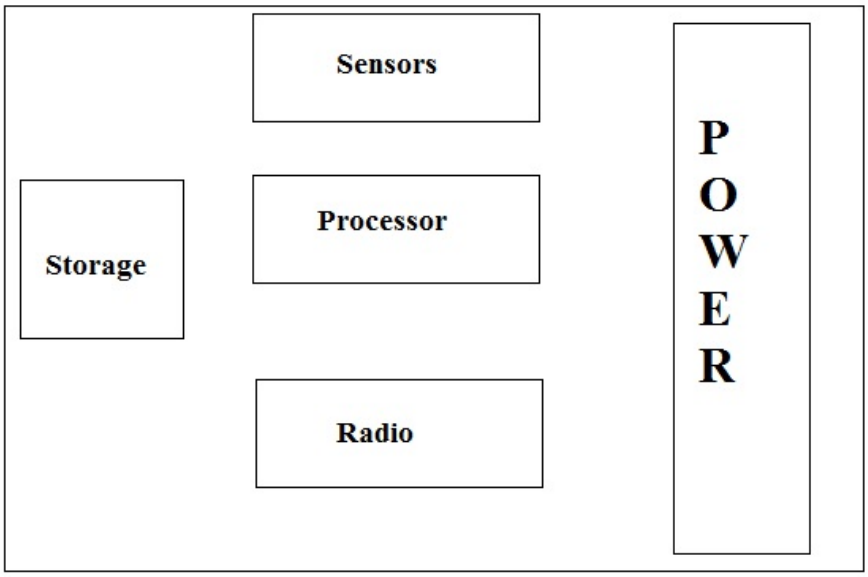


Figure 2.1: Components of a wireless sensor node.

Radios are the communication component of a sensor node. The design and selection of radios are important as the energy budget for sending and receiving messages usually dominates the total energy budget of a sensor node. Radios have a particular range called the radio coverage. Within the radio coverage, sensor nodes can communicate with each other [32]. Some low-power radio-based sensor devices use a single-channel RF transceiver operating at 916 MHz and some sensor systems use a bluetooth-compatible 2.4 GHz transceiver with an integrated frequency synthesizer.

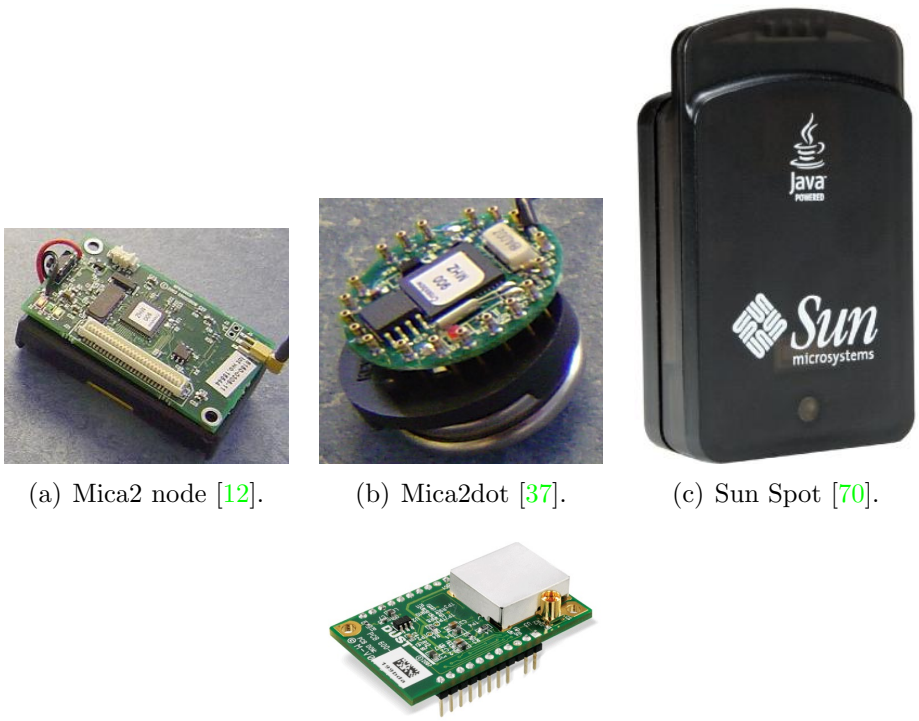
Energy is the main constraint for a sensor node. So, power supply is a very significant factor. There are two different ways to address the concept of power supplies. The foremost is to equip a sensor node with a source of energy. Presently, the most used option is to use high-density battery cells. Another alternative is to employ the energy harvesting cells. The solar cell is one form of energy harvesting cells. Battery operation for sensors used in commercial applications is typically based on two AA alkaline cells or one $Li - AA$ cell.

Processors are capable of on-board processing. Digital signal processor (DSP), field programmable gate array (FPGA) processor and application specific integrated circuit (ASIC) are some examples of the processor. There are different microcontrollers currently used in sensor node design such as Atmel AVR 8bit, Intel PXA271 "Bulverde", Texas Instruments MSP430, and Atmel AtMega 1281 [40].

Sensor nodes work together to monitor an area in order to obtain information about the environment. Based on the way of deployment of the nodes, there are two types of WSNs: structured and unstructured. In an unstructured WSN, sensor

nodes are densely deployed. Sensor nodes may be randomly placed into the field. After the deployment, the unstructured WSN is left unattended to perform the environment monitoring. Network maintenance is more difficult in an unstructured network comparing with the structured network since there are so many nodes in an unstructured network. Sensor nodes are pre-determined to be located in specified locations in a structured WSN. The advantage of a structured network is less network maintenance with less management cost [90].

Figure 2.2 shows some commercially available sensor nodes.



(a) Mica2 node [12]. (b) Mica2dot [37]. (c) Sun Spot [70].

(d) Smartdust node [39].

Figure 2.2: Examples of some commercial sensor nodes.

Figure 2.3 shows an example of a multi-hop data communication in a wireless sensor network. The sensor node which detects an event, sends the information to the neighboring nodes which terminates at a special node called the base station. The base station is like a gateway that connects one network to another. The base station has enhanced capabilities as compared to simple sensor nodes. Here, a link between two sensor nodes represents a single hop and that the data is eventually transferred via multiple hops to the base station.

When the transmission ranges of the radios of all sensor nodes are large enough to communicate directly with the base station, each sensor node can send their data directly using a single-hop. However, most sensor networks cover a huge area and in this case radio transmission power should be kept minimum in order to make the

network energy efficient.

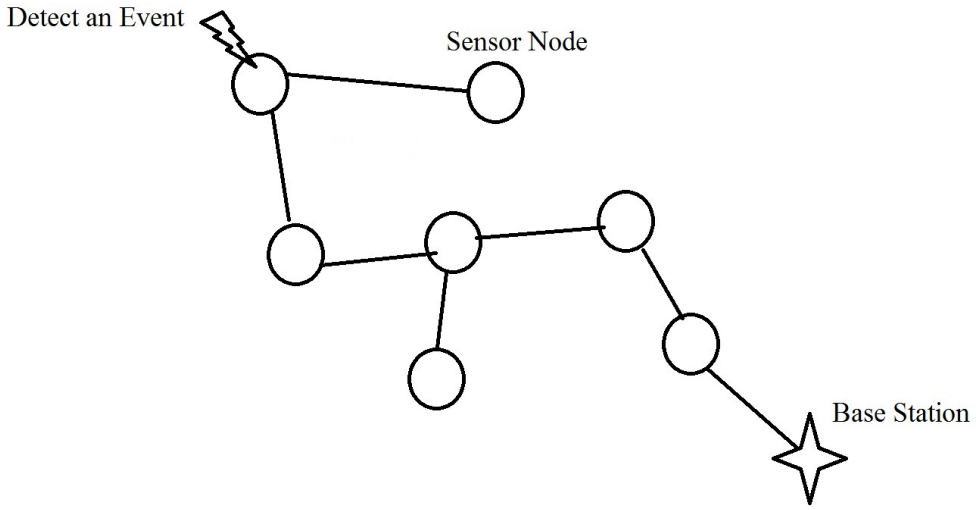


Figure 2.3: Wireless sensor network.

Basically, sensor network applications can be categorized into two types; monitoring the real world phenomena, and tracking targets. Monitoring applications include military security detection, habitat monitoring, industrial monitoring, health and environmental monitoring. Habitat monitoring includes monitoring human habitats (smart homes and residential care centers, for instance), as well as animal habitats. Business monitoring includes inventory monitoring. Public/Industrial monitoring includes structural, factory, inventory, machinery and chemical monitoring. Environmental monitoring includes weather, temperature and pressure monitoring. Tracking applications include tracking vehicles, humans, animals, and objects. Figure 2.4 shows the overview of sensor applications [90].

A WSN has its own resource and design constraints. Resource constraints include a limited energy, low bandwidth, limited processing capability of the central processing unit, limited storing capacity of the storage device, and short communication range. Design constraints are application-dependent and also depend on the environment being monitored. The environment acts as a major determinant regarding the size of the network, deployment strategy, and network topology. The number of sensor nodes or the size of the network changes based on the monitored environment. For example, in indoor environments, fewer nodes are needed to form a network in a limited space, whereas outdoor environments may require more sensor nodes to cover a huge unattended area. The deployment scheme also depends on the environment. Ad hoc deployment is preferred over a pre-planned deployment when the environment is not accessible and the network is composed of a vast number of nodes [47].

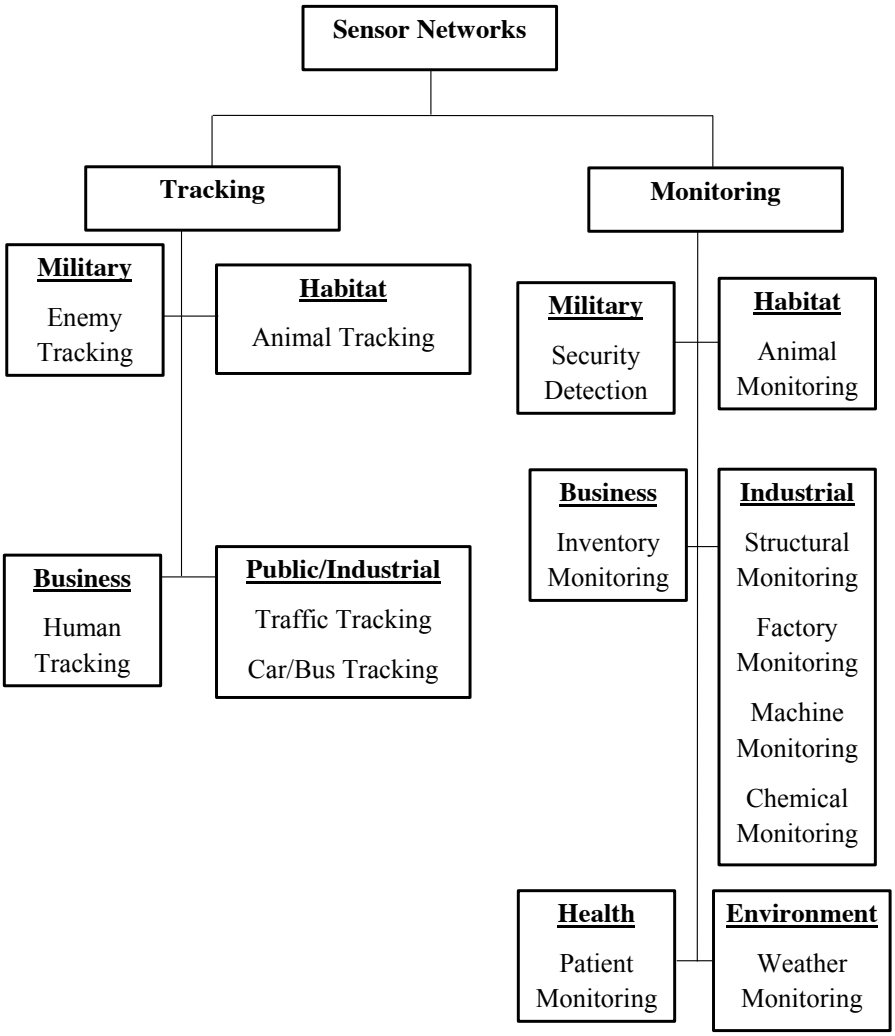


Figure 2.4: Overview of sensor applications [90].

Since sensor nodes operate on limited battery power, energy is a vital resource for a WSN. When a node is depleted of energy, it will disconnect from the network, which has a significant impact on the performance of the application.

One of the major reasons of energy consumption for the communication in WSN is idle mode consumption. When there is no transmission/reception, sensor nodes consume some energy for listening and waiting for the information from the neighboring nodes. Over hearing is another source of energy consumption. Over hearing means that a node picks up packets that are destined for other nodes. Packet collision is another issue of energy consumption. Collided packets should be retransmitted which require extra effort in energy consumption. Protocol overhead is

also a reason for energy consumption.

In some protocols, nodes are clustered together, to maintain scalability, that is; to allow spatial expansion if necessary. In that case, there is a cluster head (CH) for each cluster. A CH is responsible for collecting data from the sensor nodes of that particular cluster. The CH can communicate with other CH and can send the processed information to the base station (BS). Both CH and BS are special types of sensor nodes having more energy and processing functions [68]. Figure 2.5 shows the basic data communication in a clustered WSN. Data communication can be categorized into two types: intra-cluster communication and inter-cluster communication. In a particular cluster, sensor nodes communicating with the CH of that cluster is called intra-cluster communication. CH communicating with the CH of other clusters or with the BS is called inter-cluster communication.

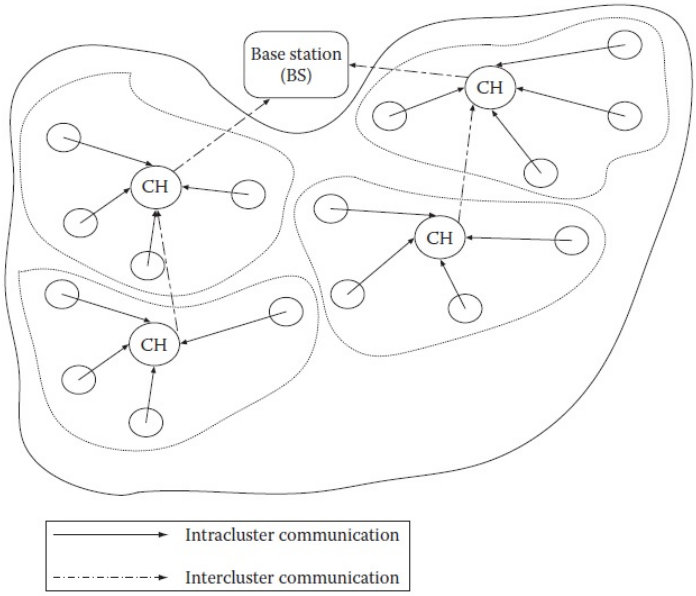


Figure 2.5: Data communication in a WSN [11].

One of the factor that needs to be considered for network and protocol design in WSN is scalability. The number of sensor nodes deployed in the area may be in the order of hundreds or thousands. A WSN must be designed so that the network adapts to the number of sensor nodes. The node density depends on the WSN application. In an indoor environment, the number of sensor nodes could be smaller as compared to an unattended outdoor environment [73].

Fault tolerance is another factor to be considered. Some nodes may fail due to power failure, physical damage or monitored environmental interference. A WSN should be fault tolerant. Fault tolerance depends on the application of WSN. If the sensor nodes are deployed in the home, there should be less consideration for fault tolerance since there should be less active, deliberate interference in an indoor envi-

ronments. On the other hand, if sensor nodes operate in a battlefield for surveillance and detection then there should be proper care for fault tolerance because sensor nodes may be damaged by hostile actions.

Production costs also need to be considered. Since a WSN consists of so many nodes, the production cost of a sensor node should be kept low. Current sensor systems based on bluetooth technology cost about 10$. However, the cost of a sensor node is generally targeted to be less than 1$, which is lower than the current state-of-the-art technology [36].

Network topology is another design factor for a WSN. A huge number of sensor nodes are deployed over the monitored area and for this network topology is a challenging factor. There are three issues related to topology maintenance. One issue is the deployment phase. Sensor nodes can be deployed by dropping from a plane, throwing by a catapult, placing in factories or by placing one by one either by a human or a robot. Post-deployment phase is another issue. After deployment, topology changes due to changes in position, available energy or malfunctioning. Another stage of network topology maintenance is re-deployment of additional nodes. Additional sensor nodes may be deployed for replacing the malfunctioning nodes [54].

Sensor nodes are densely deployed which may be close to each other or may be directly inside of the monitored phenomenon. Sensor nodes may work in busy intersections, in the interior of a large machinery, on the surface of the ocean during a tornado, in a biological or chemical contaminated field, in a home or a large building, in a large warehouse, attached to animals, attached to fast moving vehicles, etc. [62]

Transmission media is another factor to be considered. Communicating nodes are linked by radio, infrared or optical media. For the global operation of WSN, the transmission media should be available worldwide.

Data processing is another design factor for WSN. Energy spent in data processing is much less comparing with the data communication. Data processing in WSNs reduces the number of transmissions which minimizes the overall power consumption. This improves the performance and emergency response speed of the application of WSN like environmental monitoring [33]. For example, in sea water quality monitoring application, number of sensor nodes distributed on the sea surface is large. If every nodes send the data to the base station for processing to find out the abnormality, there should be a huge number of data transmissions which is very energy consuming. Each node able to do some local processing can reduce this number of data transmissions and helps to make the system energy efficient [64].

Target tracking is a challenging, typical and generic application for WSN. The tracking application may be for a single target or multiple targets. Targets may be stationary or moving. Target tracking is applied in various applications of WSN, such as the field of surveillance, finding intruder, military application, and traffic operation. For target tracking application, the protocol design should be energy efficient so that the better energy consumption/tracking performance is achieved [65] [4].

2.2 Task scheduling

In general, task scheduling means to make a plan for performing the tasks to achieve an objective. Where each task has some constraints like completion time, required resources, etc. In computer science, this term basically used in the operating system (OS) concepts. Task scheduling in operating system is the method by which running processes or programs are given access to the system resources, e.g., processor time, memory access, etc. Basically task scheduling is important for multi-tasking in OS. When the scheduler needs to perform multiple tasks in their queue to execute, then task scheduling is required to meet the deadline. Whenever the processor becomes idle, the scheduler selects another task from the queue to perform next [86].

Basic task scheduling concepts can be categorized in to two types: preemptive and non-preemptive task scheduling. The difference between preemptive and non-preemptive scheduling is that in non-preemptive scheduling, a process or a program enters the running state and is not removed from the processor until it is terminated. Preemptive scheduling allows the OS to control over the states of processes.

There are following five states which a process goes through during its life cycle.

- New: When a process is first activated and created. For example, launching a software program in OS.
- Ready: When the process is ready to be assigned to the processor.
- Running: When the process is being executed is known as “running”.
- Waiting:: The process is waiting for the communication from other processes.
- Terminated: When the execution of the process is finished.

Preemptive scheduling is when a process is interrupted and the processor is assign to another process with a higher priority. This type of scheduling occurs when a process switches from running state to a ready state or from a waiting state to ready state. Non-preemptive scheduling allows the process to run through to completion before moving onto the next task [77].

The basic non-preemptive scheduling mechanism is the first-in-first-out (FIFO). In this scheduling mechanism, the first task in the queue should be executed first. There is no priority to select the task from the queue.

Shortest-job-first (SJF) is a preemptive scheduling. In SJF, scheduler picks the shortest job that needs to be done, get it out of the way first and then pick the next smallest one and so on. Here the shortest job means, the job which requires the least processor time or central processing unit (CPU) cycle [72].

Priority scheduling can be either preemptive or non-preemptive. In preemptive priority scheduling, the scheduler preempt the processor if the priority of the newly arrived process is higher than the priority of the currently running process. In non-preemptive priority scheduling, the scheduler simply puts the new process at the

head of the ready queue. Priorities are implemented using integers within a fixed range, but there is no rule as to whether "high" priorities use large numbers or small numbers [88].

Round robin scheduling is another scheduling mechanism. It is similar to FIFO scheduling, except that CPU cycle are assigned with limits called time quantum. When a task is given the CPU, a timer is set to whatever value has been set for time quantum. If the task finishes its cycle before the time quantum timer expires, the scheduler selects the next task from the queue and allocate CPU to it [48].

Task scheduling is an important aspect in WSN. Sensor nodes have no prior knowledge about which task to execute at each time step. Each node performs task scheduling among a set of available tasks. Scheduling is performed at time instances, when the previous task has terminated. Each task requires resources and contributes to the overall performance of the application [20].

Figure 2.6 shows a small homogenous sensor network. Each sensor node needs to perform a particular set of tasks over time steps.

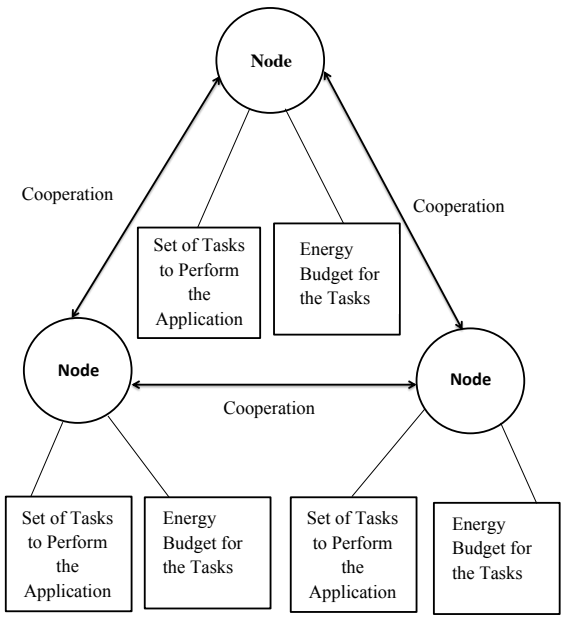


Figure 2.6: Task scheduling for better resource consumption/overall performance trade-off.

Every sensor node has an initial energy budget for performing tasks. Each task consumes an amount of energy budget and provides an impact to the overall performance. Here, cooperation means to share the local observations with the neighboring nodes. Tasks scheduling helps to schedule the tasks in a way that the better resource

consumption/overall performance is maintained.

Task scheduling can be defined as to schedule a set of tasks needed for performing the application in a way that the better resource consumption/application performance trade-off is achieved. For every application of WSN, there should be a set of tasks to be executed by each sensor node. These tasks can be scheduled offline or online [7].

In offline scheduling, the complete information about the system activities is available a priori, and the schedule can be determined at compile time. Due to the high dynamics of WSN, complete system information is only available at runtime which requires online scheduling [52].

If we consider some sensor nodes scattered in a particular area for an object tracking application, the network should work in such a way that it can efficiently allocate its resources for the performance of its tasks. For object tracking the basic tasks are sensing, transmitting, receiving, and sleeping. If we apply task scheduling for object tracking, we need to model the network in such a way that it can maximize the network lifetime.

2.3 Online learning

Online learning takes place in a sequence of rounds. At each round, the learner makes some kind of predictions and then receives some kind of feedback so that training and testing take place at the same time. Online learning is concerned with problems of decision making about the present based on the previously earned experience and knowledge. Online learning is the process of answering questions given knowledge of correct answers of previous questions. For that, the learner needs to make a prediction. This prediction mechanism is based on the mapping from the set of questions to the set of answers is called the hypothesis. After predicting the answer, the learner gets the correct answer of the question. The quality of the answer is then evaluated by a loss function that measures the difference between predicted answer and real answer [10]. Learning algorithms fall in to three groups based on the feedback it has access to. These are supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning, a learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. For example, a learning algorithm is given a problem to decide whether it will rain today or not. For that, the learner was given a set of inputs about the raining probabilities of some days. At first, the learning algorithm needs to predict about an output (rain or not rain) based on the provided inputs. The quality of the prediction or answer of the learner is evaluated by the loss function. The learner's ultimate goal is to minimize the cumulative loss [71].

In an unsupervised learning, the learner makes a decision based on past experiences by trail and error [75]. The learner simply receives inputs but obtains neither

supervised target outputs, nor rewards from the environment. However, it is possible to develop the representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. [28]

The third category of learning is reinforcement learning. This is where the learner receives feedback about the appropriateness of its response. Reinforcement learning is one kind of online machine learning. In reinforcement learning, the learner learns about what to do (actions) and the mapping of situations to actions by the past experiences to maximize the reward over the long run. The learner is not told which action to perform like most of the machine learning mechanisms do. The learner needs to discover which actions provide the most reward by trying the actions. Actions not only provide an impact on the situation of the environment but also on obtained reward [38].

One of the challenges of reinforcement learning is exploration and exploitation of the actions. In reinforcement learning, the learner tries out all actions in it's actions set. Exploitation means to perform the action based on current information. The learner select the action which has been performed in the past, and found to be effective in producing reward. Exploration means to perform a new action which has not been performed before. The learner needs to exploit actions in order to obtain reward, but the learner also needs to explore the actions in order to make better action selection in future [80].

A reinforcement learner generally has four basic components: a policy, a reward function, a value function, and a model of the environment. The policy is the decision making function of the agent. It specifies what action it should perform in which state it remains. The reward function maps the state of the environment to a single number indicating the gain of the state. The objective of the agent is to maximize the total reward over long run. The value function defines what is good in the long run. The model of the environment or external world should follow the behavior of the environment. For a given situation and action, the model predicts the resultant next state and next reward. Figure 2.7 shows the basic components of reinforcement learning.

Due to the dynamic nature of WSN, it is necessary to schedule the tasks online [16]. We need a task scheduling mechanism that improves application performance. For example, for object tracking applications, sensor nodes need to perform some tasks over some time period. Sensing, transmitting, receiving, and sleeping are some of the common tasks performed by the sensor nodes. If sensor nodes have nothing to transmit/receive or sense then its better to turn off all the components. So, a task scheduling mechanism is required which is online and able to improve the overall performance of the network. If we are able to learn efficiently and online about the tasks to perform for a particular application then there should be better performance in terms of the resource consumption/performance trade-off [27].

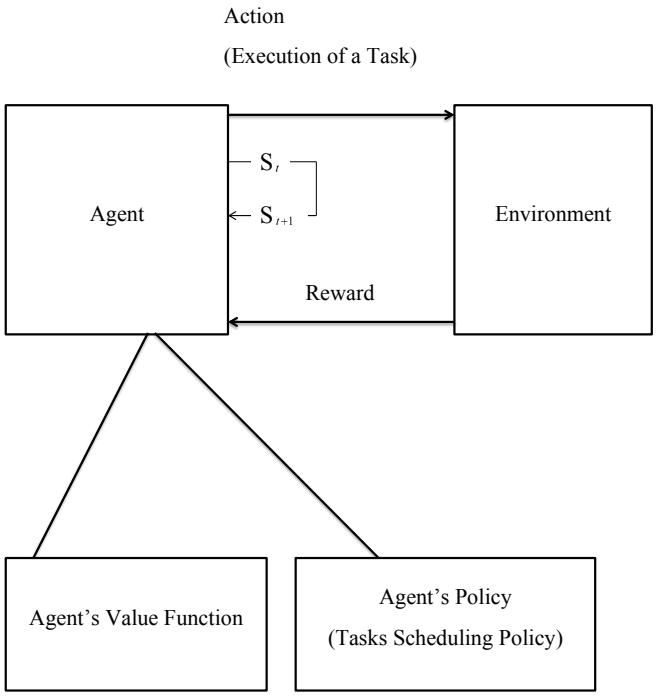


Figure 2.7: Basic components of a reinforcement learning.

2.4 Related work for task scheduling in WSN

In a resource constrained WSN, effective task scheduling is very important for facilitating the effective use of resources [26]. Cooperative behavior among sensor nodes can be very helpful to schedule the tasks in a way that the energy is optimized and also a considerable performance is maintained. Most of the existing methods of tasks scheduling do not provide online scheduling of tasks. Most of them rather consider static task allocation instead of focusing on distributed task scheduling. The main difference between task allocation and distributed task scheduling is that task allocation deals with the problem of finding a set of task assignments on a sensor network that minimizes an objective function such as total execution time [81]. On the other hand, in a task scheduling problem, the objective is to determine the best order of tasks execution for each sensor node. Each sensor node has to execute a particular task at each time step in order to perform the application, and each node determines the next task to execute based on the observed application behavior and available resources. The following subsections describe some task scheduling methods in WSN.

2.4.1 Reinforcement learning method

Reinforcement learning helps to enable applications with inherent support for efficient resource/task management. It is the process by which an agent improves task scheduling according to previously learned behavior. It does not need a model of its environment and can be used online. It is simple, and demands minimal computational resources.

Shah et al. [66] consider Q learning as reinforcement learning for the task management. They describe a distributed independent reinforcement learning approach (DIRL) for resource management, which forms an important component of any application includes initial sensor selection and task allocation as well as run-time adaptation of allocated resources to tasks. Here the optimization parameters are energy, bandwidth, network lifetime, etc. DIRL allows each individual sensor node to self schedule its tasks and allocate its resources by learning their usefulness in any given state while honoring application-defined constraints and maximizing the total amount of reward over time. The advantage of using independent learning is that no communication is required for coordination between sensor nodes and each node selfishly tries to maximize its own rewards. They apply this to the object tracking application and achieve better results in terms of the average award over time in the network.

Shah et al. [67] presents a scheme for resource management in WSN using a bottom up approach where each sensor node is responsible for task selection instead of the top down approach conventionally used by other solutions. This bottom up approach using reinforcement learning allows development of autonomous WSN applications with real time adaptation, minimal or no centralized processing requirement for task allocation, and minimal communication overhead. They use two tier learning: micro learning and macro learning, as used by each data stream sub-world to steer the system towards application goal by setting/updating rewards for micro learners. They use collective intelligence (COIN) theory to enable macro learning that can steer the system towards the application's global goal.

2.4.2 Evolutionary-based method

Guo et al. [31] propose an evolutionary based method for the task allocation problem in WSN. They consider a WSN consists of m sensors and n independent task. Tasks compete for the sensors. The goal is to allocate n tasks to the sensors reasonably with the shortest total execution time. They propose a basic particle swarm optimization (PSO) which is initialized with a population of random solutions. They consider a set of particles. Each particle is treated as a point with a velocity in a D dimensional solution space. Each particle has a fitness value which is set by an objective function. After calculating the fitness value, the system updates the local best value and then global best value. For each particle, the system executes the mutation operator on this particle if its diversity is less than the threshold. After that, the system executes

the mutation operator on the all particles if the population diversity is less than a particular threshold. If the termination conditions are satisfied, then the method terminates.

2.4.3 Rule-based method

A set of rules or predicates help to configure the wireless sensor networks, where certain functions must be automatically assigned to sensor nodes, such that the properties of a sensor node (e.g. remaining energy, network neighbors) match the requirements of the assigned function. Based on the assigned rules, sensor nodes may adapt their behavior accordingly, establish cooperation with other nodes, or may even download specific code for the selected rule.

Frank et al. [26] propose a method for generic task allocation in wireless sensor networks. They define some rules for the task execution and propose a role-rule model for sensor networks where "role" is used as a synonym for task. It is a programming abstraction of the role-rule model. This distributed approach provides a specification that defines possible roles and rules for how to assign roles to nodes. This specification is distributed to the whole network via a gateway or alternatively it can be pre-installed on the nodes. A role assignment method takes into account the rules and node properties, which may trigger execution and in network data aggregation. This generic role assignment approach does consider the energy consumption but not the ordering of tasks to sensor nodes.

Liu et al. [55] mention a system for managing autonomic, parallel sensor systems. They consider energy a resource. They describe Impala, a middleware architecture that enables application modularity, adaptability, and reparability in WSN. Impala allows software updates to be received via each node's wireless transceiver and to be applied dynamically to the running system. In addition, Impala also provides an interface for on-the-fly application adaptation in order to improve the performance, energy efficiency, and reliability of the software system. They consider the application parameters include recent histories, averages, or totals of the number of direct network neighbors encountered the amount of sensor data successfully transferred to peer sensor nodes, the amount of free storage for application data, and so on. System parameters include the battery level, the transmitter range, and the geographic position of the nodes. They design the middleware considering some rules on parameters. If the rules are satisfied then the next query is executed. This is designed to be a part of the Zebranet mobile sensor network. Impala is a lightweight run-time system that can greatly improve the system reliability, performance and energy efficiency.

2.4.4 Constraint satisfaction method

Constraint satisfaction is a formalism that is used to model a large class of problems with applications in engineering design, planning, scheduling, resource allocation,

and fault diagnosis [19]. There are a number of variables, each of which has an associated domain of values in a constraint satisfaction problem (CSP). Constraints are specified on subsets of these variables restricting the set of values they can take on jointly. The objective of a CSP is to find out if each of these variables can be assigned a value from its domain in such a way that all the constraints are satisfied. In a CSP it suffices to find a single point in the search space which satisfies all constraints. A CSP is said to be satisfiable if there exists such a point and unsatisfiable otherwise.

Krishnamachari et al. [53] examine channel utilization as a resource management problem by constraint satisfaction. They consider a wireless sensor network of n nodes placed randomly in a square area with a uniform, independent distribution. This work tests three self configuration tasks in wireless sensor networks: partition into coordinating cliques, formation of Hamiltonian cycles and conflict free channel scheduling. They explore the impact of varying the transmission radius on the solvability and complexity of these problems. In the case of partition into cliques and Hamiltonian cycle formation, they observe that the probability that these tasks to be performed undergoes a transition from zero to one. Almost every network graph is generated by random location of the nodes satisfies the desired global property. In these cases, the critical transmission range corresponds to an energy efficient operating point. However, in the third task (conflict free channel scheduling) they observe that the transition occurs in reverse; there is a critical transmission range below which almost all network graphs generated by the random location of nodes can be allocated the available number of channels, and beyond which the desired property is rarely satisfied.

Kogekar et al. [51] consider energy or power saving as a resource management problem modeled by dynamic software reconfiguration. It is a constraint guided approach for dynamic reconfiguration in WSN. Reconfiguration is performed by transitioning from one point of the operation space to another based on the constraints. System requirements are expressed as formal constraints on operational parameters such as power consumption, latency, accuracy, and other QoS properties that are measured at run-time. In this work, once a new configuration that satisfies all the constraints are found, the reconfiguration can be accomplished by online software synthesis targeting an interpreted language or a command interface. The reconfigurator is responsible for performing the necessary local application changes upon notification from the base station. Here each mote has two components: *monitor* and $reconfigurator$. *monitor* observes the existing resources such as remaining battery power. The base station has four components: $GlobalConstraintMonitor$, $GRATISPlus$, $DESERT$ and $GRATIS$. The monitor of each mote reports its remaining battery power to global constraint monitor of the base station whose has a database to check the energy level of that mote is sufficient to keep the existence configuration. If the energy level is below a certain threshold then it reports the parameters to $GRATISPlus$ which sends the design space representation as an input to the $DESERT$. $DESERT$ configures the design and sends it to the $GRATIS$, and finally the $reconfigurator$ of a mote gets direction about configuration from

GRATIS. They implement it to track the movement of people across an aisle.

2.4.5 Market-based method

An auction is a market mechanism for allocating resources. The essence of an auction is a game, where the players are the bidders, the strategies are the bids and both allocation and payments are functions of the bids. One well known auction is the Vickery-Clarke-Groves (VCG) auction [49], which requires gathering global information from the network and performing centralized computations.

Huang et al. [14] describe the signal to noise ratio (SNR) auction and the power auction that determine the relay selection and relay power allocation in a distributed fashion. Here the network rate (bps/Hz) is considered as a resource. The proposed auction-based resource allocation methods can be generalized to networks with multiple relays. Each relay announces a price and a reserve bid, without knowing the prices and reserve bids of other relays of others relays. Each user submits a non-negative bid vector, one component for each relay. Based on the bids, the relay allocates to the user transmission power. The Nash equilibrium [59] is achieved in a distributed fashion if it allows the users to iteratively update their bids based on best response functions in an asynchronous fashion. They implement this on a single relay network.

Chen et al. [35] address the problem of providing congestion-management for wireless sensor networks executing several target tracking applications. They consider information or data as resource. They employ auctions to provide two different utility loss management solutions: equalizing utility loss across all tracking applications and minimizing the total utility loss of applications. Any time a given node has multiple target report packets for different applications to be forwarded for its current transmission slot, the node conducts an auction to assign the slot to the packet with the highest bid. Hence, the bidders represent packets awaiting transmission and their bids are defined by the predicted information utility loss of the applications to which packets are sent. This in turn prevents resource starvation of any auction participant.

2.4.6 Utility-based method

The utility-based technique is a valuable tool for resource management in wireless sensor networks (WSN). The benefits of this technique include the ability to take application utilities into account for performing the application.

Dhanani et al. [21] give a comparison of utility-based information management policies in sensor networks. Here the resource is information or data. They consider two models: sensor centric utility-based (SCUB) and resource manager (RM) models. SCUB is a distributed approach that instructs individual sensors to make their own decisions about what sensor information should be reported, based on a utility model for data. RM is a consolidated approach that takes into account knowledge

from all sensors before making decisions. They evaluate these policies through simulation in the context of dynamically deployed sensor networks in military scenarios. Both SCUB and RM can extend the lifetime of a network comparison to the network which is not maintaining any policy. Without any policy, the network can send more information than SCUB and RM but it has less network lifetime compared to these. RM networks last longer than SCUB according to their experiment.

Byers et al. [13] describe utility-based decision making in wireless sensor network. They consider application domains in which a large number of distributed, networked sensors must perform a sensing task repeatedly over time. They present a model for such application in which they define appropriate global objectives based on utility functions and specify a cost model for energy consumption. They derive the utility from a consumer of sensor network resources by a monotone function $U : S \rightarrow [0, 1]$, which for a network graph $G = (V, E)$ maps the sensing subset, the set of all nodes in the graph that are sensing, to a real valued interval. In a best effort service model, nodes attempt to optimize the utilization of resources in the present without regard to future cost. With the goal of optimizing the total utility derived over the lifetime of the network, the model enables nodes to discount current gain to optimize their consumption of energy over time

2.5 Difference to own approach

The following is a brief overview of the differences between the state-of-the-art and our own approach.

- Most of the existing methods [31], [26], [55], [51], [21] of task scheduling in WSNs do not provide online scheduling of tasks. They mainly consider static task allocation instead of focusing on task scheduling. The main difference between task allocation and task scheduling is that task allocation deals with the problem of determining a set of task assignments on a sensor network that minimizes an objective function such as the total execution time [81]. On the other hand, the objective of task scheduling is to determine the best temporal order of tasks for each sensor node. In offline scheduling, the complete information about the system activities are available a priori, and the schedule can be determined at compile time. Due to the high dynamics of WSN, complete system information is only available at runtime and so online scheduling is required [52]. We propose online learning methods for task scheduling.

- Most of the existing methods of task scheduling in WSNs [29], [53], [21] neither consider distributed tasks scheduling nor the trade-off among resource consumption and performance. They also do not explicitly consider energy consumption. We consider the trade-off among resource consumption and performance.

- Shah et al. [66] introduce a task scheduling approach for WSN based on an independent reinforcement learning method (RL) for online task scheduling. They use Q learning [85] for the task 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 [66], 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.

- Most of the existing methods do not evaluate task scheduling methods. We evaluate three online task scheduling methods: Reinforcement Learning (RL), Cooperative Reinforcement Learning (CRL), and Exponential Weight for Exploration and Exploitation (Exp3). Each of these three are evaluated for task scheduling in a target tracking application, and analyzed in terms of their performance (a trade-off between tracking quality and energy consumption). Our proposed approach also takes cooperation into consideration, where each node sharing local observations with its neighbors.

CHAPTER

3 System model

In this chapter, we describe the system model of our task scheduling methods. We apply reinforcement learning (RL), cooperative reinforcement learning (CRL), exponential weight for exploration and exploitation (Exp3) bandit solvers, and combinatorial auction for the task scheduling in a WSN. We consider a target tracking application and apply our methods for task scheduling. First, we delineate the basic system model for our task scheduling followed by the system model for combinatorial auction method. Afterward, we describe our considered set of actions, set of states, and the reward function.

3.1 Formal problem definition

Before describing the problem formally, terms like resource consumption, and tracking quality need to be defined. In WSNs, resource consumption happens due to perform the various tasks needed for the application. Each task consumes an amount of energy from the fixed energy budget of the sensor nodes. Typically, tracking quality in a target tracking application of a WSN is defined as the accuracy of target location estimation provided by the network.

In our approach the WSN is composed by N nodes represented by the set $\hat{N} = \{n_1, \ldots, n_N\}$. Each node has a known position (u_i, v_i) and a given sensing coverage range which is simply modeled by circle with radius r_i. All nodes within the communication range R_i can directly communicate with n_i and are referred to as neighbors. The number of neighbors of n_i is given as $ngh(n_i)$. The available resources of node n_i is modeled by a scalar E_i. We consider the battery power of sensor nodes as resource. We consider a set of tasks to perform over time steps. Each task consumes some battery power from the energy budget of the sensor nodes. We consider a set of static values for the energy consumption of tasks. These values are assigned based on the energy demands of the task. We set higher value for the tasks which need higher energy consumption.

The WSN application is composed by A tasks (or actions) represented by the

set $\hat{A} = \{a_1, \ldots, a_A\}$. 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_i requires some resources $\tilde{E}_j$ and contributes to the overall application performance P. Thus, the execution of task a_j on node n_i is only feasible if $E_i \geq \tilde{E}_j$. 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. We consider non-preemptive scheduling based on our proposed model. Figure 3.1 shows our considered WSN model components.

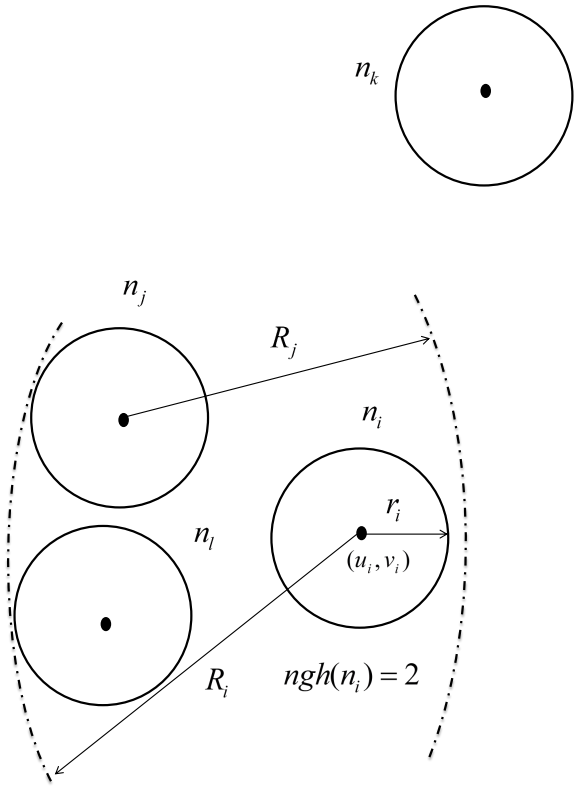


Figure 3.1: WSN model components. Here four nodes n_i, n_j, n_k, and n_l. R_i is the communication range, r_i is the sensing range, and (u_i, v_i) is the position of the node n_i. Number of neighbors of the node n_i, $ngh(n_i) = 2$.

We demonstrate our task scheduling approach using a target tracking application. We consider a sensor network which may consists of a variable 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 all time is energy demanding. Thus, task should only be executed when necessary and sufficient for tracking performance. Sensor nodes can cooperate with each other by informing neighboring nodes about "approaching" targets. Neighboring nodes can therefore become aware of approaching targets.

We define the objective function in a way that we can trade the application performance and the required energy consumption by a balancing factor. 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 resource consumption is minimized. Naturally these are conflicting optimization criteria, so there is no single best solution. The set of non-dominating solutions for such a multi-criteria problem can be typically represented by a Pareto front.

3.2 System model

We apply reinforcement learning methods (cooperative Q learning, $SARSA(\lambda)$), bandit solvers (Exp3), and a combinatorial auction method for task scheduling. First, we describe the basic system model for reinforcement learning methods, and bandit solvers then we describe the system model for the combinatorial auction method. Since reinforcement learning methods, and bandit solvers both consider a set of tasks, a set of states, and a reward for the executed actions, the system model is same for both methods. On the other hand, in combinatorial auction method, there should be some consideration of bids, auctioneer, and revenues. Due to the different components in the system, we consider two different system models.

3.2.1 Basic system model

The task scheduler operates in a highly dynamic environment, and the effect of the task ordering on the overall application performance is difficult to model. We therefore apply reinforcement learning methods, and bandit solvers to determine the "best" task order given the experiences made so far. Figure 3.2 depicts our proposed system model can be described as follows:

- *Agent:* Each sensor node embeds an agent which is responsible for executing the online learning algorithm.
- *Environment:* The WSN application represents the environment in our approach. Interaction between the agent and the environment is achieved by executing actions and receiving a reward function.
- *Action:* An agent's action is the currently executed application task on the sensor node. At the end of each time period t_i each node triggers the scheduler to determine the next action to execute.

- *State:* A state describes an internal abstraction of the application which is typically specified by some system parameters. In our target tracking application, the states are represented by the number of currently detected targets in the node's FOV, and expected arrival times of targets detected by neighboring nodes. The state transitions depend on the current state and action.

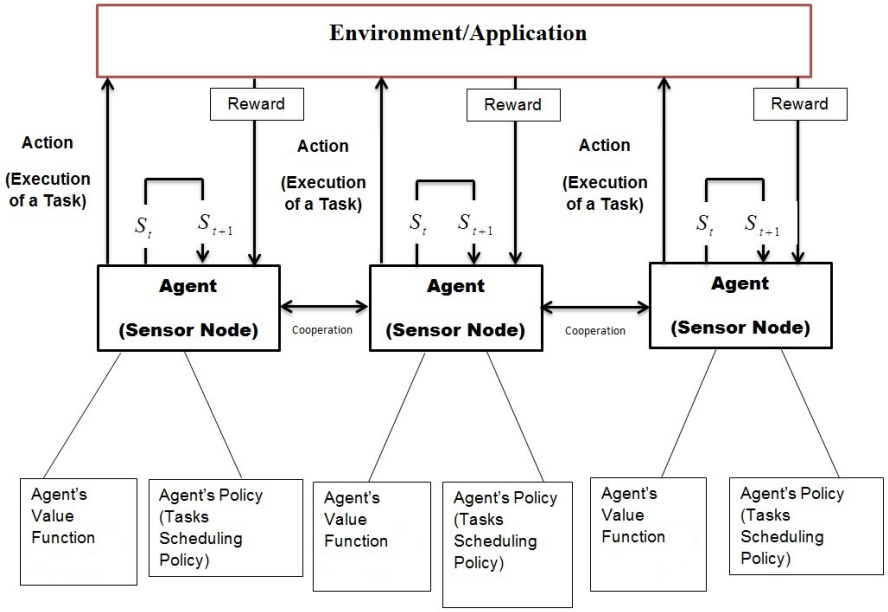


Figure 3.2: Proposed system model.

- *Policy:* An agent's policy determines what action will be selected in a particular state. In our case, this policy determines which task to execute at the perceived state. The policy can focus more on exploration or exploitation depending on the selected setting of the learning algorithm.

- *Value function:* This function defines what is good for an agent over the long run. It is built upon the reward function values over time and hence its quality totally depends on the reward function [66].

- *Reward function:* This function provides a mapping of the agent's state and the corresponding action to a reward value that contributes to the performance. We apply a weighted reward function which is capable of expressing the trade-off between energy consumption and tracking performance.

- *Cooperation:* We consider the information exchange among neighboring nodes as cooperation. The received information may influence the application's state of a sensor nodes.

3.2.2 System model for combinatorial auction

Figure 3.3 shows our proposed system model for combinatorial auction. The key components can be described as follows:

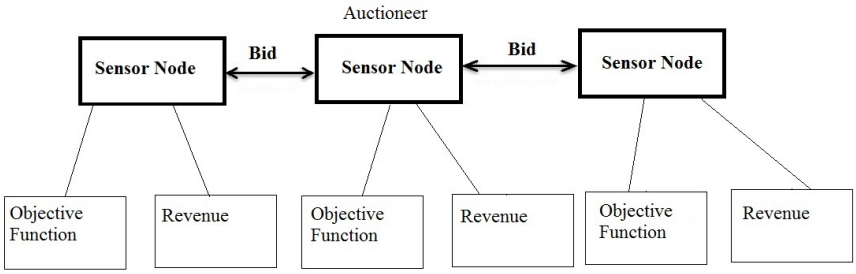


Figure 3.3: System model for the combinatorial auction.

- *Auctioneer:* Sensor nodes act as auctioneer initiate auction. The neighboring nodes bid to the auctioneer.
- *Bid:* The bid is composed of some items as this is a combinatorial auction mechanism. These items are required CPU cycle, energy consumption, and ideal time gap (cp. Section 4.1 for more details).
- *Objective function:* The neighboring nodes calculate the objective function which consists of bids. As we apply combinatorial auction method, we consider multiple resources like computation cost, remaining energy, and ideal gap (cp. Section 4.1 for more details). The goal is to maximize the objective function.
- *Revenue:* Revenue is defined by the difference between value the sensor node receives for the items, and the price it pay for those items.

3.3 Set of actions

For the application mapping to the proposed model, we need to consider a set of actions. An agent performs action to perform the application and get reward for that. We consider two different set of actions. In our initial works on resource-aware task scheduling in WSN, where we consider a small network (a simple and fixed network topology consisting of three nodes), we consider only four actions. We apply combinatorial auction method considering this simple network and without considering the actions like prediction of trajectories, intersection of trajectories, etc.

We consider the following set of actions for the combinatorial auction-based method.

- Sensing

- Transmit
- Receive
- Sleeping

We consider the following actions for a target tracking application using reinforcement learning methods, and bandit solvers (cp. Subsection 4.3.3 for more details).

- Detect_Targets
- Track_Targets
- Send_Message
- Predict_Trajectory
- Goto_Sleep
- Intersect_Trajectory

3.4 Set of states

Another component for application mapping is a set of states. Agents execute an action for performing the application and shift from one state to another. States are defined by application variables. We consider some application variables to define the state. These application variables can change for executing a particular action. As the varibales change, agent can shift from one state to another after executing the task. We consider the following variables to represent the states for the reinforcement learning method, $SARSA(\lambda)$, and bandit solvers (cp. Subsection 4.3.3 for more details).

- Currently detected targets.
- List of expected arrival times of targets.

Based on these variables, we abstract the target tracking application by three states (cp. Subsection 4.3.3 for more details).

- Idle: This state denotes that there is currently no target detected within the node's FOV and the local clock is too far from the expected arrival time of any target already detected by some neighbors.
- Awareness: There is currently no detected target in the node's FOV in this state. However, the expected arrival time of at least one target is less than a threshold value.

- Tracking: This state indicates that there is currently at least one detected target within the node's FOV.

We consider the following set of variables for the cooperative Q learning.

- Currently detected targets.
- Minimum energy required for actions.
- Data to transmit.

3.5 Reward function

The reward function is another component for application mapping. For performing each action, sensor nodes receive a reward. This reward helps to take decision in future that which task is suitable to execute at each time step.

In our proposed reinforcement learning methods, and bandit solvers, we consider a weighted reward function which balances between performance and resource consumption. Performance is measured by the ratio of the number of tracked positions of the target inside the FOV of the node to the number of all possible detected target's positions in the FOV. Resource consumption is defined as the energy consumption at each time step for executing the tasks. We consider a balancing factor which balances the trade-off between energy consumption and tracking quality (cp. Subsection 4.3.3 for more details).

For cooperative Q learning methods, we consider scalar values for the rewards of each action. Higher scalar values are set to the tasks of lower energy requirement (cp. Subsection 4.3.2 for more details).

CHAPTER

4 Task scheduling methods

In this chapter, we describe the proposed resource-aware task scheduling methods in a WSN. Here, we consider the system model, the set of tasks, the set of states, and the reward function described in Chapter 3. First, we describe the combinatorial auction method. Then we describe the cooperative reinforcement learning methods (Q learning, $SARSA(\lambda)$), and the bandit solvers method. We apply our methods in a target tracking application. We also describe the task scheduling in a target tracking application using our proposed methods in this chapter.

4.1 Combinatorial auction

We consider a market-based approach for the task scheduling in WSN. We use combinatorial auction-based method here as our bid is composed of some items like central processing unit (CPU) cycle, energy requirement, and ideal time gap. Target tracking is considered as application here. We consider two application variables to define the states of the system which are field of view (FOV) and data to transmit (DTT). Both of these variables can take the value either 1 or 0. For example, when DTT=1 then there is data to transmit. When two application variables FOV and DTT of a particular node i become 1, then the node acts as an auctioneer. We consider four tasks sensing, transmit, receive, and sleeping for the target tracking application. Each task needs some items, e.g., CPU cycle, energy, and ideal time gap to execute. The targets are sold in this auction. The auctioneer initiates an auction. The neighboring nodes send bid to the auctioneer and calculate the objective function to win.

Objective function is as follows

$$Objective(j) = \alpha S^j + \frac{\beta}{\eta^j} + \frac{\gamma}{D^j} \tag{4.1}$$

where the node j has the signal strength for detecting S^j, the node with higher signal strength carries more information about the target. The resource price η^j,

the node with lower resource price should be given priority as the node can perform a task with lower resource price can maintain the resource efficiency in the system and the term D^j defines the distance between the target and the node. α, β and γ are equilibrium constants. The resource price η^j in Equation 4.1 can be calculated by the following parameters.

- Required CPU cycle (R_{CPU}): It refers to the expected CPU cycle required for accomplishing the task.
- Available CPU: It is sensor node's CPU clock frequency.
- Computation Cost: It can be calculated by the following equation that is used in [23].

$$CompCost(R_{CPU}, f) = NCV_{dd}^2 + V_{dd}(I_0 e^{\frac{V_{dd}}{nV_T}})(\frac{R_{CPU}}{f}) \tag{4.2}$$

$$f \cong K(V_{dd} - c) \tag{4.3}$$

  where V_T is the thermal voltage, and C, I_0, n, K, c are processor dependent parameters.

- Remaining energy (E): It refers to the remaining energy of the sensor node.
- Ideal gap (T_{IG}): It is a gap between the time that sensor nodes accomplish the task and the time that sensor nodes communicate the tasks output to its neighbor. If the ideal gap is very large, the incentive for the task would rather decrease.

The resource price is calculated by the following equation

$$\eta^j = (\frac{CompCost}{E}) \times exp(T_{IG}) \tag{4.4}$$

where $CompCost$ refers to computation cost, E refers to remaining energy and T_{IG} means ideal gap. After calculating the resource price, the sensor nodes will bid for the object to track it.

Now we need an algorithm that will find out the winner and the bids that will maximize the revenue of each agent as we consider the sensor nodes here as agents in a multi-agent environment. We use the combinatorial auction named Progressive Adaptive User Selection Environment (PAUSE) auction method [84].

The agents maintain a set B of the current best bids, each consists of a set of items. The agents also maintain price of the items agent i should pay b^{value}, value of the bids $v_i(b^{items})$, and value for the items of the bids b^{items}. There are two variables g^* and u^*. g^* is the revenue of the agent and u^* is the winner bid values. There are two other variables *my_bids* and *their_bids* to track the bid values. At any point in

Algorithm 1 PAUSEBID (i)

Inputs: Price of the items agent i should pay b^{value}, Value of the bids $v_i(b^{items})$, Value for the items of the bids b^{items}
Output: Revenue of the agent g^*, Winner bid values W
Initialization: $my\_bids \leftarrow \phi$, $their\_bids \leftarrow \phi$, $g^* \leftarrow \phi$, $u^* \leftarrow (W)$
for $b \in B$ **do**
    if $b^{agent} = i$ or $v_i(b^{items}) > b^{value}$ **then**
        Set $my\_bids \leftarrow my\_bids + new\_bid(i, b^{items}, v_i(b^{items}))$
    else
        $their\_bids \leftarrow their\_bids + b$
    end if
    for $S \in$ subsets of fewer items such that $v_i(S) > 0$ **do**
        Set $my\_bids \leftarrow my\_bids + new\_bid(i, S, v_i(S))$
        $bids \leftarrow my\_bids + their\_bids$
        MAXIMUM$(bids, \phi)$
        $u^* \leftarrow (W)$
        $surplus \leftarrow \sum_{b^{agent}=i} W(b^{items}) - b^{value}$
        if $surplus = 0$ **then**
            return g^*
        end if
        $my\_payment \leftarrow v_i(b^{items}) - b^{value}$
        for $b \in u^* | b^{agent} = i$ **do**
            if $my\_payment \leq 0$ **then**
                $g^* \leftarrow 0$
            else
                $g^* \leftarrow W(b^{items}) + my\_payment.\frac{W(b^{items}) - b^{value}}{surplus}$
            end if
        end for
    end for
end for
return g^*
MAXIMUM$(bids, g)$
for each iteration **do**
    Sort the bids in increasing order
    $N = bid_1, bid_2, ..., bid_n$
    for $i = 1$ to n **do**
        Set $g \leftarrow$ the list of bids in increasing order.
    end for
end for

the auction, there will also be a set $W \subseteq B$ of currently winning bids. This is the set of bids that currently maximizes the revenue, where the revenue of W is given as follows

$$g^* \leftarrow W(b^{items}) + my\_payment. \frac{W(b^{items}) - b^{value}}{surplus} \tag{4.5}$$

where $W(b^{items})$ is the value of the winning bids, $my\_payment$ is defined by the value it receives for the items minus the price it must pay for those items, $my\_payment \leftarrow v_i(b^{items}) - b^{value}$. $surplus$ is defined by the difference of the winning bid values minus the price it should pay for those items. The goal of the bidding agents in the PAUSE auction is to maximize their revenue.

Given an agent's value function and current set of winning bids W, we can calculate the agent's utility from W as

$$u_i(W) = \sum_{b \in W | b^{agent} = i} V_i(b^{items} - b^{value}) \tag{4.6}$$

That is, the agents utility for a bid set W is the value it receives for the items it wins in W minus the price it must pay for those items. If the agent is not winning any items then its utility is zero. The goal of the bidding agents in the PAUSE auction is to maximize their utility.

4.2 Learning methods

We need suitable methods for learning resource-aware task scheduling. Learning methods help to learn the next suitable task to execute based on the past observed behavior. The learning methods could be offline or online. Due to the dynamic nature of a WSN, we do not have any a priori information about the scheduling of tasks. So, our target is to learn the scheduling online. Reinforcement learning methods, bandit solvers, are online learning methods for task scheduling. By these methods, it is possible to learn the best task to execute from the past experiences which provide more reward. We choose reinforcement learning methods and bandit solvers for learning task scheduling online and to get the better trade-off between resource consumption and performance.

4.2.1 Reinforcement learning

Reinforcement learning is a branch of machine learning and is concerned with determining an optimal policy (cp. Section 2.3 for more details). It maps the states of the environment to the actions that an agent should take in those states so as to maximize a numerical reward over time. We propose cooperative Q learning for task scheduling. Here this is cooperative in a sense that each sensor node sends the value function to the neighbors. We also consider a weight factor [89] for the neighbors. As our aim is to learn task scheduling in a cooperative manner we choose cooperative

Q learning. We propose cooperative state-action-reward-state-action, $SARSA(\lambda)$ learning algorithm. Here we also consider the cooperation among neighboring nodes with the local observations. We also consider the weight factor for the neighboring nodes. Here λ is the learning parameter which helps for guaranteed convergence learning [80].

Cooperative Q learning for task scheduling

Cooperative Q learning is a reinforcement learning approach to learn the usefulness of some tasks over time in a particular environment. We consider the wireless sensor network as a multi-agent system. The nodes correspond to agents in the multi-agent reinforcement learning. The world surrounding the sensor nodes forms the environment. Tasks are considered as activities for the sensor nodes at each time step such as transmit, receive, sleep, sense, etc. States are formed by set of system variables such as object in the FOV of sensor nodes, required energy for a specific action, data to transmit, etc. A reward value provides some positive or negative feedback for performing a task at each time step. Value functions define what is good for an agent over long run described by reward function and some parameters. In cooperative Q learning every agent needs to maintain a Q matrix for the value functions like independent Q learning. Initially all entries of the Q matrix are zero and the nodes or agents may be in any state. Based on the application defined variable or system variables, the system goes to a particular state. Then it performs an action which depends on the status of the nodes (Example: For transmit action, a node must have residual energy which is greater than transmission cost). It calculates the Q value for this (state, task) pair with the immediate reward.

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha(r_{t+1}(s_{t+1}) + \gamma \sum f V_t(s_{t+1})) \tag{4.7}$$

$$V_{t+1}(s_t) = \max_{a \in A} Q_{t+1}(s_t, a) \tag{4.8}$$

where $Q_{t+1}(s_t, a_t)$ means the update of Q value at time $t+1$, after executing the action a at time step t. r_{t+1} means the immediate reward after executing the action a at time t. V_t is the value function at time t. V_{t+1} is the value function at time $t+1$. $\max_{a \in A} Q_{t+1}(s_t, a)$ means the maximum Q value after performing an action from the action set A. γ is the discount factor which can be set to a value in $[0, 1]$. The higher the value, the greater the agent relies on future reward than the immediate reward. α is the learning rate parameter which can be set to a value in $[0, 1]$. It controls the rate at which an agent tries to learn by giving more or less weight to the previously learned utility value. When α is set close to 1, the agent gives more priority to the previously learned utility value.

f is the weight factor [89] for the neighbors of agent i and can be defined as follows

$$f = \frac{1}{ngh(n_i)} \quad \text{if } ngh(n_i) \neq 0 \tag{4.9}$$

$$f = 1 \quad \text{otherwise.} \tag{4.10}$$

The algorithm can be stated as follows:

Algorithm 2 Q learning for task scheduling.

1: Initialize $Q(s, a) = 0$. Where s is the set of states and a is the set of actions
2: **while** Residual energy is not equal to zero **do**
3: Determine current state s by application variables
4: Select an action a which has the highest Q value
5: Execute the selected action
6: Calculate Q value for the executed action (Eq. 4.7)
7: Calculate the value function for the executed action (Eq. 4.8)
8: Send the value function to the neighbors
9: Shift to next state based on the executed action
10: **end while**

Cooperative $SARSA(\lambda)$ learning for task scheduling

$SARSA(\lambda)$ [80], also referred to as state-action-reward-state-action, is an iterative algorithm that approximates the optimal solution without knowledge of the transition probabilities which is very important for a dynamic system like WSN. At each state s_{t+1} of iteration $t + 1$, it updates $Q_{t+1}(s, a)$, which is an estimate of the Q function by computing the estimation error δ_t after receiving the reward in the previous iteration. The $SARSA(\lambda)$ algorithm has the following updating rule for the Q values

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s, a) + \alpha\delta_t e_t(s_t, a_t) \tag{4.11}$$

for all s,a.

In Equation 4.11, $\alpha \in [0, 1]$ is the learning rate which decreases with time. δ_t is the temporal difference error which is calculated by following rule

$$\delta_t = r_{t+1} + \gamma_1 f Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{4.12}$$

In Equation 4.12, γ_1 is a discount-factor which varies from 0 to 1. The higher the value, the more the agent relies on future rewards than on the immediate reward. r_{t+1} represents the reward received for performing action. f is the weight factor [89] for the neighbors of agent i and can be defined as follows

$$f = \frac{1}{ngh(n_i)} \quad \text{if } ngh(n_i) \neq 0 \tag{4.13}$$

$$f = 1 \quad \text{otherwise.} \tag{4.14}$$

An important aspect of an RL-framework is the tradeoff between exploration and exploitation [13]. Exploration deals with randomly selecting actions which may not have higher utility in search of better rewarding actions, while exploitation aims at the learned utility to maximize the agent's reward.

In our proposed algorithm, we use a simple heuristic where exploration probability at any point of time is given by

$$\epsilon = \min(\epsilon_{max}, \epsilon_{min} + k * (S_{max} - S)/S_{max}) \tag{4.15}$$

where ϵ_{max} and ϵ_{min} define upper and lower boundaries for the exploration factor, respectively. S_{max} represents maximum number of states which is three in our work and S represents current number of states already known. At each time step, the system calculates ϵ and generates a random number in the interval of $[0, 1]$. If the selected random number is less than or equal to ϵ, the system chooses a uniformly random task (exploration) otherwise it chooses the best task using Q values (exploitation).

$SARSA(\lambda)$ improves learning through eligibility traces. $e_t(s, a)$ is the eligibility traces in Equation 4.11. Here λ is another learning parameter similar to α for guaranteed convergence. γ_2 is the discount factor. In general, eligibility traces give a higher update factor for recently revisited states. This means that the the eligibility trace for a state-action pair (s, a) will be reinforced if $s_t \in s$ and $a_t \in a$. Otherwise, if the previous action a_t is not greedy, the eligibility trace is cleared.

The eligibility trace is updated by the following rule

$$e_t(s_t, a_t) = \gamma_2 \lambda e_{t-1}(s_t, a_t) + 1 \quad \text{if } s_t \in s \text{ and } a_t \in a \tag{4.16}$$

$$e_t(s_t, a_t) = \gamma_2 \lambda e_{t-1}(s_t, a_t) \qquad \text{otherwise.} \tag{4.17}$$

The algorithm can be stated as follows:

Algorithm 3 $SARSA(\lambda)$ learning algorithm for target tracking application.

1: Initialize $Q(s, a) = 0$ and $e(s, a) = 0$
2: **while** Residual energy is not equal to zero **do**
3: Determine current state s by application variables
4: Select an action a, using policy
5: Execute the selected action
6: Calculate reward for the executed action (Eq. 4.48)
7: Update the learning rate (Eq. 4.18)
8: Calculate the temporal difference error (Eq. 4.12)
9: Update the eligibility traces (Eq. 4.17)
10: Update the Q-value (Eq. 4.11)
11: Shift to next state based on the executed action
12: **end while**

The learning rate α is decreased slowly in such a way that it reflects the degree to which a state-action pair has been chosen in the recent past. It is calculated as

$$\alpha = \frac{\zeta}{visited(s,a)} \tag{4.18}$$

where ζ is a positive constant. $visited(s,a)$ represents the visited state-action pairs so far [46].

Independent reinforcement learning (RL)

RL task scheduling follows the work of Shah et al. [66] which uses traditional Q learning [85] as online learning strategy. In Q learning the scheduling policy is represented by a two-dimensional matrix $Q_{t+1}(s,a)$ indexed by state-action pairs. The optimal Q value 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 [85].

The main idea of RL is to allow each individual sensor node to self-schedule its tasks and allocate its resources by learning their usefulness in any given state while honoring the application defined constraints and maximizing the total amount of reward over time.

In Q learning every agent needs to maintain a Q matrix for the value functions. Initially all entries of the Q matrix are zero and the agent of the nodes may be in any state. Based on the application defined variables, the system goes to a particular state. Then it performs an action which depends on the status of the nodes.

Algorithm 4 depicts the RL algorithm.

Algorithm 4 Q learning for task scheduling.

1: Initialize $Q(s,a) = 0$. Where s is the set of states and a is the set of actions
2: **while** Residual energy is larger than zero **do**
3: Determine current state s by application variables
4: Select an action a which has the highest Q value
5: Execute the selected action
6: Calculate Q value for the executed action (Eq. 4.19)
7: Calculate the value function for the executed action (Eq. 4.20)
8: Shift to next state based on the executed action
9: **end while**

It calculates the Q value for this (state, action) pair as

$$Q_{t+1}(s_t,a_t) = (1-\alpha)Q_t(s_t,a_t) + \alpha(r_{t+1}(s_{t+1}) + \gamma V_t(s_{t+1})) \tag{4.19}$$

$$V_{t+1}(s_t) = \max_{a \in A} Q_{t+1}(s_t,a) \tag{4.20}$$

where $Q_{t+1}(s_t, a_t)$ means the update of the Q value at time $t+1$ after executing the action a at time step t. r_{t+1} represents the immediate reward after executing the action a at time t, V_t represents the value function for node at time t and V_{t+1} represents the value function at time $t+1$. $\max_{a \in A} Q_{t+1}(s_t, a)$ means the maximum Q value after performing an action from the action set A for the agent i. γ is the discount-factor which can be set to a value in $[0, 1]$. For higher γ values, the agent relies more on the future than the immediate reward. α is the learning rate parameter which can be set to a value in $[0, 1]$. It controls the rate at which an agent tries to learn by giving more or less weight to the previously learned utility value. When α is close to 1, the agent gives more priority to the previously learned utility value.

4.2.2 Bandit solvers

We use the classical adversarial algorithm Exp3 (Exponential-weight algorithm for exploration and exploitation) for task scheduling [6].

The algorithm can be stated as follows:

Algorithm 5 Task Scheduling by Bandit Solver Exp3

1: Parameters: Number of tasks A, Factor $\kappa \leq 1$
2: Initialization: $w_{i,0} = 1$ and $P_{i,1} = 1/A$ for $i = 1, 2, \ldots, A$
3: **while** Residual energy is not equal to zero **do**
4: Determine current s based on application variables
5: Select an action $a \in \{1, 2, \ldots, A\}$ based on the P_t
6: Execute the selected action
7: Calculate the reward (Eq. 4.23)
8: Update the weights (Eq. 4.22)
9: Calculate the updated probability distribution (Eq. 4.21)
10: Shift to next state based on the executed action
11: **end while**

Exp3 has a parameter κ which controls the probability with which arms are explored in each round. At each time step t, Exp3 draws an action a according to the distribution $P_{1,t}, P_{2,t}, ..., P_{A,t}$. The distribution can be calculated by the following equation

$$P_{j,t+1} = (1 - \kappa)\frac{w_{a,t}}{\sum_{j=1}^{A} w_{j,t}} + \frac{\kappa}{A}, j = 1, 2, \ldots, A \tag{4.21}$$

where $w_{a,t}$ is the weight associated with the action a at time t.

This distribution is a mixture of the uniform distribution and a distribution which assigns to each action a probability mass exponential in the estimated reward for that action. Intuitively, mixing in the uniform distribution is done to make sure

that the algorithm tries out all actions A and gets good estimates of the rewards for each action.

Weight for each action can be calculated by following equation

$$w_{a,t} = w_{a,t-1} e^{\kappa r_{t+1}} \tag{4.22}$$

where r_{t+1} is the reward after executing the action a.

Reward can be calculated by following equation

$$r_{t+1} = \frac{r_t}{P_{a,t}} \tag{4.23}$$

where $P_{a,t}$ is the calculated probability distribution for the action a by the Equation 4.21.

Exp3 works by maintaining a list of weights w_i by the Equation 4.22 for each of the actions, using these weights to decide which action to take next based on a probability distribution P_t, and increasing the relevant weights when the reward is positive. The egalitarianism factor $\kappa \in [0, 1]$ tunes the desire to pick an action uniformly at random. If $\kappa = 1$, the weights have no effect on the choices at any step.

4.3 Task scheduling for target tracking

Tracking mobile targets is a typical and generic application for WSNs. Since monitoring the environment is one of the fundamental applications of WSNs, location-tracking of the mobile targets helps to trace the paths of the moving targets in the area where sensor nodes are deployed. Target tracking is challenging mainly for two reasons: limited energy budget of the sensor nodes and the short range of communications. Target tracking application consists of several tasks like sensing, detecting, tracking, sending messages, etc. Scheduling of these tasks has an effect on the performance of the tracking. There should be a trade-off between energy consumption and tracking quality [57]. We demonstrate our task scheduling approach using such target tracking application. We consider a sensor network which consists of a set of randomly deployed nodes.

4.3.1 Target tracking using combinatorial auction

Figure 4.1 shows a particular movement of a single target, i.e., the red dotted circles represents the target's position at different time points. We consider three nodes, which are fully connected. Each node has no information of what actions are better for them in terms of energy consumption. They will learn by performing some actions over time based on their utility. Here we consider the tasks needed for target tracking. These are transmit, receive, sleep, and sense. Every node maintains a set B of the current best bids which consist of set of items. At any point in the auction, there will also be a set $W \subseteq B$ of currently winning bids. All bids are broadcasted

and when a node receives a bid from another node it updates the set of best bids and determine if the new bid is indeed better than the currently winning bid. After performing the PUASEBID algorithm (cp. Section 4.1 for details) node A, B, and C will aware that which task is giving higher utility. Each of the sensor nodes are equipped with microprocessor with the CPU frequency randomly selected between 100 MHz and 300 MHz, and initial energy level of each node is 2.8 Joule. The other parameters for computational and communication energy consumption are $V_T = 26$ mV, C=0.67 nF, $I_0 = 1.196$ mA, $n = 21.26$, $K = 239.28$ MHz/V, $c = 0.5$, α =0.5, β =0.5 and γ =0.5 used for calculating the computational cost and communication energy in [23]. The signal strength and the distance from the target are randomly chosen from the values between [1,10].

To calculate the resource price, we need the following information for performing the tasks [23]:

- **Required CPU cycle:**
  Sensing: 10 MHz
  Transmit: 26 MHz
  Receive: 26 MHz
  Sleeping: 0

- **Energy requirement:**
  Sensing: 0.0000841 J
  Transmit: 0.00233 J
  Receive: 0.00231 J
  Sleeping: 0.000012 J

- **Ideal time gap:**
  Sensing: 25 msec
  Transmit: 10 msec
  Receive: 10 msec
  Sleeping: 0

We calculate the variance of the available energy as $Var = \frac{1}{n}\sum_{i=1}^{n}(E_i - E^{'})^2$, where n is number of sensor nodes, E is the remaining energy level of sensor nodes and $E^{'}$ is the mean which is calculated by $E^{'} = \frac{1}{n}\sum_{i=1}^{n} E_i$. Variance of the available energy is a way to measure the energy efficiency which is used in an existing auction based method [23] and we also use for this application scenario. The higher variance value corresponds to the unbalanced available energy among the sensor nodes and also not energy efficient.

4.3.2 Target tracking using cooperative Q learning

Figure 4.1 shows the movement of a single target, i.e., the "red dotted circles" represents the target's position at different time points. We consider three nodes which are fully connected. Each node has no information of what actions are better for them in terms of energy consumption. Nodes learn by performing some actions over time based on their utility.

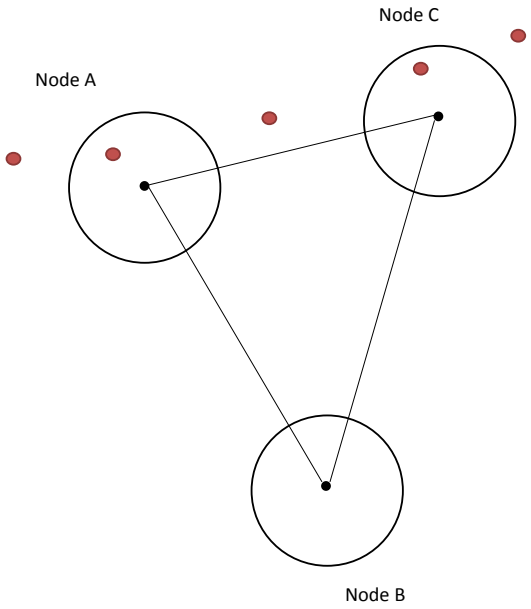


Figure 4.1: Target tracking example. Red dots denote the different positions of a moving target at different time steps. Circles denote the sensing range and black lines denote the communication link between the sensor nodes.

We consider a snapshot of the system that the target is within the FOV of Node A and is moving and finally entering the FOV of Node C. Suppose Node A is in a particular state based on application defined or system variables. In that particular state, the node tries out the actions and finds out a particular one which maximizes the Q value. Then it moves to the next state after performing that action. Node A sends the value function to its neighbors. Going to that state, it selects another action which has maximum Q value. Now in this example, when an object enters into the FOV of Node A, it receives a reward after executing tasks. So, Node A learns after some times that it receives a positive reward for sensing if something in its FOV. The object moves from the sensing range of node A and does not exist in the range of any nodes. In this case, all nodes gets maximum reward by staying in the sleeping mode. The object moves in to the field of view of node C and it receives

a higher reward for sensing. After that it will move to another state by calculating the Q value. Node B has nothing to transmit or receive. So, Node B gets higher reward by staying in sleeping at this particular state of the environment. In this way, each node is able to decide on the best available tasks for a particular state in a dynamic environment.

We consider two different application scenarios for finding the efficiency of our proposed approach based on cumulative reward over time, total number of execution for each action, and the residual energy of the network over time.

In the first application scenario, we consider three variables which are "object in field of view (FOV)", "data to transmit (DTT)", and "residual resource level (RL)". In the second application scenario, we consider the variables which correspond to "number of generated packets", the "timer of completing delivery of a packet", the "residual energy of the node", the "transmit cost", the "receive cost", the "sense cost", the "sleep cost", and the "number of connections to the neighbor nodes". We consider sleep, sense, transmit, and receive as our tasks set for our both application scenarios.

First Application Scenario

Based on the system variables, we abstract the application by the following set of states:

- FOV=0 and RL >=Minimum energy required for Transmission and DTT= 0
- FOV=0 and RL >= Minimum energy required for Transmission and DTT=1
- FOV=0 and RL < Minimum energy required for Transmission and DTT=1
- FOV=1 and RL >= Minimum energy required for Sensing and DTT= 0
- FOV=1 and RL < Minimum energy required for Sensing and DTT= 0
- FOV=1 and RL < Minimum energy required for Sensing and DTT=1
- FOV=1 and RL >= Minimum energy required for Sensing and DTT=1
- FOV=0 and RL < Minimum energy required for Transmission and DTT= 0

Suppose if the variable, object in field of view is equal to 1 that means there is object in the sensing range of that node. In each state, the node tries out the tasks from task sets and finds the task which has higher Q value. Task changes the state variables. For example, after performing sense task in state 1, resource level will be equal to (residual energy minus energy required for sensing) and field of view will be equal to 1. Now to calculate the Q values we need reward. In this experiment, we consider some static values for the reward. We assign the most reward to the

task which needs the least amount of energy. Here is the ascending order of tasks for energy consumption: Sleeping, Sensing, Receiving, and Transmitting.

Tasks	***Reward***
Sleeping	5 units
Sensing	4 units
Receiving	3 units for success -3 units for not success
Transmitting	2 units for success -2 units for not success
Discount factor, γ	0.5
Learning rate, α	0.5

Table 4.1: Reward values for application scenario 1.

Table 4.1 shows the values for the reward we consider for application scenario 1. Here "success" means the node has the minimum energy to execute the task.

Second Application Scenario

In our second application scenario we consider a different set of states, the same tasks set (sleep, sense, transmit, receive) and different rewards. This state space considers more state variables. We consider a multi-target tracking systems. There are some moving targets throughout the system. Each node is connected with neighbors like a fully connected graph. If a target enters in to the sensing area of a particular node, it generates a packet or data to transmit.

For the number of data to transmit, we use "sNodes.aPackets.Count". For the timer of completing delivery of a packet, we use "iTransmitting". For the residual energy of the node, we use "sNodes.iResidualEnergy". For sensor cost, transmit cost, receive cost and sleep cost we use "iSensorCost", "iTransmitCost", "ireceive-Cost" and "iSleepCost" respectively. "sNodes.aconnections"denotes the number of connections to neighbors for each node.

Based on these variables, we abstract the application with the following states:

- sNodes.aPackets.Count > 0 and sNodes.iResidualEnergy $>=$ iSensorCost.
- sNodes.aPackets.Count > 0 and iTransmitting $= 0$ and sNodes.iResidualEnergy $<$ iTransmitCost
- sNodes.aconnection $!= NULL$ and sNodes.iResidualEnergy $<$ iReceiveCost

Tasks	***Reward***
Sleeping	0.05 units
Sensing	0.001 units
Receiving	(0.3-iReceiveCost) for success (-iReceiveCost) for not success
Transmitting	(0.2-iTransmitCost) for success (-iTransmitCost) for not success
Discount factor, γ	0.5
Learning rate, α	0.5

Table 4.2: Reward values for application scenario 2.

Table 4.2 shows the reward function we consider for the application scenario 2.

4.3.3 Target tracking using cooperative $SARSA(\lambda)$ learning and Exp3 bandit solvers

We consider the following set of actions, states, and reward function for a target tracking application using $SARSA(\lambda)$ and Exp3.

Set of actions
We consider the following actions in our target tracking application:

- *Detect_Targets:* This function scans the field of view (FOV) and returns the number of detected targets in the FOV.
- *Track_Targets:* This function keeps track of the targets inside the FOV and returns the current 2D positions of all targets. Every target within the FOV is assigned with a unique ID number.
- *Goto_Sleep:* This function shuts down the sensor node for single time period. It consumes the least amount of energy of all available actions.
- *Send_Message:* This function sends information about the target's trajectory to neighboring nodes. The trajectory information includes (i) the current position and time of the target and (ii) the estimated speed and direction. This function is executed when the target is about to leave the FOV.
- *Predict_Trajectory:* This function predicts the velocity of the trajectory. A simple approach is to use the two most recent target positions, i.e., (x_t, y_t) at time t_t and (x_{t-1}, y_{t-1}) at t_{t-1}. Then the constant target's speed can be estimated as

$$v = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}/(t_t - t_{t-1}) \tag{4.24}$$

- *Intersect_Trajectory:* This function checks whether the trajectory intersects with the FOV and predicts the expected time of the intersection. This function is executed by all nodes which receive the "target trajectory" information from a neighboring node. Trajectory intersection with the FOV of a sensor node is computed by basic algebra. The expected time to intersect the node is estimated by

$$\tilde{t}_i = D_{P_i P_j}/v \tag{4.25}$$

where $D_{P_i P_j}$ is the distance between points P_j and P_i. P_j represents the point where the trajectory is predicted at node j and P_i corresponds to the trajectory's intersection points with the FOV of node i (cp. Figure 4.2). v is the estimated velocity as calculated by Equation 4.24.

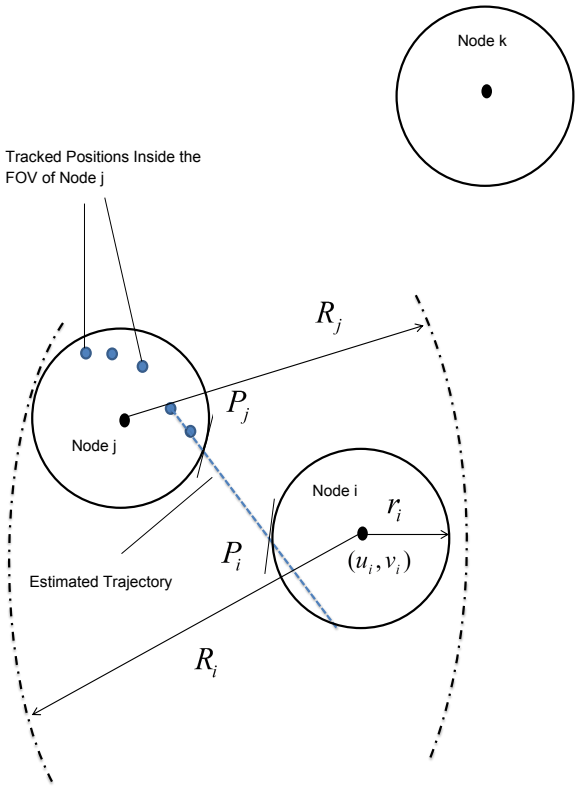


Figure 4.2: Target prediction and intersection. Node j estimates the target trajectory and sends the trajectory information to its neighbors. Node i checks whether the predicted trajectory intersects its FOV and computes the expected arrival time.

We consider the advanced trajectory prediction and intersection for these methods. Inputs for this prediction task is the last few tracked positions of the target.

Here we consider last 6 tracked positions of the target based on simulation studies. We linearize the trajectory which is given by the last 6 tracked positions of the target considering the constant speed and direction. The speed is calculated by the Equation 4.24.

Suppose (x_1, y_1), (x_2, y_2) $\dots (x_n, y_n)$ are tracked positions of the moving object inside the FOV of the sensor node at time steps t_1, t_2 $\dots t_n$.

The trajectory can be predicted by the regression line [58] in Equation 4.26

$$y = bx + a + \epsilon \tag{4.26}$$

where b is the slope, a is the intercept and ϵ is the residual or error for the calculation.

So residual, ϵ can be calculated by following

$$\epsilon_i = y_i - bx_i - a \tag{4.27}$$

where $i = 1, 2, 3, \dots, n$.

If we sum up the squares of the residuals of all the points from the line we get is a measure of the fitness of the line. Our aim should be to minimize this value.

So, the square of the residual as follows

$$\epsilon_i^2 = (y_i - bx_i - a)^2 \tag{4.28}$$

To calculate the sum of square residuals, we add all the individual square residuals together as follows

$$J = \sum_{i=1}^{n} (y_i - bx_i - a)^2 \tag{4.29}$$

where J is the sum of square residuals. n is the number of considered points.

We need to minimize the J in the Equation 4.29. The minimum value for J has to occur when is first derivative is zero. The partial derivatives for J with respect to the two parameters of the regression line b and a. We want these to be zero to get the minimum [8].

$$\frac{\partial J}{\partial b} = \sum_{i=1}^{n} 2(y_i - bx_i - a)(-x_i) = 0 \tag{4.30}$$

$$\frac{\partial J}{\partial a} = \sum_{i=1}^{n} 2(y_i - bx_i - a)(-1) = 0 \tag{4.31}$$

Equations 4.30 and 4.31 can be shuffled and divided by two and which as follows

$$\sum_{i=1}^{n} bx_i + \sum_{i=1}^{n} a = \sum_{i=1}^{n} y_i \tag{4.32}$$

$$\sum_{i=1}^{n} bx_i^2 + \sum_{i=1}^{n} ax_i = \sum_{i=1}^{n} x_i y_i \tag{4.33}$$

We can pull some constants out in front of the summations. The $\sum_{i=1}^{n} a$ can be written as na in the Equation 4.32. We can also pull out the b and a from the Equations 4.32 and 4.33. These give us two equations as follows

$$b\sum_{i=1}^{n} x_i + na = \sum_{i=1}^{n} y_i \tag{4.34}$$

$$b\sum_{i=1}^{n} x_i^2 + a\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \tag{4.35}$$

Now from the Equations 4.34 and 4.35, some simple substitutions between the two equations provide as follows

$$a = \frac{\sum y}{n} - b\frac{\sum x}{n} \tag{4.36}$$

$$b = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \tag{4.37}$$

These formulas in Equations 4.36 and 4.37 do not tell us how precise the estimates are. That is, how much the estimators a and b can deviate from the "true" values of a and b. It can be solved by confidence intervals.

Using Student's t-distribution with $n-2$ degrees of freedom [76], we can construct a confidence interval for a and b as follows

$$\hat{b} \in \left[\ b - s_b t^*_{n-2},\quad b + s_b t^*_{n-2}\ \right] \tag{4.38}$$

$$\hat{a} \in \left[\ a - s_a t^*_{n-2},\quad a + s_a t^*_{n-2}\ \right] \tag{4.39}$$

where $\hat{a}$ and $\hat{b}$ are the new estimated values of a and b. t^*_{n-2} is the $(1-\tau/2)$-th quantile of the t_{n-2} distribution. For example, if $\tau = 0.05$ then the confidence level is 95 percent. s_a and s_b are the standard deviations as follows

$$s_b = \sqrt{\frac{\frac{1}{n-2}\sum_{i=1}^{n} \varepsilon_i^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} \tag{4.40}$$

$$s_a = s_b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{1}{n(n-2)}\left(\sum_{i=1}^{n} \varepsilon_i^2\right)\frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}} \tag{4.41}$$

where $\bar{x}$ is the average of the x values.

In Figure 4.3, we observe that there are some tracked positions of the target which is denoted by the "black" dots. At first, we predict the regression line and find the middle line. Then we calculate the confidence band by which we get two other lines.

For the intersection with the circles, we consider the line as follows

$$y = bx + a \tag{4.42}$$

where b is the slope, a is the intercept.

The line given by Equation 4.42 intersects a circle (sensing range is considered as a circle) given by Equation 4.43

$$(x - u_1)^2 + (y - v_1)^2 = r_1^2 \tag{4.43}$$

where (u_1, v_1) is the center and r_1 is the radius of the circle.

Substituting the value of Equation 4.42 in Equation 4.43 gives as follows

$$(x - u_1)^2 + ((bx + a) - v_1)^2 = r_1^2 \tag{4.44}$$

Simply expanding the Equation 4.44 by algebraic formula gives as follows

$$(1 + b^2)x^2 + 2(ab - bv_1 - u_1)x + (u_1^2 + v_1^2 + a^2 - 2av_1 - r_1^2) = 0 \tag{4.45}$$

Equation 4.45 is a quadratic equation of x and can be solved using the quadratic formula. Labeling the terms of the Equation 4.45 like $(1+b^2) = A$, $2(ab-bv_1-u_1) = B$ and $(u_1^2 + v_1^2 + a^2 - 2av_1 - r_1^2) = C$, we can get $Ax^2 + Bx + C$, then we solve this by following formula

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{4.46}$$

if $B^2 - 4AC < 0$ then the line misses the circle. If $B^2 - 4AC = 0$ then the line is tangent to the circle. If $B^2 - 4AC > 0$ then the line meets the circle in two distinct points.

We can substitute x in Equation 4.42 from the Equation 4.46 to get the y values

$$y = b(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}) + a \tag{4.47}$$

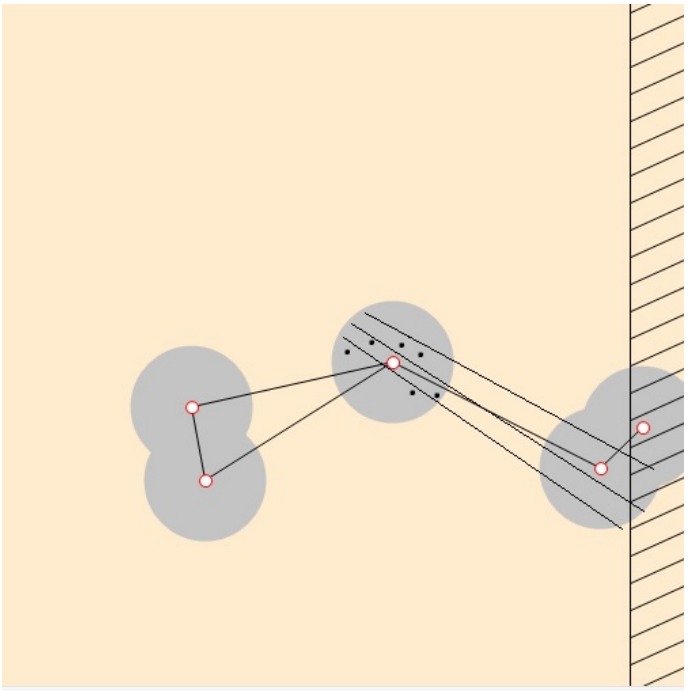


Figure 4.3: Trajectory prediction and intersection. Black dots denote the tracked positions of a target. The middle line is drawn based on linear regression. The other two lines are drawn by confidence interval.

Set of states
We abstract the application by three states at every node.

- *Idle:* This state indicates that there is currently no target detected within the node's FOV and the local clock is too far from the expected arrival time of any target already detected by some neighbor. If the time gap between the local clock L_c and the expected arrival time N_{ET} is greater than or equal to a threshold Th_1 (cp. Figure 4.4), then the node remains in the idle state. The threshold Th_1 is set to 5 based on our simulation studies. In this state, the sensor node performs *Detect_Targets* less frequently to save energy.

- *Awareness:* There is currently also no detected target in the node's FOV in this state. However, the node has received some relevant trajectory information and the expected arrival time of at least one target is in less than Th_1 clock ticks. In this state, the sensor node performs *Detect_Targets* more frequently, since at least one target is expected to enter the FOV.

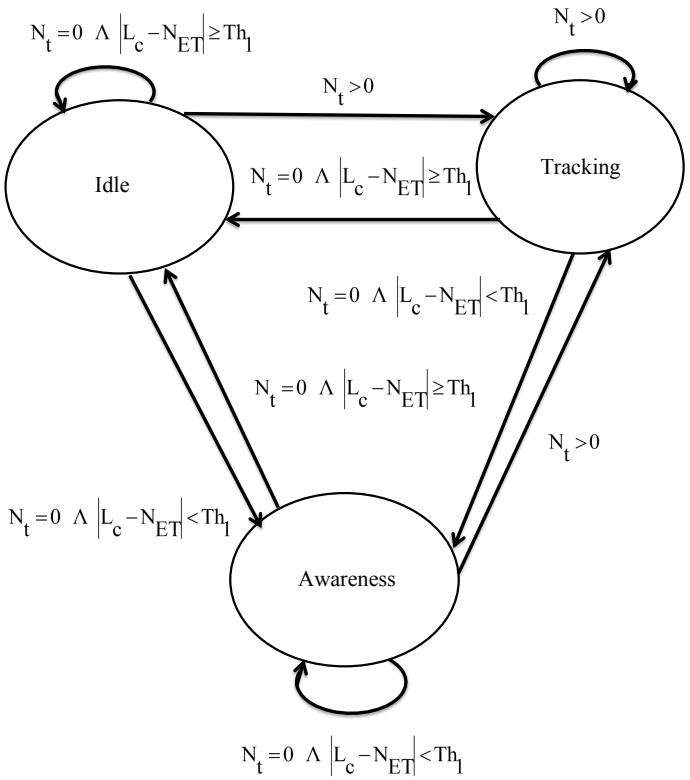


Figure 4.4: State transition diagram. States change according to the value of two application variables N_t and N_{ET}. L_c represents the local clock value and Th_1 is a time threshold.

- *Tracking:* This state indicates that there is currently at least one detected target within the node's FOV. Thus, the sensor node performs tracking frequently to achieve high tracking performance.

Obviously, the frequency of executing $Detect\_Targets$ and $Track\_Targets$ depends on the overall objective, i.e., whether to focus more on tracking performance or energy consumption. The states can be identified by two application variables, i.e., the number of detected targets at the current time N_t and the list of arrival times of targets expected to intersect with node N_{ET}. N_t is determined by the task $Detect\_Targets$ which is executed at time t. If the sensor node executes the task $Detect\_Targets$ at time t then N_t returns the number of detected targets in the FOV. Each node maintains a list of appearing targets and the corresponding arrival time. Targets are inserted in this list if the sensor node receives a message and the estimated trajectory intersects with the FOV. Targets are removed if a target is detected by the node or the expected arrival time with an additional threshold Th_1 has expired.

Figure 4.4 depicts the state transition diagram where L_c is the local clock value of the sensor node and Th_1 represents the time threshold between L_c and N_{ET}.

Reward Function

The reward function in our algorithm is defined as

$$r = \beta(E_i/E_{max}) + (1 - \beta)(P_t/P) \tag{4.48}$$

where the parameter β balances the conflicting objectives between E_i and P_t. E_i represents the residual energy of the node. P_t represents the number of tracked positions of the target inside the FOV of the node. E_{max} is the maximum energy level of sensor node and P is the number of all possible detected target's positions in the FOV.

CHAPTER

5 Implementation and Evaluation

In this chapter, we describe our simulation environment first. We describe our experimental setup. We implement our proposed methods in our simulation environment. We show the results varying the balancing factor of the reward function, varying the number of nodes, varying the target movement, and varying the sensing range. We evaluate our proposed methods in terms of tracking quality/energy consumption trade-off. Discussion of the results concludes this chapter.

5.1 Simulation environment

We implement and evaluate the task scheduling methods using a WSN multi-target tracking scenario implemented in a C# simulation environment.

The simulator consists of two stages: the deployment of the nodes and the execution of the tracking application. In our evaluation scenario the sensor nodes are uniformly distributed in a 2D rectangular area. A given number of sensor nodes are placed randomly in this area which can result in partially overlapping FOVs of the nodes. However, placement of nodes on the same position is avoided. Before deploying the network, the network parameters should be configured using the configuration sliders.

The following network parameters can be configured by our simulator.

- Network size: Network size means the number of nodes in the network. In current settings of the simulator, number of sensor nodes can be varied between [3,40].
- Sensor radius: Sensor radius is the sensing range of the sensors in the network. Sensor radius can be varied between [1,50].
- Transmission radius: Transmission radius is the maximum distance within two sensor nodes communicating with each other. If set to a high value, nodes on the opposite side of the rectangular area may be able to reach each other. If

set to a low value, nodes must be very close to communicate with each other. Transmission radius can be varied between [1,50].

Once these network parameters are configured, the sensor nodes can be deployed by pressing the "Deploy Network" button. After deploying the sensor nodes, the simulation may be run by pressing the "Start Simulation" button with our selected algorithm. The simulator shows the targets moving through the area and the sensors. The progress of the network can be monitored via the "Simulation Status" box. In this box, we observe the node status, running time of the simulation, remaining energy of the network, and number of sensors alive. A new simulation may be run by stopping and restarting the simulation. The previous simulation may be reviewed by pressing the "Replay Simulation" button. Figure 5.1 shows our simulation environment. Some of the graphical parts of our simulation environment are obtained from [78].

The network is displayed on the simulation environment as a set of red circles surrounded by gray circles. The red circles denote the sensor nodes and the gray circles denote the sensing range of the nodes. Each node is connected to nearby nodes by black lines which represent the communication links. When a message is being exchanged, it appears as red. The color in the center of the red circle represents the battery status of the node, which gradually shifts from white to black. White color denotes the nodes in full power and black color denotes the nodes with no power. When a node loss all power, the node becomes completely black. The gray area of the node shrinks and disappears. All of the communication links associated with the node disappear as well.

Targets move around in the area based on a Gauss-Markov mobility model [1]. The Gauss-Markov mobility model was designed to adapt to different levels of randomness via tuning parameters. Initially, each mobile target is assigned with a current speed and direction. At each time step t, the movement parameters of each target are updated based on the following rule

$$S_t = \eta S_{t-1} + (1-\eta)S + \sqrt{1-\eta^2}S^G_{t-1} \tag{5.1}$$

$$D_t = \eta D_{t-1} + (1-\eta)D + \sqrt{1-\eta^2}D^G_{t-1} \tag{5.2}$$

where S_t and D_t are the current speed and direction of the target at time t. S and D are constants representing the mean value of speed and direction. S^G_{t-1} and D^G_{t-1} are random variables from a Gaussian distribution. η is a parameter in the range $[0, 1]$ and is used to vary the randomness of the motion. Random (Brownian) motion is obtained if $\eta = 0$, and linear motion is obtained if $\eta = 1$. At each time t, the target's position is given by the following equations

$$x_t = x_{t-1} + S_{t-1}cos(D_{t-1}) \tag{5.3}$$

$$y_t = y_{t-1} + S_{t-1}sin(D_{t-1}) \tag{5.4}$$

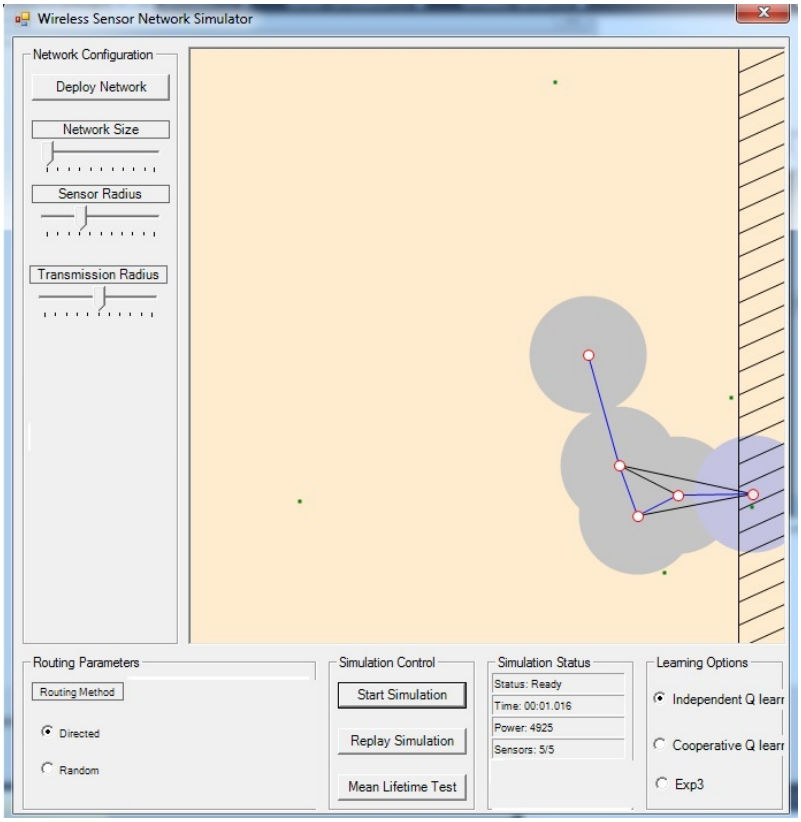


Figure 5.1: Simulation environment.

In our simulation we limit the number of concurrently available targets to seven. The total energy budget for each sensor node is considered as 1000 units. Table 5.1 shows the energy consumption for the execution of each action. Sending messages over two hops consumes energy on both the sender and relay nodes. To simplify the energy consumption at the network level, we aggregate the energy consumption to 10 units on the sending node only. We set the discount factors $\gamma = 0.5$, $\gamma_1 = 0.5$ and $\gamma_2 = 0.5$ for the online learning algorithms and vary the learning rate according to Equation 4.18. We set $\zeta = 1$ for calculating learning rate in Equation 4.18. We set $k = 0.25$, $\epsilon_{min} = 0.1$, $\epsilon_{max} = 0.3$ and $S_{max} = 3$ in Equation 4.15. We set $\lambda = 0.5$ for the eligibility trace calculation by Equation 4.17. We set the egalitarianism factor $\kappa = 0.5$ for Exp3. We consider the sensing radius as $r_i = 5$ and communication radius as $R_i = 8$. We set these fixed values for the parameters based on our simulation studies. For each simulation run we aggregate the achieved tracking quality and energy consumption and normalize the tracking quality and energy consumption to $[0, 1]$.

Action	*Energy Consumption*
Goto_Sleep	1 unit
Detect_Targets	2 units
Intersect_Trajectory	3 units
Predict_Trajectory	4 units
Send_Message (one hop)	5 units
Send_Message (two hops)	10 units
Track_Targets	6 units

Table 5.1: Energy consumption of the individual actions.

5.2 Experimental setup

We implement the methods cooperative $SARSA(\lambda)$ learning (CRL) for one hop and two hops, bandit solvers Exp3, and independent reinforcement learning (RL) in our simulation environment (cp. Section 5.1 for more details). We set the discount factors of the learning algorithms, egalitarianism factor for Exp3, eligibility trace factor, learning rate parameters, exploration factors, number of concurrently available targets, total energy budget of each sensor node, energy consumption for each task to fixed values based on simulation studies. We consider 5, 10, and 20 sensor nodes which are placed randomly with partially overlapping FOVs of the nodes. We set sensing radius and communication radius to fixed values based on simulation studies. For each complete simulation run we aggregate the achieved tracking quality and energy consumption. We find out the achieved trade-off between tracking quality and energy consumption for our methods varying the balancing factor of the reward function, number of sensor nodes, and randomness of target movement. We also calculate the average execution time and average number of transferred messages for RL, CRL, and Exp3.

We consider 3 fully connected sensor nodes without overlapping FOVs of the nodes (cp. Figure 4.1) for cooperative Q learning and combinatorial auction method. We set learning rate, discount factor, reward values, energy consumption for each task to fixed values based on simulation studies. We calculate the cumulative reward over time, residual energy of the network, total number of executions of each action for cooperative Q learning.

We calculate the variance of the available energy for different methods, residual energy of the network, and cumulative revenue of the network for combinatorial auction method.

5.3 Simulation results

In this section, first we describe the results comparing our three methods CRL(one hop), CRL(two hop), and RL in terms of tracking quality/energy consumption trade-

off varying the balancing factor of the reward function, number of sensor nodes, and randomness of target movement considering the system model of the Subsection 3.2.1 and the application scenario of the Subsection 4.3.3. We then show the results of cooperative Q learning considering the system model of the Subsection 3.2.1 and both application scenarios of the Subsection 4.3.2. After that, we show the results of the combinatorial auction method considering the system model of Subsection 3.2.2 and the application scenario of the Subsection 4.3.1. We compare cooperative Q learning and combinatorial auction method with static and random scheduling of tasks. We also compare our methods RL, CRL, and Exp3 in terms of tracking quality/energy consumption trade-off considering the system model of the Subsection 3.2.1 and the application scenario of the Subsection 4.3.3 varying the balancing factor of the reward function, number of sensor nodes, randomness of moving targets, and sensing range.

5.3.1 Results of RL, CRL (one hop and two hop)

We perform three experiments with the following assumptions of parameters.

1. To find out the trade-off between tracking quality and energy consumption, we set the balancing factor β to one of the following values $\{0.10, 0.30, 0.50, 0.70, 0.90\}$, keep the randomness of moving target as $\eta = 0.5$ and fix the topology to five nodes.

2. We vary the network size to check the trade-off between tracking quality and energy consumption. We consider three different topologies consisting of 5, 10, and 20 sensor nodes. We keep the balancing factor $\beta = 0.5$ and the randomness of the mobility model $\eta = 0.5$ constant for this experiment.

3. We set the randomness of moving targets η to one of the following values $\{0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.7, 0.9\}$ and set the balancing factor $\beta = 0.5$ and fix the topology to five nodes.

Results varying the balancing factor

We vary the balancing factor β to one of these values $\{0.1, 0.3, 0.5, 0.7, 0.9\}$, keep the randomness of moving target as $\eta = 0.5$, and fix the topology to 5 nodes. Figures 5.2, 5.3, 5.4, 5.5, and 5.6 show the evaluation of tracking quality/energy consumption trade-off for cooperative reinforcement learning (CRL) (one hop), independent reinforcement learning (RL), and cooperative reinforcement learning (CRL) (two hop).

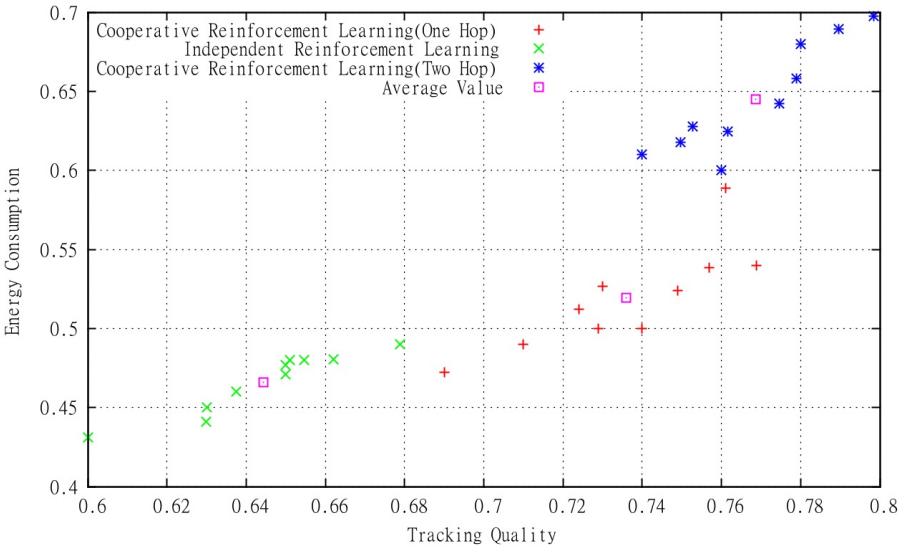


Figure 5.2: Achieved trade-off between tracking quality and energy consumption for $\beta = 0.1$.

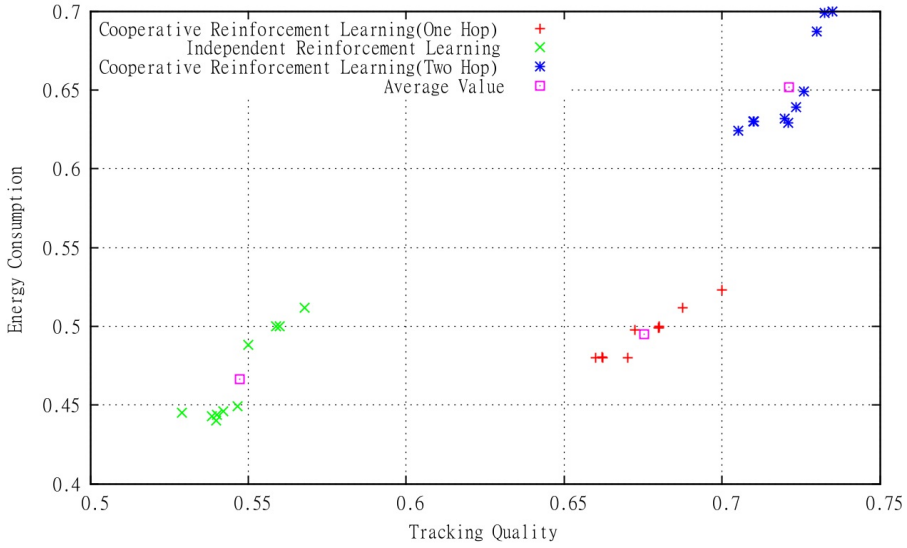


Figure 5.3: Achieved trade-off between tracking quality and energy consumption for $\beta = 0.3$.

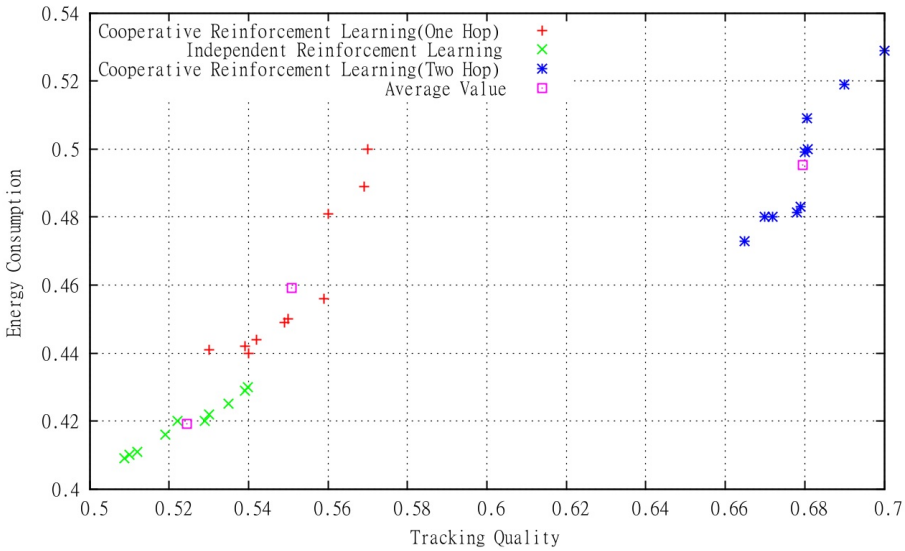


Figure 5.4: Achieved trade-off between tracking quality and energy consumption for $\beta = 0.5$.

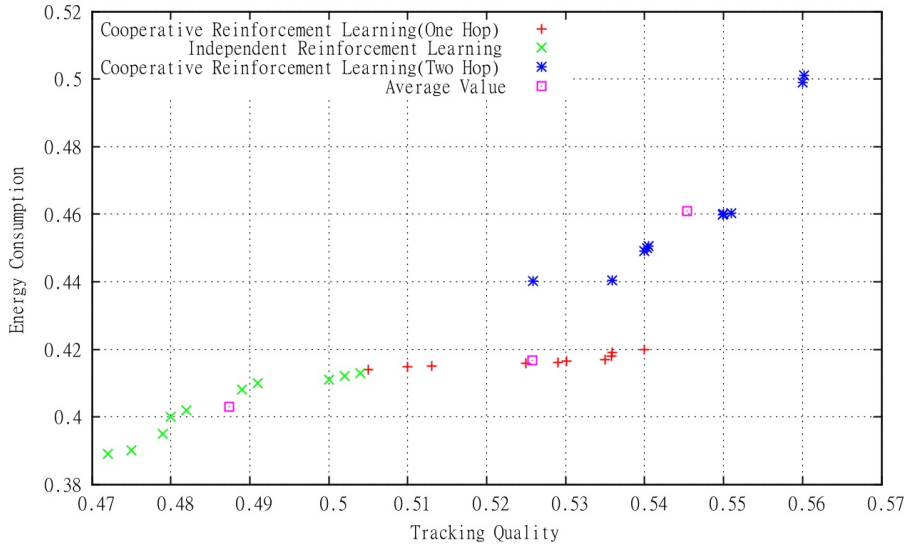


Figure 5.5: Achieved trade-off between tracking quality and energy consumption for $\beta = 0.7$.

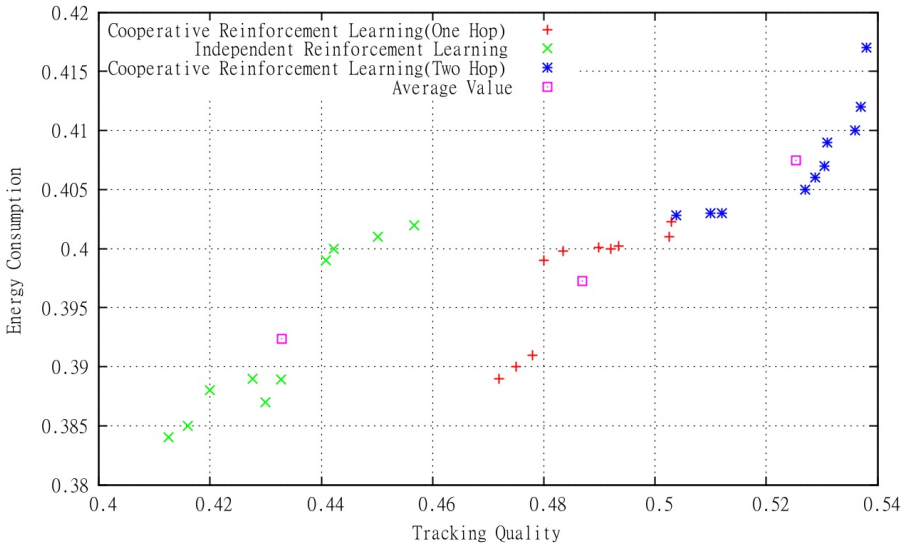


Figure 5.6: Achieved trade-off between tracking quality and energy consumption for $\beta = 0.9$.

Each data point in these figures represents the normalized tracking quality and energy consumption of one complete simulation run. The square symbols represent the average values among the 10 simulation runs for each method. For example with $\beta = 0.1$ in Figure 5.2, the achieved tracking results vary within $(0.69, 0.77)$ and the energy consumption varies within $(0.47, 0.58)$ for our one hop cooperative approach. The average value for this setting is 0.73 and 0.53. It can be clearly seen from these figures that our cooperative approaches outperform the non-cooperative approach with regard to the achieved tracking performance. There is a slight increase in the energy consumption especially for the two hop cooperative approach. Figure 5.7 shows the results of the evaluation of RL, CRL(one hop), and CRL(two hop) for various network sizes.

Results varying the number of nodes

We vary the network size to check the trade-off between tracking quality and energy consumption. We consider three different topologies consisting of 5, 10 and 20 sensor nodes. We keep the balancing factor $\beta = 0.5$ and the randomness of the mobility model $\eta = 0.5$ constant for this experiment. Figure 5.7 shows the tracking quality/energy consumption trade-off for various network sizes. Here, each data point in this figure represents the average of the normalized tracking quality and energy consumption of ten complete simulation runs. Here the same trend can be identified, i.e., the cooperative approaches outperform the non-cooperative approach with regard to the achieved tracking performance.

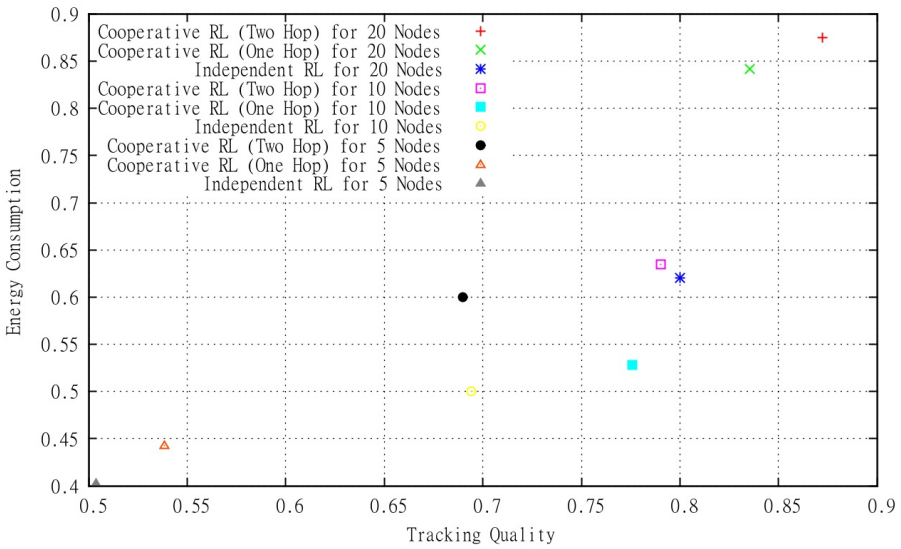


Figure 5.7: Tracking quality versus energy consumption for various network sizes.

Results varying the target movement

Figures 5.8, 5.9, and 5.10 show the results of varying the randomness of target movement.

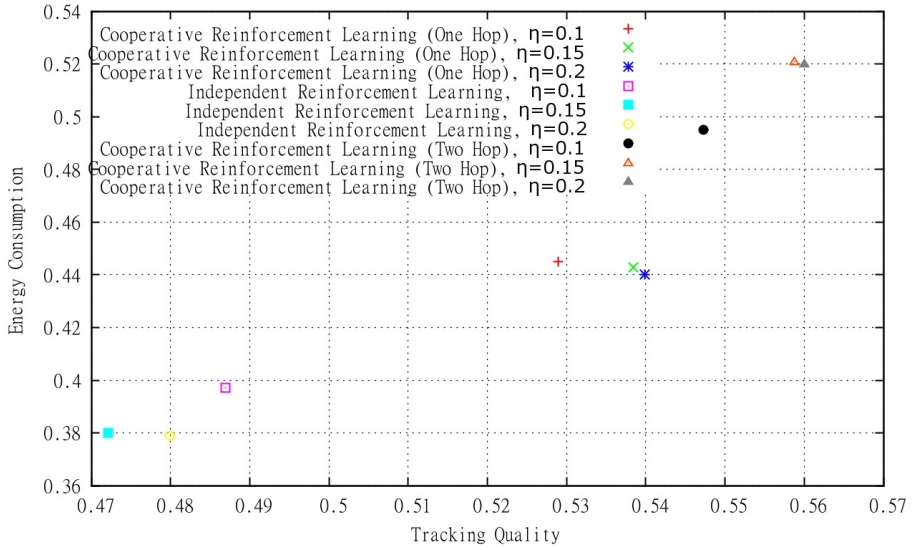


Figure 5.8: Randomness of target movement, η=0.1, 0.15, and 0.2

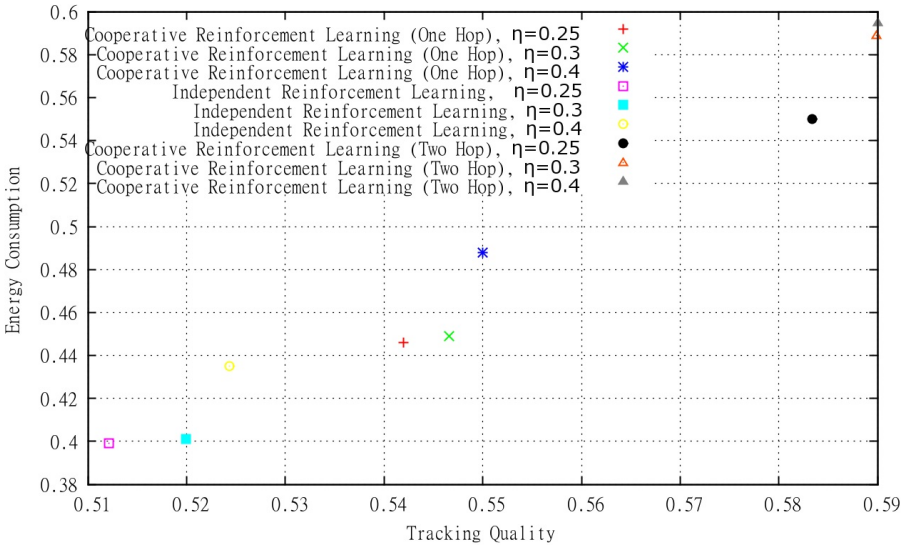


Figure 5.9: Randomness of target movement, η=0.25, 0.3, and 0.4

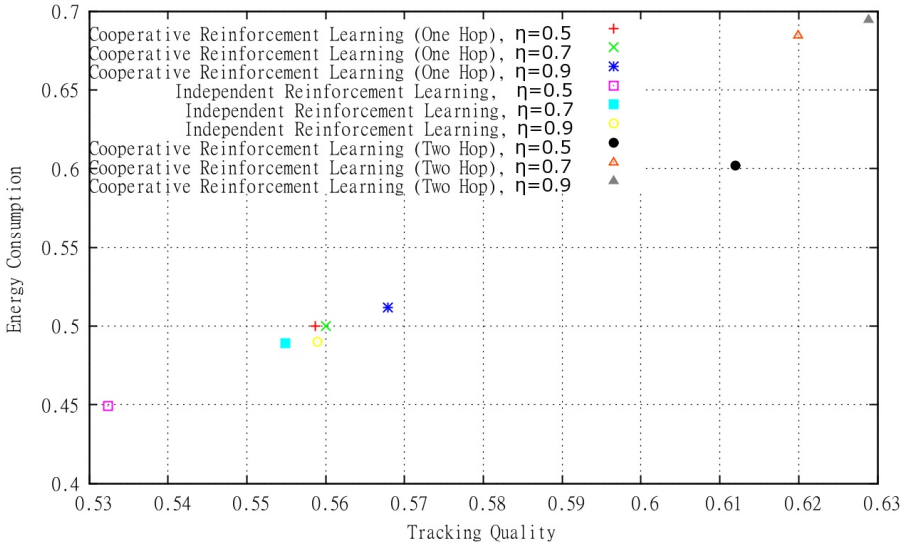


Figure 5.10: Randomness of target movement, η=0.5, 0.7, and 0.9

We set in our simulation the randomness of moving targets η to one of the following values $\{0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.7, 0.9\}$, set the balancing factor $\beta = 0.5$ and fix the topology to five nodes for evaluating the CRL(one hop), RL, and CRL(two hop). Here, each data point in this figure represents the average of the normalized tracking quality and energy consumption of ten complete simulation runs.

From these figures, it can be seen that our cooperative approaches outperform the non cooperative approach in terms of achieved tracking performance. We can see

that for lower randomness, η=0.5, 0.7, and 0.9, independent learning and one-hop cooperative learning show very close results for tracking performance. But for higher randomness, η=0.1, 0.15, and 0.2, independent learning gives poor performance with regard to tracking quality.

5.3.2 Results of cooperative Q learning

In Figure 5.11, we show that the cumulative reward over time of our approach is greater than the other approaches considering the first application scenario (cp. Subsection 4.3.2 for more details). Higher cumulative reward means that the node has chosen tasks in such a way that it maximized the reward. As our reward is defined to minimize the energy consumption, we will get less energy consumption for tasks by getting a higher cumulative reward. We can see there are variations in cumulative rewards. These are the Q values for each method. Q value has also the consideration of the value function of the neighboring nodes which provides a variation in cumulative rewards.

In Figures 5.12, 5.13, and 5.14, we show the task scheduling of node A, B, and C considering the case of Figure 4.1 with cooperative Q learning. Each node has no information about what tasks are better for them and try to learn over time based on the utility of their tasks. For example, node A does not know that it needs to sense as the object is in the field of view. Here each bar represents a task executed at each time step. We represent "Receive", "Transmit", "Sense", and "Sleep" tasks in descending order of height of the bars. We can observe that node A immediately learns that it is getting paid to sense as the object is in its field of view. In the middle of the simulation time, as the target is out of reach of all sensor nodes, all nodes will be rewarded for sleeping. Similarly, we can observe that for node C, it will be rewarded for sensing after some times when the object is in its field of view. After that when the object is out of reach of all sensor nodes, all the nodes will be rewarded for sleeping.

In Figure 5.15, we show the number of executions of each task for each method. We perform this experiment for all methods by considering the case of Figure 4.1. Here we can see that in our approach the number of executions of sleep task is larger than other methods. As sleeping requires less energy among all tasks, our approach shows more energy efficiency by performing more sleep action over time.

Figure 5.16 shows the comparison graph of our approach with the other approaches in terms of cumulative reward over time considering the second application scenario (cp. Subsection 4.3.2 for more details). It shows a better cumulative reward over time for our approach. We consider here the negative rewards with the different set of application variables for states to prove the better efficiency of our system.

Figure 5.17 shows the comparison graph for the residual energy of the network. We calculate the residual energy of the network at each time step. We assume energy budget for each node as 1000 units. For sleeping we spend 2 units, for sensing 3

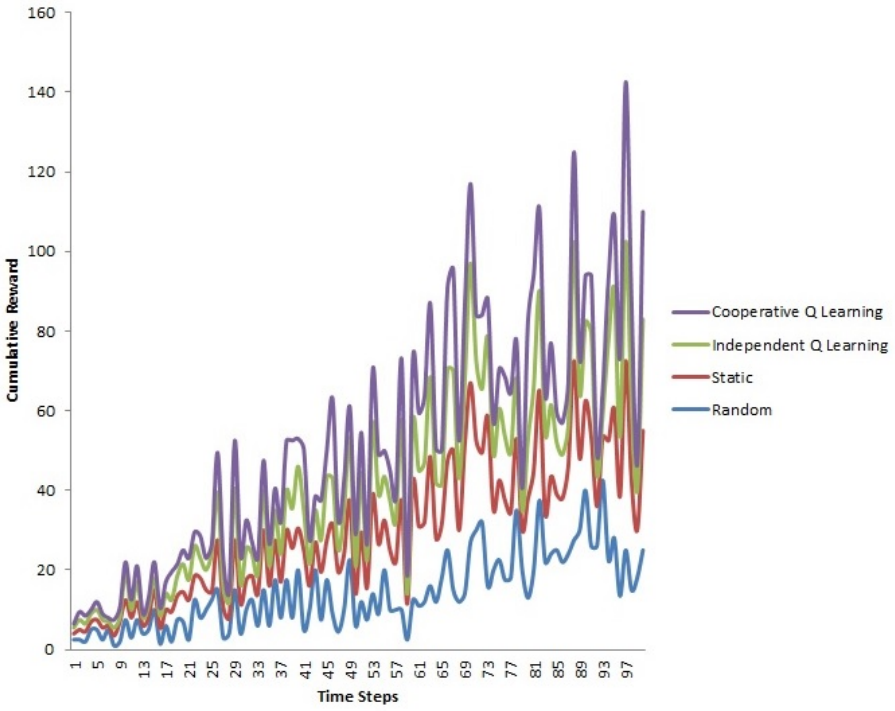


Figure 5.11: Cumulative reward over time by application scenario 1 (cp. Subsection 4.3.2).

units, for receiving 4 units and for transmitting 5 units of energy. We get better result for cooperative Q learning comparing with other approaches. So, it helps to increase the lifetime of the network.

From the simulation results and evaluation, we can say that by applying cooperative Q learning it is possible to learn the usefulness of actions to perform in different states. Cooperative Q learning helps to schedule the tasks in such a way that the energy consumption is reduced. It also receives better cumulative reward over time compared with other approaches.

5.3.3 Results of combinatorial auction method

The simulation results in Figure 5.18 shows that our proposed combinatorial auction based algorithm gives better performance in terms of energy efficiency comparing with static and random scheduling of tasks. We calculate the variance of the available energy of the sensor nodes (cp. Subsection 4.3.1 for more details). Figure 5.19 shows the residual energy of the network at each time step. We can observe that our proposed algorithm gives better performance comparing with other existing techniques. In Figures 5.20, 5.21, and 5.22, we show the task scheduling of node A, B, and C considering the case of Figure 4.1 with combinatorial auction based method. Each node has no information of what tasks are better for them and try to learn

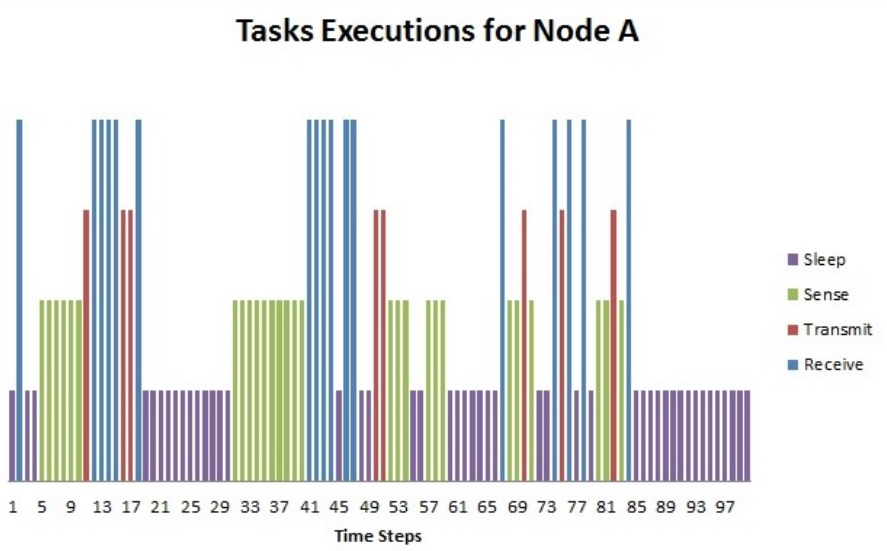


Figure 5.12: Tasks execution for Node A in Figure 4.1 by cooperative Q learning.

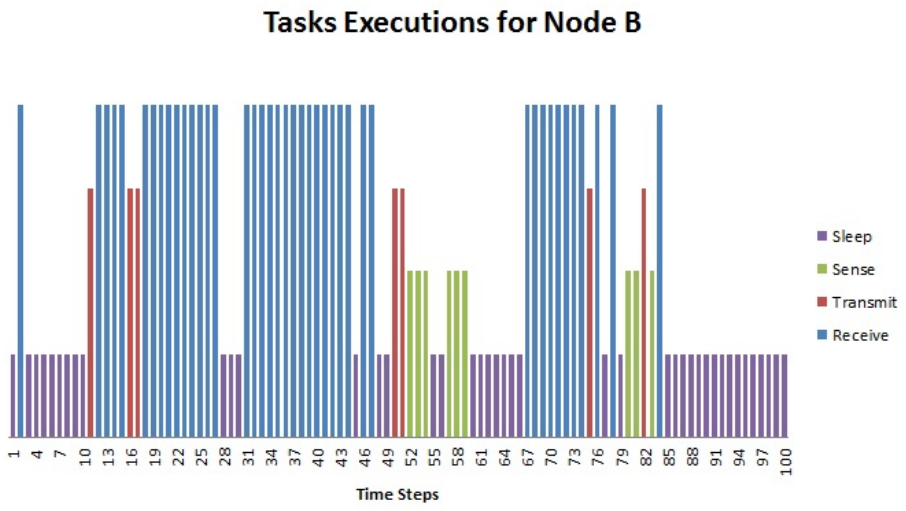


Figure 5.13: Tasks execution for Node B in Figure 4.1 by cooperative Q learning.

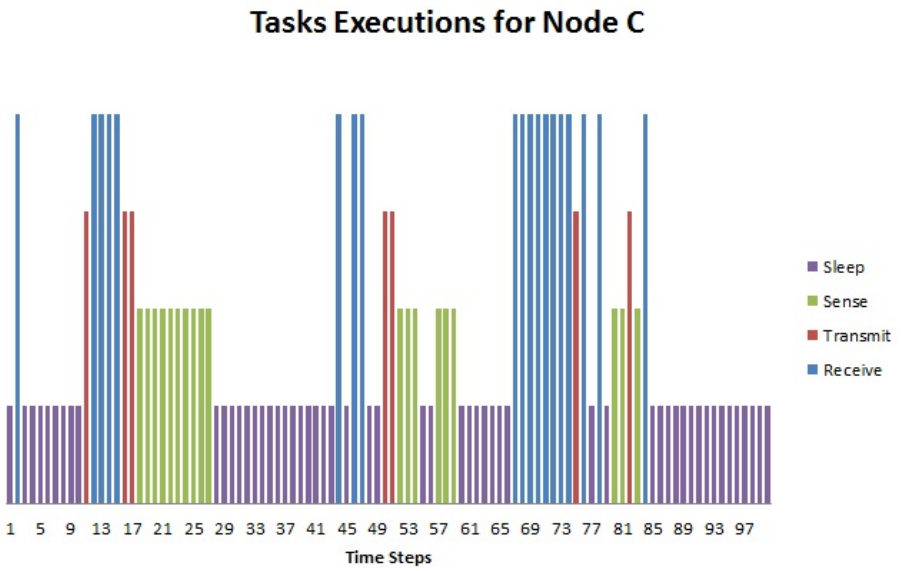


Figure 5.14: Tasks execution for Node C in Figure 4.1 by cooperative Q learning.

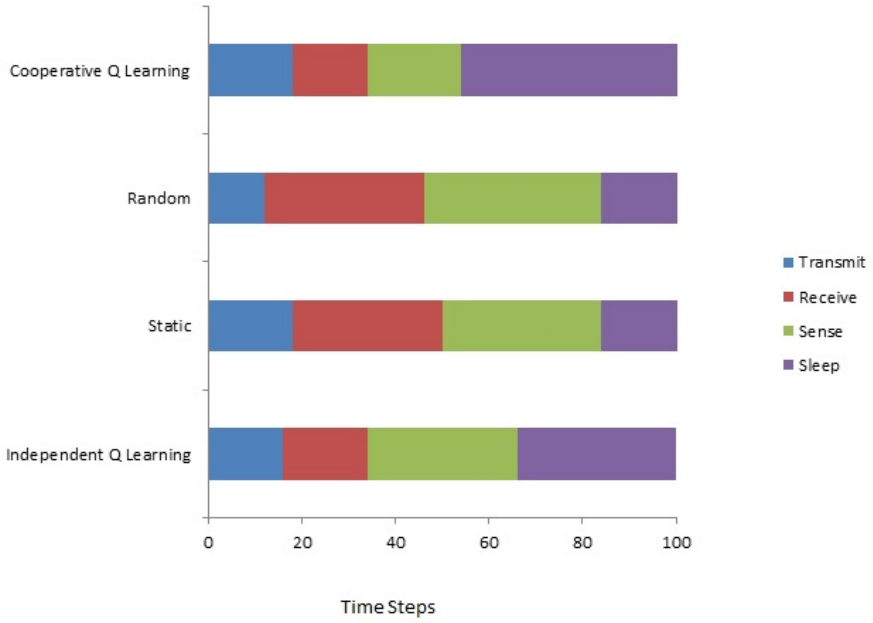


Figure 5.15: Total number of execution for each action.

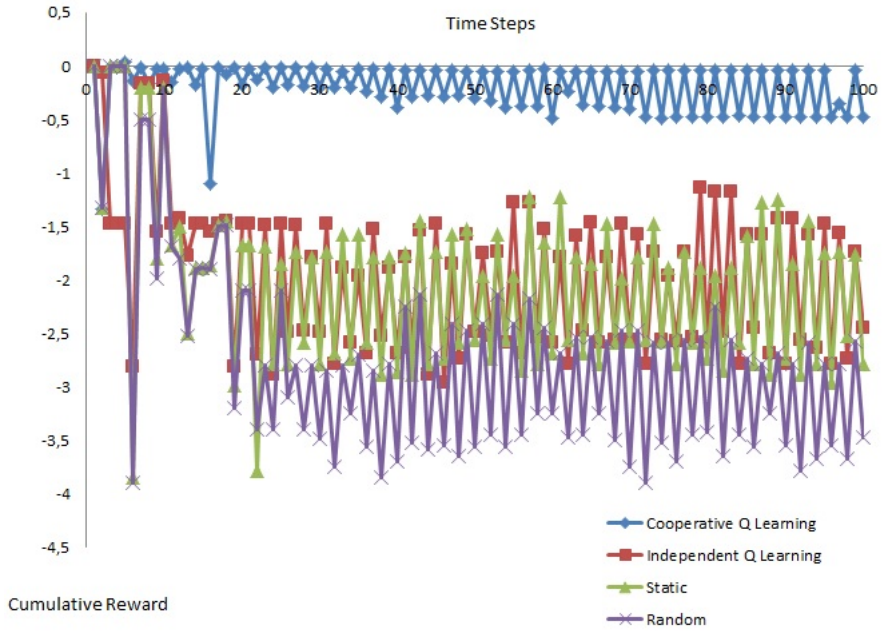


Figure 5.16: Cumulative reward over time by application scenario 2 (cp. Subsection 4.3.2).

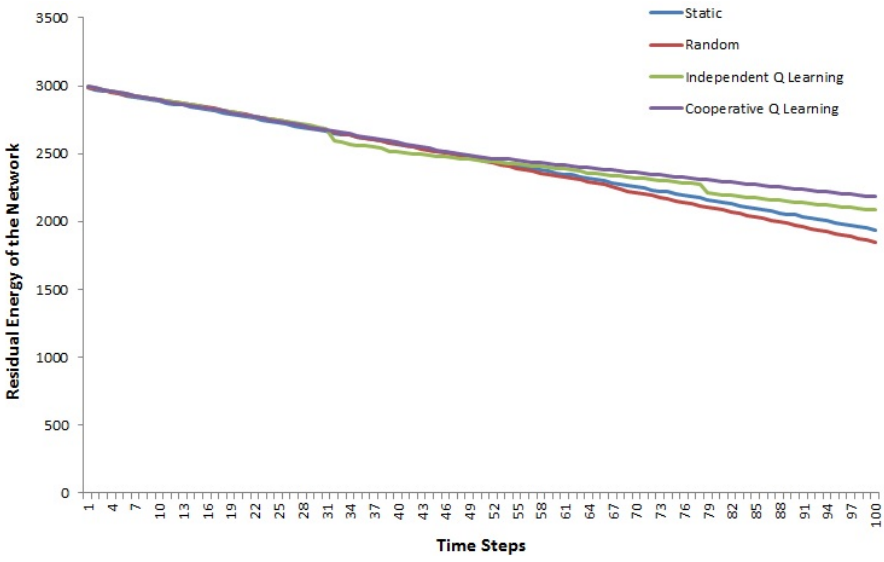


Figure 5.17: Residual energy of the network over time.

over time based on the utility of their tasks.

For example, node A does not know that it needs to sense as the object is in its field of view. Here each bar represents a task executed at each time step. We represent "Receive", "Transmit", "Sense", and "Sleep" tasks in descending order of height of the bars. We can observe that node A immediately learns that it is getting paid to sense as the object is in its field of view. In the middle of the simulation time, as the target is out of reach of all sensor nodes, all nodes will be rewarded for sleeping. Similarly, we can observe that for node C, it will be rewarded for sensing after some times when the object is in its field of view. After that when the object is out of reach of all sensor nodes, all the nodes will be rewarded for sleeping.

We also calculate the revenue of the network by NetLogo [87] simulator. Three agents are considered for this simulation. On each agent, the proposed algorithm is implemented. Figure 5.23 shows the cumulative revenue for the network with three agents. We can observe the cumulative revenue of the network converges after certain time periods. Here cumulative revenue is calculated by the summation of revenues of all nodes in the network (cp. Subsection 4.3.1 for more details). This revenue depends on the executed task of sensor nodes and their gained utilities. There are variations in cumulative revenue due to the dynamics of the network and the movement of the target.

From the simulation result and evaluation, we can say that by applying combinatorial auction based method it is possible to learn the usefulness of actions to perform in different states. Our proposed method helps to schedule the tasks in such a way that the energy consumption is reduced. It also receives better cumulative revenue over time compared with other approaches. The residual energy of the network also remains more in our proposed approach compared with other approaches.

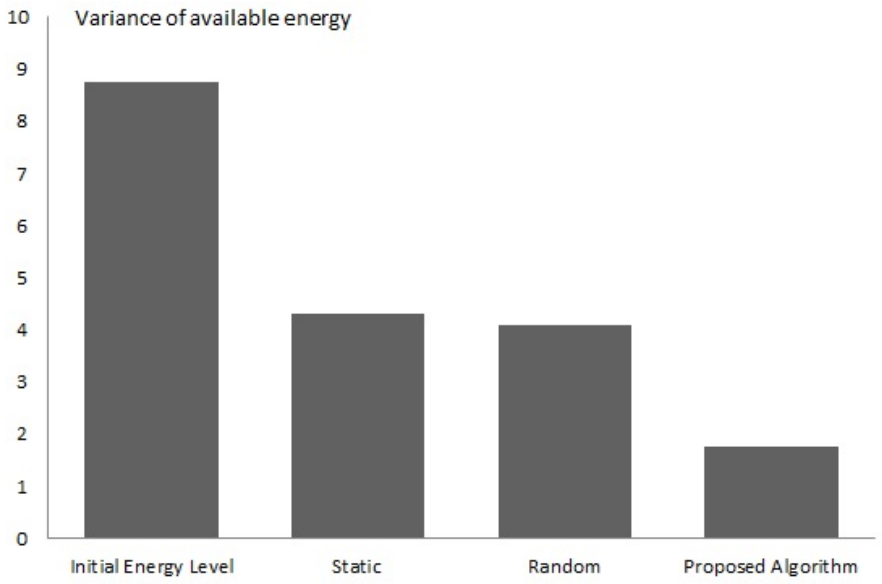


Figure 5.18: Variance of the available energy for different methods.

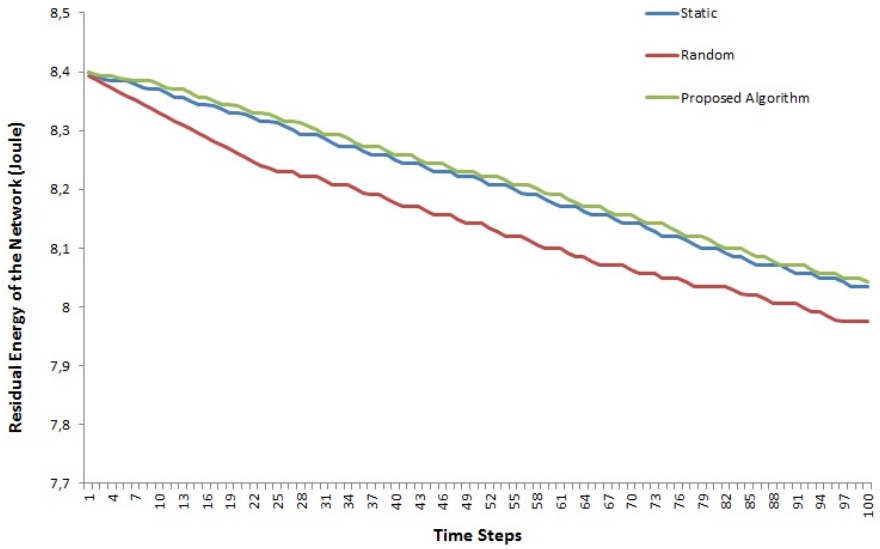


Figure 5.19: Residual energy of the network.

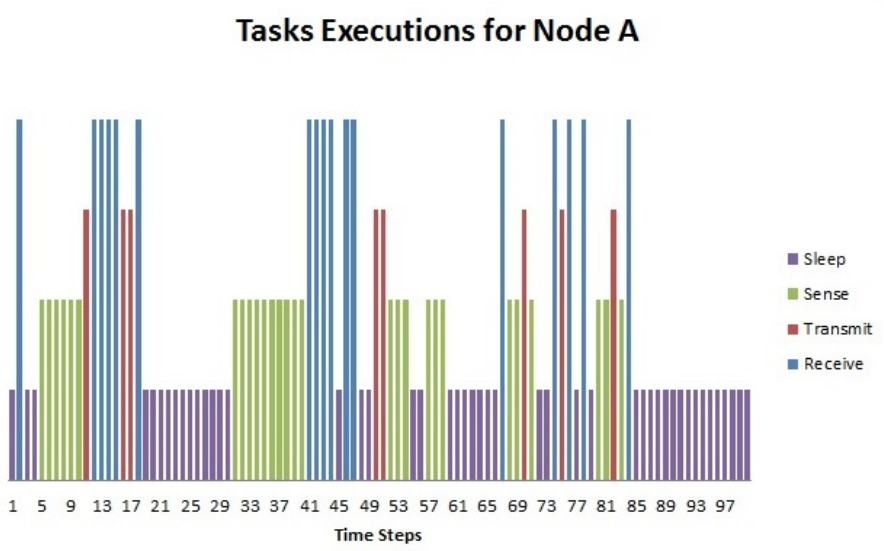


Figure 5.20: Tasks execution for Node A in Figure 4.1 by combinatorial auction method.

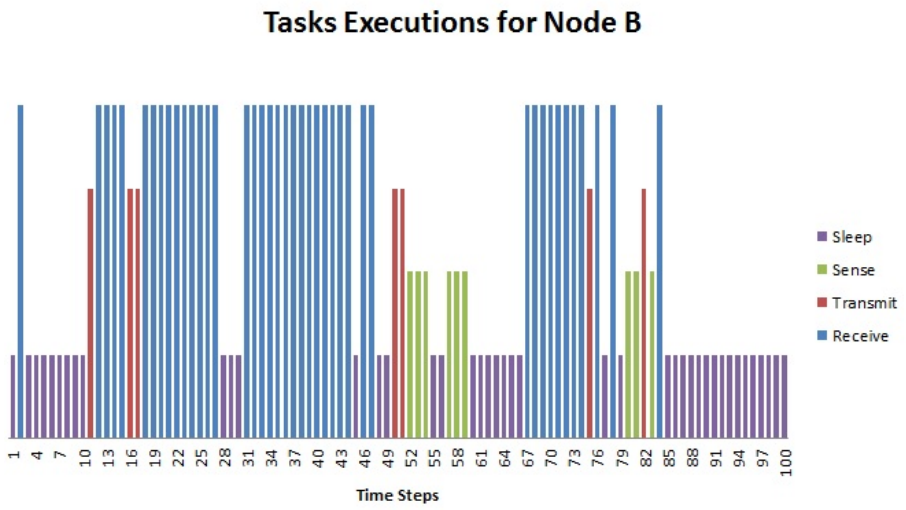


Figure 5.21: Tasks execution for Node B in Figure 4.1 by combinatorial auction method.

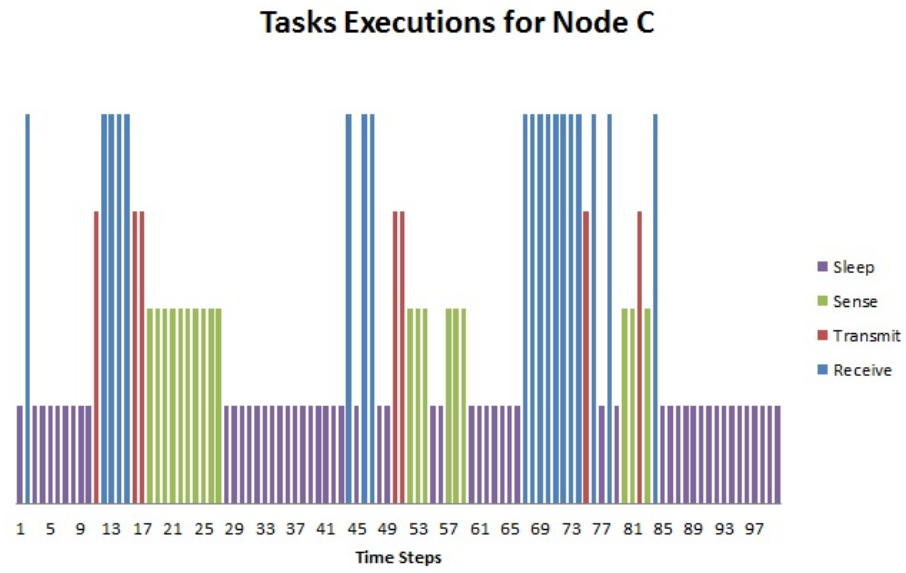


Figure 5.22: Tasks execution for Node C in Figure 4.1 by combinatorial auction method.

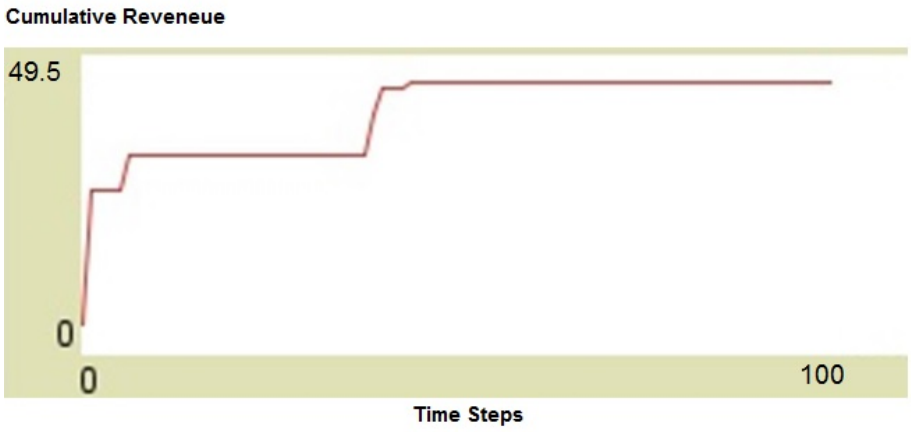


Figure 5.23: Cumulative revenue of the network with three agents.

5.4 Comparison of RL, CRL, and Exp3

We perform the evaluation of RL, CRL, and Exp3 considering the system model described in the Subsection 3.2.1. For our evaluation we perform the following five experiments with the following assumptions of parameters.

1. To find out the trade-off between tracking quality and energy consumption, we set the balancing factor β of the reward function between [0.1,0.9] in 0.1 steps, keep the randomness of moving targets as $\eta = 0.5$, set the egalitarianism factor of Exp3 as $\kappa = 0.5$ and fix the topology to five nodes.

2. We vary the network size to check the trade-off between tracking quality and energy consumption. We consider three different topologies consisting of 5, 10, and 20 sensor nodes where the coverage ratio is 0.0029, 0.0057, and 0.0113, respectively. The coverage ratio is defined as the ratio of the aggregated FOV of all deployed sensor nodes over the area of the entire surveillance area. We keep the balancing factor $\beta = 0.5$ and the randomness of the mobility model $\eta = 0.5$ constant for this experiment.

3. We set the randomness of moving targets η to one of the following values $\{0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.7, 0.9\}$ and set the balancing factor $\beta = 0.5$ and fix the topology to five nodes.

4. We vary the sensing range r to one of these values {10,15,20,25,30,35,40,45} for this experiment.

5. We evaluate RL, CRL, and Exp3 in terms of average execution time and average communication effort. These values are measured from twenty iterations and represent the mean execution times and the mean of $Send\_Message$ task executions.

Figure 5.24 shows the tracking quality/energy consumption trade-off for RL, CRL, and Exp3 by varying the balancing factor, β between [0.1,0.9] in 0.1 steps. Table 5.2 shows the evaluation of RL, CRL, and Exp3 based on tracking quality, energy consumption, and average communication effort varying the balancing factor β. Each data point in these figures represents the average of normalized tracking quality and energy consumption of ten complete simulation runs. We observe that CRL and Exp3 provide similar results, i.e., the corresponding data points are closely co-located. RL is more energy aware but is not able to achieve high tracking quality.

Figure 5.25 shows the tracking-quality/energy consumption trade-off varying the number of nodes to 5, 10, and 20. Table 5.3 shows the mean and variance values of the tracking quality and energy consumption for the RL, CRL, and Exp3 varying the number of nodes, N. In this experiment, each data point represents the average of normalized tracking quality and energy consumption of ten complete simulation runs. Here the same trend can be identified, i.e., the CRL and Exp3 achieve almost similar results in terms of tracking quality/energy consumption trade-off and RL shows less tracking performance with higher energy efficiency.

Figures 5.26, 5.27, and 5.28 show the results by varying η to one of these values $\{0.1,0.15,0.2,0.25,0.3,0.4,0.5,0.7$, and $0.9\}$. Each data point in these figures represents the average of normalized tracking quality and energy consumption of ten complete simulation runs. From these figures, it can be seen that CRL and Exp3 outperform RL in terms of achieved tracking performance. We can see that for lower randomness, η=0.5, 0.7, and 0.9, RL and Exp3 show very close results for tracking performance. But for higher randomness, η= 0.1, 0.15, and 0.2, RL gives poor performance with regard to tracking performance.

Figure 5.29 shows the results of tracking quality/energy consumption trade-off of RL, CRL, and Exp3 for different coverage ratio. The coverage ratio is defined as the ratio of the aggregated FOV of all deployed sensor nodes A_s over the area of the entire surveillance area A_n. Here, each data point represents average of the normalized tracking quality and energy consumption of ten complete simulation runs. We can observe that higher the A_s/A_n ratio, less the tracking quality. Higher A_s/A_n means more spaces are not covered by the considered sensor nodes and the sensing radius r of each sensor node is smaller. We vary r to one of these values $\{10,15,20,25,30,35,40,45\}$ for this experiment. The surveillance area is a rectangle with the height of 300 unit and width of 460 unit. Area of a particular node is considered as πr^2. A_s is the surveillance area considered as $(height \times width)$. From the figure, it can be seen that Exp3 outperforms the RL in terms of tracking quality. CRL shows higher tracking quality but poor performance with regard to energy consumption.

Table 5.4 shows the comparison of RL, CRL, and Exp3 in terms of average execution time and average communication effort. These values are derived from twenty iterations and represent the mean execution times and the mean of *Send_message* task executions. We find that RL is the most resource-aware scheduling algorithm. Exp3 requires 25% more and CRL requires 86% more execution time, respectively.

The communication overhead is similar for both Exp3 and CRL.

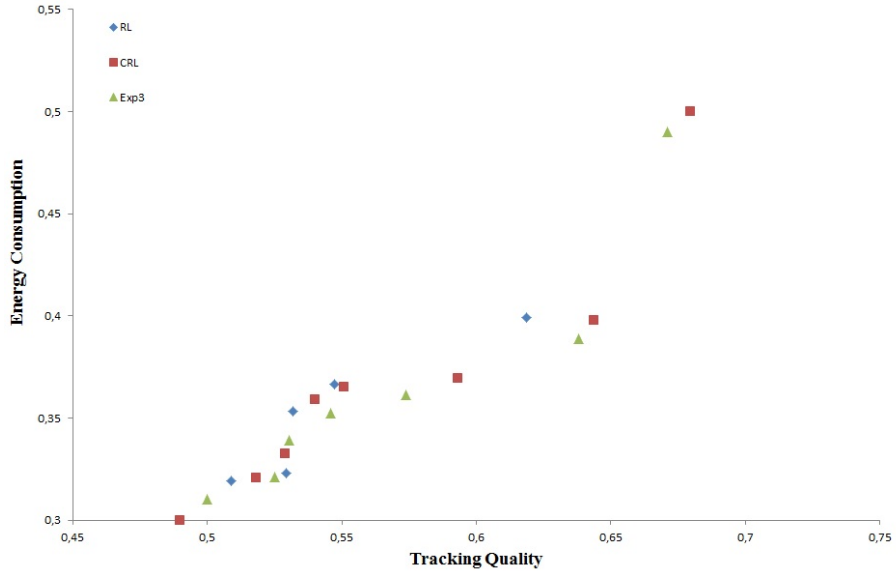


Figure 5.24: Tracking quality/energy consumption trade-off for RL, CRL, and Exp3 by varying the balancing factor of the reward function β.

	RL			CRL			Exp3		
	TQ	EC	ACE	TQ	EC	ACE	TQ	EC	ACE
β=0.1	0.43	3.90	0	0.49	4.00	21	0.48	3.96	18
β=0.2	0.45	3.92	0	0.51	4.20	23	0.50	4.10	20
β=0.3	0.48	3.99	0	0.52	4.33	26	0.52	4.20	23
β=0.4	0.50	4.18	0	0.54	4.59	28	0.53	4.39	25
β=0.5	0.52	4.23	0	0.55	4.65	30	0.54	4.52	28
β=0.6	0.53	4.53	0	0.59	4.69	32	0.57	4.61	30
β=0.7	0.54	4.67	0	0.64	4.97	33	0.64	4.89	32
β=0.8	0.62	4.99	0	0.68	6.00	35	0.67	5.89	34
β=0.9	0.74	5.19	0	0.77	6.50	37	0.76	6.39	36

Table 5.2: Comparison of RL, CRL, and Exp3 based on Tracking Quality (TQ), Energy Consumption (EC), and Average Communication Effort (ACE) by varying the balancing factor of the reward function β.

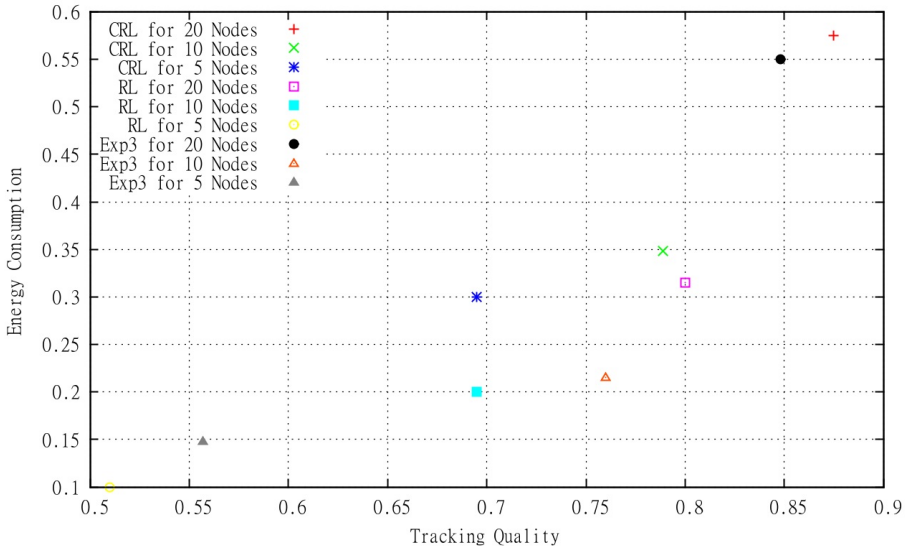


Figure 5.25: Tracking quality/energy consumption trade-off for RL, CRL, and Exp3 by varying the network size.

	RL		CRL		Exp3	
	TQ	EC	TQ	EC	TQ	EC
Mean (N=5)	0.51	4.10	0.54	5.20	0.54	4.45
Variance (N=5)	0.0016	0.26	0.0012	0.23	0.0011	0.21
Mean (N=10)	0.69	5.10	0.78	6.01	0.78	5.49
Variance (N=10)	0.0016	0.26	0.0012	0.23	0.0011	0.21
Mean (N=20)	0.81	6.49	0.87	6.99	0.85	6.75
Variance (N=20)	0.0014	0.09	0.0013	0.07	0.0010	0.07

Table 5.3: Mean and variance of the tracking quality and the energy consumption by varying the number of nodes, N=5, 10, and 20. Here TQ means Tracking Quality and EC means Energy Consumption.

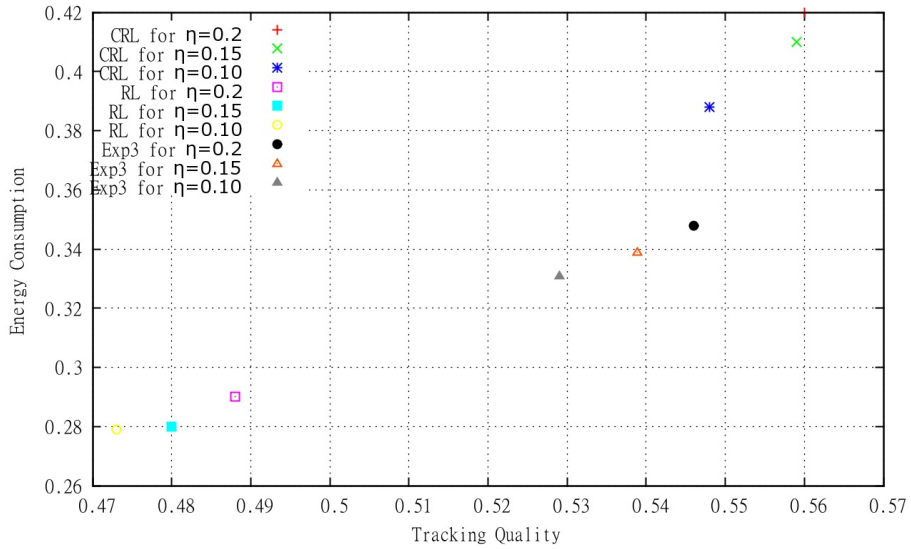


Figure 5.26: Tracking quality/energy consumption trade-off for RL, CRL, and Exp3 by varying the randomness of target movement, $\eta = 0.10, 0.15$, and 0.20.

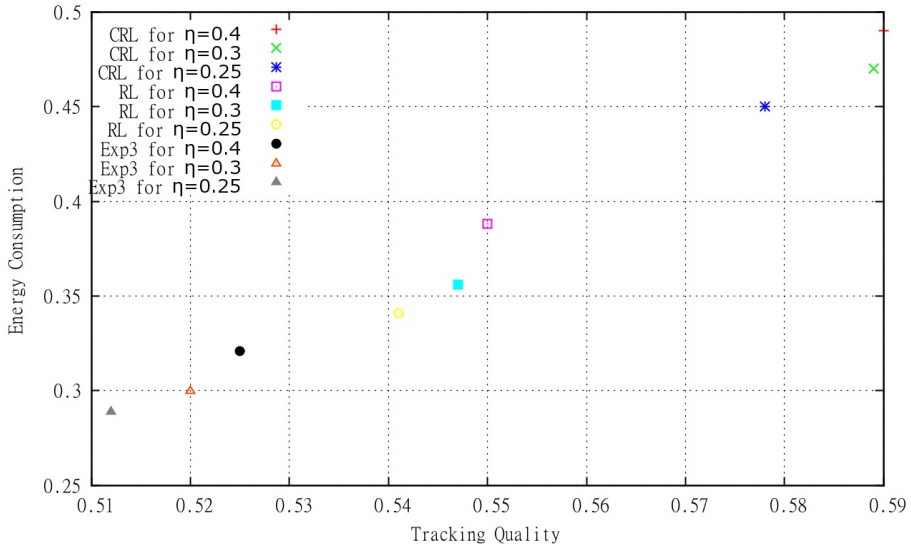


Figure 5.27: Tracking quality/energy consumption trade-off for RL, CRL, and Exp3 by varying the randomness of target movement, $\eta = 0.25, 0.30$, and 0.40.

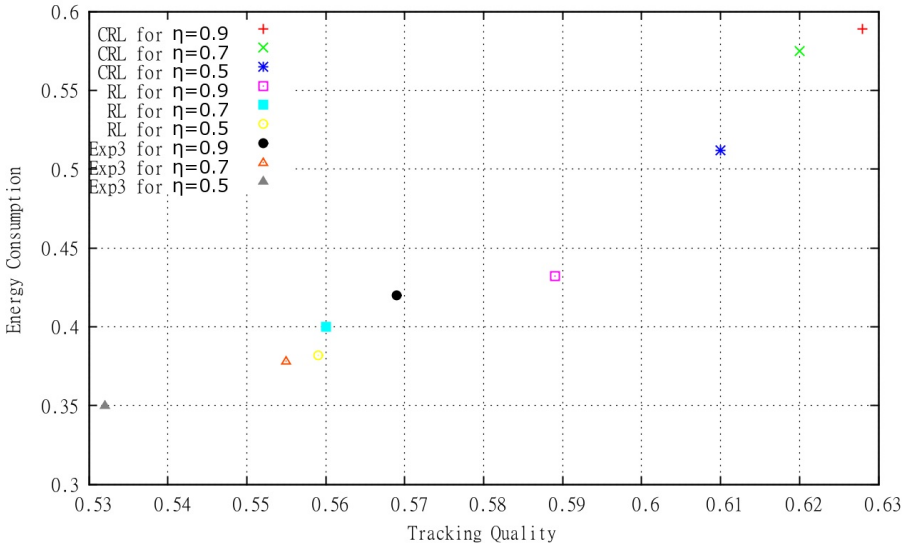


Figure 5.28: Tracking quality/energy consumption trade-off for RL, CRL, and Exp3 by varying the randomness of target movement, $\eta = 0.50, 0.70$, and 0.90.

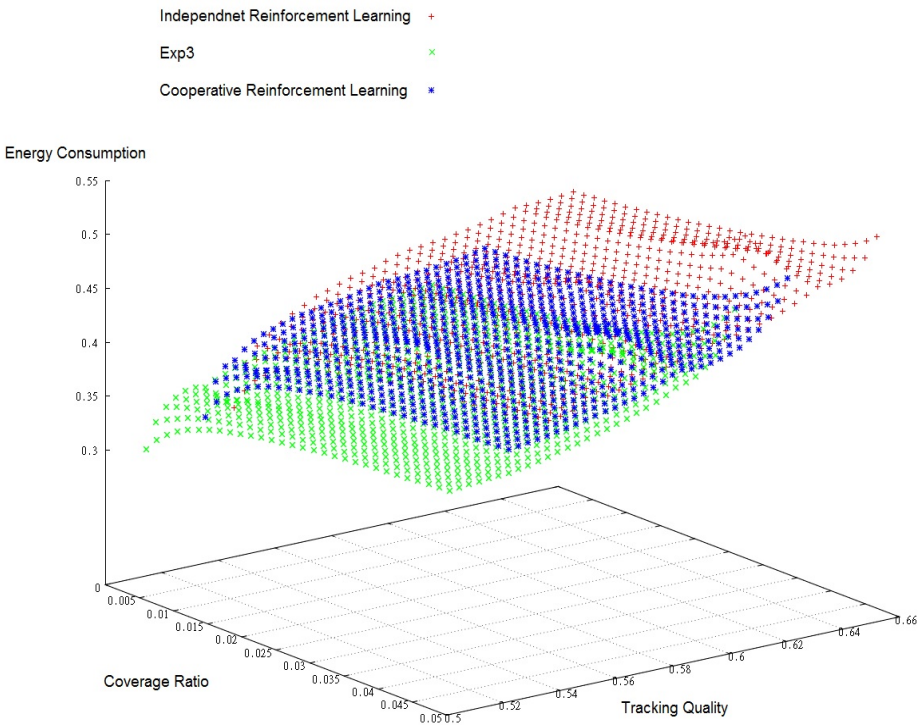


Figure 5.29: Results varying the sensing radius

	Avg. Execution Time	Avg. Comm. Effort
RL	0.036 s	0
CRL	0.067 s	29
Exp3	0.045 s	27

Table 5.4: Comparison of average execution time and average number of transferred messages (based on 20 iterations).

5.5 Discussion

All above results help us to reach the following decisions.

- Cooperative task scheduling methods (CRL (one hop and two hop), Cooperative Q learning, Exp3) provide better performance, e.g., tracking quality than non-cooperative or independent task scheduling method (RL).
- Cooperative approaches require more resources, e.g., energy due to the increase communication effort.
- By appropriately setting the balancing factor of the reward function β the desired performance or energy consumption can be achieved.
- Further cooperation among neighboring nodes with the local observations help to achieve better performance. CRL(two hop) outperforms the RL and CRL(one hop) in terms of tracking quality.
- Online task scheduling methods outperform the offline task scheduling methods in terms of remaining energy of the network.
- Our evaluation shows that different methods have different properties concerning resource-awareness and achieved performance. The selection of the scheduling algorithm thus depends on the application requirements and available resources of the WSN.

CHAPTER

6 Conclusion and future work

Resource-aware effective task scheduling is very important for WSN to know the best task to execute on next time slots. We propose two cooperative reinforcement learning methods (cooperative Q learning and SARSA(λ)), bandit solvers (Exp3), and market based method for online scheduling of tasks in a way that the better energy/performance trade-off is achieved. We perform the evaluation of our proposed cooperative methods (one hop and two hop distance neighbors) and non-cooperative method. Our experimental results show that our cooperative scheduling outperforms the non-cooperative scheduling in terms of tracking quality. We show that cooperation among neighboring nodes improves the energy/performance trade-off comparing with the non-cooperative approach. We also perform the evaluation of RL, CRL, and Exp3. We apply our methods in a target tracking application. We find that RL, CRL, and Exp3 have different properties based on tracking quality and energy efficiency. The selection of a particular scheduling algorithm depends on the requirement of the application and available resources of sensor nodes.

6.1 Summary of contributions

We propose cooperative Q learning for task scheduling in a WSN. This is our first initiative to apply a reinforcement learning method for task scheduling. We assume a set of tasks and a set of states for applying Q learning. To our best knowledge, this is the first work to consider some cooperation among neighbors for task scheduling in a WSN. We consider static values for the reward of the performed actions. We provide higher reward values for the tasks which consume less energy for the execution. We apply this method in a target tracking application and compare with static, random and existing individual reinforcement learning method. We consider only three nodes with a particular movement of a target. We consider two application scenarios with two sets of reward values. We calculate the cumulative reward over time for all methods. We find task execution at each time step considering three sensor nodes. We calculate the total number of executions for each action and also the residual

energy of the network over time. We find that our proposed cooperative learning outperforms the existing approaches in terms of energy efficiency.

We propose a combinatorial auction-based method for task scheduling in a WSN. We have a set of tasks. Each task requires CPU cycle, energy requirements, and ideal sleep time for execution. We calculate the variance of the available energy of the sensor nodes. The higher variance value corresponds to the unbalanced available energy among the sensor nodes which is not energy efficient. We calculate the residual energy of the network and perform the evaluation of our proposed approach with the existing static and random task scheduling. We find out the task execution for each considered node over time steps. We also calculate the revenue of the network by a simulator. We observe that our proposed approach outperforms in terms of energy efficiency and task scheduling comparing with static and random task scheduling.

We propose a cooperative SARSA(λ) learning algorithm for task scheduling in a WSN. For this work, we update our state and action space with the trajectory prediction action and the awareness state. We consider a weighted reward function to trade the application performance and energy consumption. We perform the experiments to find out the trade-off between application performance and resource consumption by varying the balancing factor of the reward function. We also vary the randomness of the target movement and the number of nodes to identify the performance/resource consumption trade-off. We observe that proposed cooperative approach outperforms the non-cooperative approach in terms of tracking quality.

We propose an exponential bandit solvers Exp3 for task scheduling in WSN. We perform the evaluation of Exp3 with RL and CRL. We find the tracking quality/energy consumption trade-off for methods. We observe that Exp3 and CRL provide close results in terms of tracking quality/energy consumption trade-off. RL provides the most energy efficiency but less tracking quality. We also calculate the average execution time and communication overhead for these methods. Here we find that CRL and Exp3 provide close results in terms of execution time and communication overhead.

We observe that cooperation among neighboring nodes helps to achieve better performance. Cooperative task scheduling methods outperform the non-cooperative or independent task scheduling methods in terms of tracking quality. Independent scheduling methods provide better energy efficiency compared with cooperative task scheduling as there is no message exchange for the cooperation in independent learning methods. Online task scheduling methods help to learn the best next task to execute based on past observed application behavior. We find that our proposed methods have different properties based on resource-awareness and achieved performance. The selection of the appropriate method depends on the application demand and the remaining resources of the WSN.

6.2 Future works

This thesis opens up so many issues based on resource-aware task scheduling in WSNs. To find suitable methods for better application performance with optimized resource consumption is one of the challenges. Future work includes the application of our resource-aware scheduling approach to different WSN applications, the implementation on our visual sensor network platforms [24] and the comparison of our approach with other variants of reinforcement learning methods.

Currently we are using our task scheduling methods for a multi-target tracking application. There are some other applications, e.g., routing protocols, medium access control (MAC) protocols, etc. There are some existing works of RL-based routing protocol design to find optimal routing paths [22] [25] [83]. There are some existing RL-based MAC protocols [15] [56]. One of our future works is to apply our proposed CRL for the routing protocol and MAC protocol design.

We have a plan to apply our proposed task scheduling methods on our visual sensor network platforms [24]. Currently we are considering sensor nodes with limited energy supply in a rectangular area for moving target tracking application.

We plan to compare our proposed RL method with the other variants of RL. Currently we are using cooperative Q and $SARSA(\lambda)$ learning. There are some other variants of RL like $Q(\lambda)$, temporal difference (TD), $TD(\lambda)$ and $SARSA$ [80]. We plan to evaluate our proposed methods with $Q(\lambda)$, TD, $TD(\lambda)$, and $SARSA$.

Bibliography

[1] T. Abbes, S. Mohamed, and K. Bouabdellah. Impact of Model Mobility in Ad Hoc Routing Protocols. *Computer Network and Information Security*, 10:47–54, 2012.

[2] K. Akkaya and M. Younis. A survey of routing protocols in wireless sensor networks. *Ad Hoc Network (Elsevier)*, 3(3):325–349, 2005.

[3] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. A survey on wireless multimedia sensor networks. *Computer Networks*, 51:921–960, 2007.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Elsevier Computer Networks*, 38:393–422, 2002.

[5] T. Arampatzis, J. Lygeros, and S. Manesis. A survey of applications of wireless sensors and wireless sensor networks. In *Proceedings of IEEE Mediterrean Conference on Control and Automation*, pages 719–724, 2005.

[6] P. Auer, N. C. Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32:48–77, 2002.

[7] S. Ben-david, E. Kushilevitz, and Y. Mansour. Online learning versus offline learning. *Computational Learning Theory*, 904:38–52, 1995.

[8] N. Bery. Linear regression. Technical report, DataGenetics, 2009.

[9] A. Bharathidasan, V. Anand, and S. Ponduru. Sensor networks: An overview. In *IEEE Potentials*, volume 22, pages 20–23, 2003.

[10] A. Blum. On-line algorithms in machine learning. In *Proceedings of the Workshop on On-Line Algorithms*, pages 306–325, 1996.

[11] O. Boyinbode, H. Le, A. Mbogho, M. Takizawa, and R. Poliah. A survey on clustering algorithms for wireless sensor networks. In *International Conference on Network-Based Information Systems*, pages 358–364, 2010.

[12] M. Bramwell. Implementing a mica2 mote sensor network. Technical report, Crossbow-Mica2 Mote, 2006.

[13] J. Byers and G. Nasser. Utility Based Decision making in Wireless Sensor Networks. In *Proceedings of the Workshop on Mobile and Ad Hoc Networking and Computing*, pages 143 – 144, 2000.

[14] L. Chen, B. K. Szymanski, and J. W. Branch. Auction-based congestion management for target tracking in wireless sensor networks. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, pages 1–10, 2011.

[15] Y. Chu, P. Mitchell, and D. Grace. Aloha and q-learning based medium access control for wireless sensor networks. In *International Symposium on Wireless Communication Systems*, pages 511–515, 2012.

[16] B. Coltin and M. Veloso. Mobile robot task allocation in hybrid wireless sensor networks. In *Proceedings on IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2932–2937, 2010.

[17] D. Culler, D. Estrin, and M. Srivastava. Overview of sensor networks. *IEEE Computer Society*, pages 41–49, 2004.

[18] W. Dargie and C. Poellabauer. *Fundamentals of Wireless Sensor Networks*. Wiley, 2010.

[19] R. Dechter and D. Frost. Backtracking algorithms for constraint satisfaction problems. Technical report, University of California, Irvine, 1999.

[20] S. Denef, L. Ramirez, and T. Dyrks. Letting tools talk: Interactive technology for firefighting. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, pages 4447–4452, 2009.

[21] S. Dhanani, J. Arseneau, A. Weatherton, B. Caswell, N. Singh, and S. Patek. A comparison of utility based information management policies in sensor networks. In *Systems Technology Integration Lab of the Department of Systems and Information Engineering at the University of Virginia, Charlottesville, VA.*, pages 84–89, 2006.

[22] S. Dong, P. Agrawal, and K. Sivalingam. Reinforcement learning based geographic routing protocol for uwb wireless sensor network. In *IEEE Global Telecommunications Conference*, pages 652–656, 2007.

[23] N. Edalat, W. Xiao, N. Roy, S. Das, and M. Motani. Combinatorial auction based task allocation in multi application wireless sensor networks. In *Proceedings of the international conference on Embedded and Ubiquitous Computing*, pages 174–181, 2011.

[24] L. Esterle, P. R. Lewis, X. Yao, and B. Rinner. Socio-Economic Vision Graph Generation and Handover in Distributed Smart Camera Networks. *ACM Transactions on Sensor Networks*, 10(2):24, 2014.

[25] A. Förster and A. Murphy. Froms: Feedback routing for optimizing multiple sinks in wsn with reinforcement learning. In *IEEE International Conference on Intelligent Sensors, Sensor Networks and Information*, pages 371–376, 2007.

[26] C. Frank and K. Römer. Algorithms for Generic Role Assignments in Wireless Sensor Networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*, pages 230–242, 2005.

[27] A. Galstyan. Resource allocation and emergent coordination in wireless sensor networks. In *Workshop on Sensor Networks at the The Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 1–6, 2004.

[28] Z. Ghahramani. Unsupervised learning. In *Advanced Lectures on Machine Learning*, pages 72–112. Springer-Verlag, 2004.

[29] S. Giannecchini, M. Caccamo, and C. Shih. Collaborative Resource Allocation in Wireless Sensor Networks. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 35–44, 2004.

[30] A. Goldsmith and S. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communications Magazine*, 9:8–27, 2002.

[31] W. Guo, N. Xiong, H.-C. Chao, S. Hussain, and G. Chen. Design and Analysis of Self Adapted Task Scheduling Strategies in WSN. *Sensors*, 11:6533–6554, 2011.

[32] A. Hac. *Wireless Sensor Network Designs*. John Wiley and Sons, 2003.

[33] V. Handziski, A. Kopk, H. Karl, and A. Wolisz. A common wireless sensor network architecture? Technical report, Technical University of Berlin, 2003.

[34] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174–185, 1999.

[35] J. Huang, Z. Han, M. Chiang, and H. V. Poor. Auction-based resource allocation for cooperative communications. *IEEE Journal on Selected Areas in Communications*, pages 1226–1237, 2008.

[36] M. M. Islam, M. M. Hassan, G. W. Lee, and E. N. Huh. A survey on virtualization of wireless sensor networks. *Sensors*, 12(2):2175–2207, 2012.

[37] D. Jea and M. B. Srivastava. Channels characteristics for on-body mica2dot wireless sensor networks. Technical report, Networked and Embedded Systems Laboratory, University of California, Los Angeles, 2003.

[38] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[39] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Emerging challenges: Mobile networking for smart dust. *Journal of Communications and Networks*, 2(3), 2000.

[40] A. E. Kateeb, A. Ramesh, and L. Azzawi. Wireless sensor nodes processor architecture and design. In *Canadian Conference on Electrical and Computer Engineering*, pages 1031–1034, 2008.

[41] 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*, pages 895 – 900, 2012.

[42] M. I. Khan and B. Rinner. Energy-Aware Task Scheduling in Wireless Sensor Networks by Cooperative Reinforcement Learning. In *Proceedings of the IEEE International Conference on Communications Workshops*, pages 871–877, 2014.

[43] M. I. Khan and B. Rinner. Evaluation of Resource-Aware Task Scheduling Methods in Wireless Sensor Networks. *IEEE Sensor Journal*, page 2, 2014.

[44] M. I. Khan and B. Rinner. Performance Analysis of Resource-Aware Task Scheduling Methods in Wireless Sensor Networks. *Hindawi International Journal of Distributed Sensor Networks*, page 9, 2014.

[45] M. I. Khan, B. Rinner, and C. S. Regazzoni. Resource Coordination in Wireless Sensor Networks by Combinatorial Auction Based Method. In *Proceedings of the IEEE International Conference on Networked Embedded Systems for Every Applications*, pages 1–6, 2012.

[46] U. A. Khan and B. Rinner. Online Learning of Timeout Policies for Dynamic Power Management. *ACM Transactions on Embedded Computing Systems*, page 25, 2013.

[47] C. Kim and J. Ahn. Causally ordered delivery protocol for overlapping multicast groups in broker-based sensor networks. *International Journal of Computer Science Issues, Vol 8, Issue 1*, pages 46–54, 2011.

[48] L. Kleinrock. Distributed systems. *Communications of the ACM*, 28(11), 1985.

[49] P. Klemperer. Auction theory: A guide to the literature. *Journal of Economic Surveys*, 13(3):227–286, 1999.

[50] J. Ko, K. Klues, C. Richter, M. B. Wanja Hofer, Branislav Kusy, T. Schmid, Q. Wang, P. Dutta, and A. Terzis. Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone). In *Proceedings of European Conference on Wireless Sensor Networks*, pages 98–114, 2012.

[51] S. Kogekar, E. Neema, O. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti. Constraint-guided dynamic reconfiguration in sensor networks. In *Information processing in sensor networks*, pages 379–387. ACM Press, 2004.

[52] H. Kopetz. *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Springer, 2011.

[53] B. Krishnamachari and S. Wicker. On the complexity of distributed self-configuration in wireless networks. *Journal of Telecommunication Systems*, pages 33–59, 2003.

[54] J. E. Lee, S. H. Cha, D. Y. Kim, and K. H. Cho. Autonomous management of large-scale ubiquitous sensor networks. *Emerging Directions in Embedded and Ubiquitous Computing*, pages 609–618, 2006.

[55] T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118. ACM Press, 2003.

[56] Z. Liu and I. Elhanany. Rl-mac: A reinforcement learning based mac protocol for wireless sensor networks. *International Journal of Sensor Networks*, 1(3):117–124, 2006.

[57] T. A. Malik. *Target Tracking in Wireless Sensor Networks*. PhD thesis, Maharshi Dayanand University, 2003.

[58] D. C. Montgomery, E. A. Peck, and G. G. Vining. *Introduction to Linear Regression Analysis*. Wiley, 2007.

[59] J. F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 36, pages 48–49, 1950.

[60] H. Park and M. B. Srivastava. Energy-efcient task assignment framework for wireless sensor networks. Technical report, University of California, Los Angeles, 2012.

[61] M. A. Perillo and W. B. Heinzelman. Wireless sensor network protocols. Technical report, University of Rochester, 2005.

[62] C. S. Raghavendra, K. M. Sivalingam, and T. Znati. *Wireless Sensor Networks*. Springer, 2006.

[63] M. L. Rasneet Kaur. Wireless Sensor Networks: A Survey. *International Journal of Electronics and Communication Technology*, 4(3):102–106, 2013.

[64] Z. Ruyan, L. Huifang, H. Shijun, and W. Dongyun. Data processing and node management in wireless sensor network. In *International Symposium on Computer Network and Multimedia Technology*, pages 1–4, 2009.

[65] M. G. Sandra Sendra, Jaime Lloret and J. F. Toledo. Power saving and energy optimization techniques for wireless sensor networks. *Journal of communications*, 6:439–459, 2011.

[66] K. Shah and M. Kumar. Distributed Independent Reinforcement Learning (DIRL) Approach to Resource Management in Wireless Sensor Networks. In *Proceedings of IEEE Mobile Adhoc and Sensor Systems*, pages 1–9, 2007.

[67] K. Shah and M. Kumar. Resource Management in Wireless Sensor Networks Using Collective Intelligence. In *Proceedings of Intelligent Sensors, Sensor Networks and Information Processing*, pages 423–428, 2008.

[68] J. Shanbehzadeh, S. Mehrjoo, and A. Sarrafzadeh. An intelligent energy efficient clustering in wireless sensor networks. In *Proceedings of the international multiconference of engineers and computer scientists*, volume 1, pages 1–5, 2011.

[69] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pages 160–171, 2002.

[70] H. Shih, X. Zhang, D. S. L. Wei, K. Naik, and R. C. Chen. Design and implementation of a mobile sensor network testbed using sun spots. *Journal of Future Computer and Communication*, 2(2):115–120, 2013.

[71] S. S. Shwartz. *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, Senate of the Hebrew University, 2007.

[72] A. Silberschatz, G. Gagne, and P. Galvin. *Operating system concepts*. Addison-Wesley, 2008.

[73] P. Sivaram and S. Angadi. Wireless sensor networks: Routing protocols, challenges, solutions. *Interantional Journal of P2P Network Trends and Technology (IJPTT)*, 3(4):214–217, 2013.

[74] K. Sohraby, D. Minoli, and T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley, 2007.

[75] E. Soulas and D. Shasha. Online machine learning algorithms for currency exchange prediction. Technical report, NYU CS Technical Report TR-2013-953, 2013.

[76] M. Spiegel. *Theory and Problems of Probability and Statistics*. McGraw-Hill, 1992.

[77] W. Stallings. *Operating Systems*. Pearson Education, 2006.

[78] D. J. Stein. Wireless sensor network simulator v1.0, March 2005.

[79] I. Stojmenovic. *Handbook of Sensor Networks, Algorithms and Architectures*. Wiley, 2010.

[80] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[81] Y. Tian, E. Ekici, and F. Ozguner. Energy-constrained Task Mapping and Scheduling in Wireless Sensor Networks. In *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, pages 210–218, 2005.

[82] N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling large scale storage in sensor networks with coffee file system. In *International Conference on Information Processing in Sensor Networks*, pages 349–360, 2009.

[83] R. A. Valles, R. A. Rodriguez, A. G. Curieses, and J. C. Sueiro. Q-probabilistic routing in wireless sensor networks. In *IEEE International Conference on Intelligent Sensors, Sensor Networks and Information*, pages 1–6, 2007.

[84] J. M. Vidal. Multiagent coordination using a distributed combinatorial auction. In *Proceedings of the AAAI Workshop on Auction Mechanisms for Robot Coordination*, pages 1–7, 2006.

[85] C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, pages 279–292, 1992.

[86] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *USENIX Conference on Operating Systems Design and Implementation*, pages 13–23, 1994.

[87] U. Wilensky. NetLogo. Technical report, Center for connected learning and computer based modeling, Northwestern University, Evanston, IL, 1999.

[88] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Annual Symposium on Foundations of Computer Science*, pages 374–382, 1995.

[89] K. L. A. Yau, P. Komisarczuk, and P. D. Teal. Reinforcement Learning for Context Awareness and Intelligence in Wireless Networks: Review, New Features and Open Issues. *Journal of Network and Computer Applications*, 35:253–267, 2012.

[90] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Journal of Computer Networks*, 52(12):2292–2330, 2008.

[91] E. Yoneki and J. Bacon. A survey of wireless sensor network technologies: research trends and middleware's role. Technical Report UCAM-CL-TR-646, University of Cambridge, Computer Laboratory, Sept. 2005.