REINFORCEMENT LEARNING BASED STRATEGIES FOR ADAPTIVE WIRELESS SENSOR NETWORK MANAGEMENT

by

KUNALBHAI SHAH

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2010

To my lovely daughter Deetya,

my wife Parinda

and

my parents

ACKNOWLEDGEMENTS

I am deeply indebted to my supervisor Dr. Mohan Kumar for his constant encouragement, motivation, advice and all the persistent help and guidance that made completion of my dissertation possible. I would also like to extend my heartfelt gratitude to Dr. Manfred Huber for his insightful advice and suggestions. Courses taken under Dr. Huber have provided major inspiration and ideas for my work. I would like to thank my other committee members, Dr. Gergely Zaruba, Dr. Yonghe Liu and Dr. Donggang Liu, for taking time to serve in my committee.

I thank my friends and fellow members at PICO for their valuable comments and advice. I am thankful to all the professors at UT Arlington under whom I took various courses that provided me knowledge to pursue my doctoral studies. I would also like to thank SensorLogic Inc. and Securus Technologies Inc. for their financial support during my graduate study.

Finally, I would like to express my deepest gratitude to my wife, Parinda, whose love, perseverence, and patience have been a constant source of sustenance and inspiration for me. I am indebted to my parents for their enormous support and love that went a long way in sustaining me through my doctoral studies.

November 22, 2010

ABSTRACT

REINFORCEMENT LEARNING BASED STRATEGIES
FOR ADAPTIVE WIRELESS SENSOR
NETWORK MANAGEMENT

KUNALBHAI SHAH, Ph.D.

The University of Texas at Arlington, 2010

Supervising Professor: Mohan Kumar

In wireless sensor networks (WSN), resource-constrained nodes are expected to operate in highly dynamic and often unattended environments. WSN applications need to cope with such dynamicity and uncertainty intrinsic in sensor networks, while simultaneously trying to achieve efficient resource utilization. A middleware framework with support for autonomous, adaptive and distributed sensor management, can simplify development of such WSN applications. We present a reinforcement learning based WSN middleware framework to enable autonomous and adaptive applications with support for efficient resource management. The uniqueness of our framework lies in using a bottom-up approach where each sensor node is responsible for its resource allocation/task selection while ensuring optimization of system-wide parameters like total energy usage, network lifetime etc. The framework allows creation of a distributed and scalable system while meeting applications' goals.

In this dissertation, a Q-learning based scheme called DIRL (Distributed Independent Reinforcement Learning) is presented first. DIRL learns the utility of per-

forming various tasks over time with mostly local information at nodes. DIRL uses these utility values along with application constraints for task management subject to optimal energy usage. DIRL scheme is extended to create a two-tier reinforcement learning based framework consisting of *micro-learning* and *macro-learning*. Micro-learning enables individual sensor nodes to self-schedule their tasks using local information allowing for a real-time adaptation as in DIRL. Macro-learning governs the micro-learners by setting their utility functions in order to steer the system towards applications' optimization goal (e.g. maximize network lifetime etc). The effectiveness of our framework is exemplified by designing a tracking/surveillance application on top of it. Finally, results of simulation studies are presented that compare performance of our scheme against other existing approaches. In general for applications requiring autonomous adaptation, our two-tier reinforcement learning based scheme on average is about 50% more efficient than micro-learning alone and many-fold more efficient than traditional resource management schemes like static scheduling, while maintaining necessary accuracy/performance.

Efficient data collection in sparse WSNs by special nodes called Mobile Data Collectors (MDCs) that visit sensor nodes is investigated. As contact times are not known a priori and in order to minimize energy consumption, the discovery of an incoming MDC by the static sensor node is a critical task. Discovery is challenging as MDCs participating in various applications exhibit different mobility patterns and hence require unique design of a discovery strategy for each application. In this context, an adaptive discovery strategy is proposed that exploits the DIRL framework and can be effectively applied to various applications while minimizing energy consumption. The principal idea is to learn the MDC's arrival pattern and tune the sensor node's duty cycle accordingly. Through extensive simulation analysis, the energy efficiency and effectiveness of the proposed framework is demonstrated.

Finally, design and evaluation of a complete and generalized middleware framework called DReL is presented with focus on distributed sensor management on top of our multi-layer reinforcement learning scheme. DReL incorporates mechanisms and communication paradigms for task, data and reward distributions. DReL provides an easy-to-use interface to application developers for creating customized applications with specific QoS and optimization requirements. Adequacy and efficiency of DReL is shown by developing few sample applications on top of it and evaluating those applications' performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In recent years, wireless sensor networks (WSNs) have become increasingly popular for observing many real world phenomena. A WSN consists of a large number of tiny sensor nodes capable of interfacing with the real world, communicating wirelessly and computing on-board. Low cost, low power sensor nodes are easily deployable and can be readily networked for a wide range of applications. Most popular areas for WSN applications are health, military, environment and security.

As sensor network applications become pervasive, monolithic and ad hoc approaches used to date may not be feasible and a systematic application design method based on standards and higher level abstractions is required [1, 2]. To simplify sensor network application development, the need for programming abstractions such as a middleware framework is well acknowledged [1, 3]. WSN middleware can be considered as a software that resides above the operating system and below the application abstracting low-level functionalities through a high-level API. A complete middleware solution should provide a holistic view of the network, services for data collection, aggregation and control while efficiently and adaptively managing system resources as per the application's requirements.

The objective of this dissertation is to design a WSN middleware framework that provides adaptive resource management for highly dynamic and large scale WSNs. Our unique bottom-up and distributed approach involves resource management by individual sensor nodes providing real-time adaptation, while ensuring acquisition of

global, system-wide goal. The scope of the work addressed in this dissertation and challenges associated with them are described next.

1.1 Scope and Challenges

WSN nodes are remarkably constrained in terms of their resources such as energy, computational power and radio bandwidth. WSNs normally operate in uncertain and dynamic environments where the state of the system changes considerably over time. For example, in data collection applications, uncertainty exists due to intermittent links or traffic conditions. Moreover, the network itself is dynamic due to such events as node mobility and depleted battery [4]. WSN applications need to cope with such dynamics and uncertainty inherent in sensor networks, while simultaneously trying to achieve each application's requirements for QoS and optimization goal. Consequently, adaptive resource management is a key to any successful middleware solution enabling such applications [1, 5, 6].

Resource management includes initial sensor-selection and task allocation as well as runtime adaptation of allocated task/resources. There are many proposed middleware solutions that have advocated strong need for proactive adaptation of resources [1, 5, 7]. However, there are only few solutions that have actually tried to resolve the issue of enabling adaptive resource management for WSN applications. This problem of resource management/adaptation (see Figure 1.1) can be described as follows:

Given application structure, QoS requirements and current system state, what is the best way of task/resource allocation so that system-wide, application-driven, global parameters can be optimized?

In the above, the application structure is in the form of underlying tasks and their interactions. QoS requirements include such constraints as latency, reliability, coverage etc. while the current state of the system is defined by parameters like mobility,

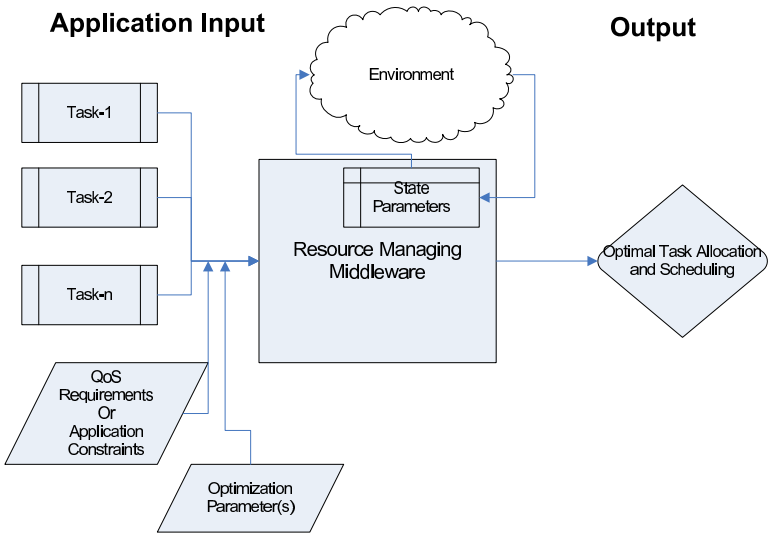


Figure 1.1. Resource management problem in WSNs.

energy availability, and neighboring nodes. The parameters to be optimized include energy, bandwidth, and network lifetime. Without loss of generality, we illustrate the resource management problem using a simplified object/entity tracking application (see Figure 1.2). An object tracking application can be considered to consist of the following tasks: 1) *sample*- sense the environment (e.g. signal strength of a moving object). 2) *transmit (Tx)*- transmit a message to next hop towards the base-stations. 3) *receive (Rx)*- turn radio to receive mode to listen for incoming messages. 4) *aggregate*- aggregate two or more local and remote same target readings into a single reading (e.g. data triangulation for better position estimation or mapping to a known track or simply *last value* aggregation function). 5) *sleep*- put CPU and radio in sleep mode to minimize battery consumption. State representation may consist of the following variables: have one or more neighbors, successful in recent sampling, successful in recent receive, signal strength (or quality of reading). QoS requirements here may include quality of signal, tracking coverage area as well as maximum allowed latency. Our goal in this case is to optimize energy usage among all sensor nodes. Thus the

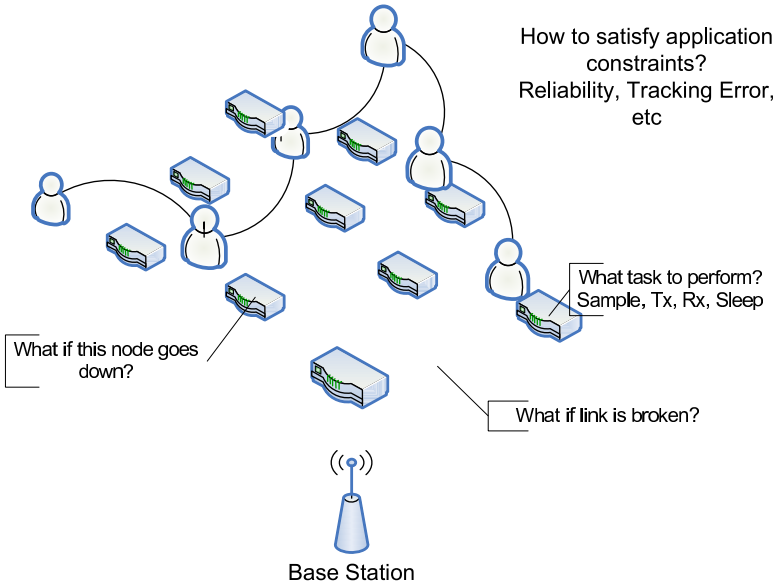


Figure 1.2. Resource management for object tracking application.

goal of our resource management framework is to schedule and allocate tasks on each sensor node in the system, so that energy usage among all sensor nodes is minimized while fulfilling the coverage and latency requirements of the application.

Most sensor network applications studied today consist of a large number of sensor nodes deployed over a geographical area. Sensors use multi-hop communication to send data acquired from the external environment to a sink node or to an Access Point (AP) in the infrastructure. However, several applications do not require fine-grained sensing. Examples of such applications include monitoring of weather conditions in large areas, air quality in urban scenarios, terrain conditions for agriculture, and so on. In such cases, *sparse wireless sensor networks* are likely to be used. In sparse WSNs, the density of nodes is so low that they cannot communicate with each other directly through multi-hop paths. In order to make communication feasible, data collection in sparse WSNs can be accomplished by means of *mobile data collectors* (MDCs). MDCs are special mobile nodes responsible for data gathering and/or dis-

semination. They are assumed to be powerful in terms of data storage and processing capabilities, and are not energy constrained. However, the data collection paradigm in sparse WSNs with MDCs is different, and introduces significant challenges including contact detection and energy conservation.

Communication between an MDC and each sensor node takes place in two phases. First, sensor node discovers the presence of the MDC in its communication range. Then, it transfers collected data to the MDC while satisfying any required reliability constraints. Unlike MDCs, sensor nodes have a limited energy budget, so that the data-collection process has to be energy efficient in order to prolong their network lifetime. In addition, such energy-conserving mechanisms should not compromise the timeliness of communication. This is critical especially when the MDC has only a short contact time with sensors, and also in the case when such contacts cannot be predicted accurately. In fact, a major problem in data collection is that sensor nodes usually do not have *a priori* knowledge of the MDC mobility pattern. Furthermore, even in cases where the arrivals can be predicted, there is a chance that the MDC contacts can be affected by delays or can change their rate. Hence robust and flexible mechanisms have to be defined in order to adapt to operating conditions autonomously.

WSN applications are becoming increasingly pervasive, requiring support for multiple heterogeneous applications executing simultaneously on the sensor network infrastructure. Thus, rather than building a WSN infrastructure for each application, a single WSN infrastructure may be utilized by a wide range of applications. The CitySense [8] project is an example of such a sensing network infrastructure deployed all over a city to allow development of a variety of WSN applications such as traffic statistics/monitoring, accident alerting, public safety and crime watch, driver advisory applications, noise and air pollution monitoring etc. In recent years, utility based

computing has played a significant role in proliferation of cloud computing allowing ease of development of applications using shared computing infrastructure. Similarly, one can envision use of utility theory to enable rapid development of pervasive applications on top of shared sensing infrastructure. The novel concept of developing a middleware that presents WSN infrastructure as a utility which can be shared by many applications, has not yet been explored.

Support for a scalable and robust communication paradigm is another important characteristic of a middleware solution. As all sensor network applications are data-driven, middleware should ideally support a data-centric model for task and data dissemination. Middleware should allow application specific in-network processing and data filtering to enable application assisted routing, aggregation, data fusion and collaborative information processing.

1.2 Major Contributions

Contributions in this dissertation include development of a dynamic, adaptive and autonomous middleware for WSN management. Application of reinforcement learning techniques in the development of a generalized sensor management framework is a unique contribution of this dissertation. The bottom-up design approach used in this dissertation allows usage of the developed middleware in a wide-scale distributed sensor systems with inherent support for uncertainty and dynamicity prevailing in WSNs.

1.2.1 Adaptive Resource Management Using Distributed Independent Reinforcement Learning (DIRL)

The main idea of DIRL is to allow each individual sensor node to self-schedule its tasks and allocate local resources by learning their usefulness (utility) in any given state while honouring application defined constraints and maximizing total amount of reward over time. DIRL is based on Q-learning [9], a form of model-free reinforcement learning. Q-learning is simple to implement, demands minimal computational resources and doesn't require a model (e.g. state transitions) of the environment in order to operate. Hence it is ideal for implementation on resource-constrained sensor nodes. DIRL uses a classic exploration and exploitation strategy as used in most RL based approaches to learn utilities of various tasks. DIRL addresses structural credit assignment (propagation of reward spatially across states in order to define the notion of similar states) by using weighted hamming distance between two states. Here, the application state is represented in the form of system and application variables each carrying an associated weight. DIRL employs independent learning where each agent applies the learning algorithm in a classic sense (like in single agent systems) ignoring the presence of other agents. The main advantage of using independent learning in DIRL is that no communication is required for co-ordination among sensor nodes and each node selfishly tries to maximize its own rewards. Our analysis shows that DIRL is not only feasible but also provides higher efficiency for task/resource management compared to other related approaches.

1.2.2 Ensuring Global Optimization With Multi-Tier Reinforcement Learning

DIRL works well when each node in a WSN application is acting on its own and doesn't need to co-operate or compete with other nodes. In other words, if all nodes are acting independently and their actions do not affect others, then any increase in

a node's utility cannot decrease anyone else's utility and hence will always increase world (system-wide) utility which is merely the sum of all node's utilities over all time. But most of the real-world WSN applications need some sort of co-operation among sensor nodes and hence nodes cannot work independently. In this case, increase in utility of an individual node may result in the reduction of other nodes' utility and hence may not increase World utility. It is also possible that such a system can lead to phenomena like Tragedy of the Commons (TOC) or Braes' Paradox, wherein an individual's selfishness leads to significantly lower world utility [10]. Such phenomena can be avoided by carefully designing the agents' utility functions as well as constraints under which an agent performs task selection. In other words, it is required to make sure that an individual's utility is 'aligned' with the World utility, i.e. any increase in an agent's private utility due to its action will also result in increase of World utility. COllective INtelligence (COIN) theory [10, 11] provides principles on designing such private (individual node) as well as global utility functions such that they are aligned. These principles are utilized to design a multi-tier reinforcement learning based framework for WSN resource management.

The design goal is to create a system using a bottom-up approach where each sensor node is responsible for task selection, rather than a top-down approach (where some central entity dictates nodes what task to execute) used by many other middleware solutions [1, 5]. Main advantages of a bottom-up approach are pro-active and real-time adaptation, no centralized processing requirement for task allocation and minimal communication overhead. But the principal challenge of a bottom-up approach is how to make sure that the system is actually meeting the global application goals and is not just acting randomly or creating chaos. This issue is resolved by using two-layer learning: micro-learning is used by individual nodes to self-schedule their

tasks and macro-learning is used by each set of closely connected nodes to steer the system towards the application goal by setting/updating rewards for micro-learners.

1.2.3 Resource-aware Data Accumulation in Sparse Wireless Sensor Networks

The next contribution addresses the issue of energy-aware resource allocation in sparse WSNs with Mobile Data Collectors (MDCs). Discovery and data transfer protocols are defined for energy-efficient data collection in sparse WSNs with MDCs and an adaptive strategy that exploits the DIRL scheme is proposed. We design a generic resource-aware data accumulation (RADA) framework that can be applied to a wide range of applications while minimizing energy consumption. The principal idea is to learn the underlying pattern of MDCs' arrival and tune the sensor node's duty cycle accordingly. Through extensive simulations it is shown that RADA is highly energy efficient for adaptively managing a sensor node's duty cycle while maintaining high discovery rate and data transfer efficiency.

1.2.4 Design of DReL Middleware Using Multi-Tier Reinforcement Learning and Directed Diffusion

After devising strategies and frameworks for resource management, next a complete middleware solution called DReL (Distributed Reinforcement Learning) is designed. DReL is specifically designed to enable utility computing based application development and incorporates other important aspects of a WSN middleware such as task and data dissemination. DReL addresses the issue of defining structure for tasks/data and acts as a generic middleware that provides an easy-to-use interface for development of WSN applications with different QoS and optimization requirements. We provide detailed mechanisms for publication of tasks, collection of data and com-

putation and distribution of rewards to achieve a global optimization goal using our multi-tier learning scheme and directed diffusion [12, 13].

1.3 Organization of Dissertation

Chapter 2 reviews existing literature and provides some necessary background on topics that are related to this dissertation. Chapter 3 describes our preliminary DIRL scheme for resource management in WSN. An extended multi-tier reinforcement learning based framework using COIN and DIRL is presented in Chapter 4. Chapter 5 explores application of DIRL to sparse WSN for energy-aware MDC discovery. Chapter 6 provides a detail design of a complete adaptive and autonomous middleware DReL.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, some of the associated topics pertinent to the research performed in this dissertation are reviewed. Accordingly, background and related work in the areas of wireless sensor networks (WSN) middleware, resource management in WSN and sparse WSN are presented. The chapter also includes a brief introduction to reinforcement learning (RL), multi-agent systems and collective intelligence (COIN) theory which play a significant role in our work. A brief background on directed diffusion which is utilized as communication paradigm for task, data and reward distribution in our middleware is also provided.

2.1 Wireless Sensor Networks Middleware

Complexity of sensor network applications is continuously increasing with the proliferation of distributed computing and wireless connectivity in sensor networks. Heterogeneity of sensor networks in terms of hardware characteristics and application requirements further adds to application complexity. As a result, developing sensor network applications has become very difficult. A middleware infrastructure is required which can ease up such application development. Next challenges for development of WSN middleware are discussed. Requirement and design principles of a middleware emerging out from these challenges are also described later.

2.1.1 WSN Middleware Challenges

Development and implementation of a sensor network middleware is not a trivial task and the main reasons for this are some inherent characteristics of wireless sensor networks. We have tried to capture some of these characteristics here.

- Sensor nodes are limited in the amount of energy they can store or achieve from the environment. This size and energy limit implies extremely resource constrained devices in terms of CPU power, memory and wireless bandwidth and range which in turn limits processing and interactions normally required by distributed systems.
- Sensor nodes are subject to frequent failures, either due to depletion of battery or environmental influences. Also nodes can be highly mobile. This brings in a high degree of dynamics and uncertainty in WSNs and can result in frequent topology changes and network partitions.
- Wireless links used by low power sensor nodes are also error prone. Hence communication failure is also a problem to be addressed.
- WSNs are often heterogeneous, e.g. a network often consists of nodes/devices with various capabilities in terms of sensors, computing power and memory. This further complicates WSN management.
- A WSN application consisting of hundreds or even thousands of sensor nodes is normal and hence scalability is a major issue here.
- Large numbers of sensor nodes and the possibility of their deployment in hostile or in-accessible areas mandates that such a system provides *exception-free unattended operation* [14]. This suggests the necessity of autonomous adaptation in terms of reconfiguration and task dynamics.
- Concern for security and privacy is more aggravated for WSN applications because of sensitiveness of data collected and lack of resources.

- With the pervasiveness of sensor network applications, it will become necessary to support multiple heterogeneous applications with different requirements on top of a single network.

2.1.2 WSN Middleware Requirements

Challenges described in the previous section can be interpreted to frame a list of requirements for WSN middleware. This list is as follows:

- Middleware systems should support high level abstraction, hiding complexity of dealing with individual nodes and provide a holistic view of the network. Middleware should also support multiple programming paradigms like publish-subscribe, pull-push models, event based etc.
- It is necessary for WSN middleware to be very efficient in terms of energy, bandwidth and computational resources consumption.
- Middleware should be proactive and support adaptive fidelity algorithms to improve performance.
- Middleware should be able to provide support for deployment, configuration and reconfiguration. It should also provide mechanisms for application policy creation and distribution.
- WSN middleware needs to provide scalability in terms of the number of sensor nodes, applications running on top of it and associated users.
- Middleware needs to support execution of multiple applications simultaneously on a single sensor network. It should be able to support QoS requirements of individual applications in terms of availability, timeliness, fault-tolerance etc.
- WSN middleware should be able to provide optimal network configurations in order to support a high degree of dynamics of WSN.

- WSN middleware should provide security in the context of data processing, data communications, device tampering etc.
- Finally, middleware itself should be lightweight in terms of computation and communciation requirements in order to be successful in resource constrained WSNs.

2.1.3 WSN Middleware Design Principles

Based on some previous work [14, 1] and driven by the above requirements, we hereby list design principles for WSN middleware.

- A WSN middleware framework should support a data-centric or data-driven model for data processing and querying.
- A WSN middleware should allow incorporation of application knowledge and hence allow overall software to be tailored to the executing application. This conflicts with the desire to support and optimize for a wide class of applications. This issue can be resolved by allowing an application to put its unique features as code or specification which can be interpreted by middleware.
- Cluster-based localized algorithms should be used for efficiently coordinating local interactions between sensor nodes in order to achieve global goals. This can help in creating more scalable applications and can improve robustness and efficiency in resource utilization.
- Middleware should support various *data reduction techniques* required for collection and processing of sensory data. Complex sensing tasks often require that data collected at many nodes be fused to obtain a high-level sensing result. For reducing communication and energy overhead, it is necessary to process sensory data at the source to extract relevant features. Middleware will require

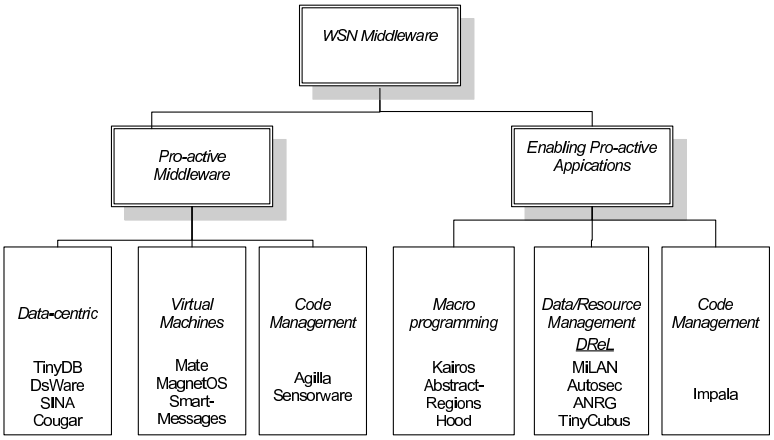


Figure 2.1. WSN Middleware Classification.

means to specify application knowledge and ways to inject this knowledge into the nodes of the network.

- Sensor networks are mostly connected to external background infrastructures mainly for tasking the sensor network, as well as for evaluation and storage of sensing results and also may provide computing and storage resources not available to sensor networks. Hence, middleware should allow the integration with such background infrastructures, providing a homogeneous view of the entire system.

2.2 WSN Middleware Classification

A high level classification of the WSN middleware frameworks is given in Figure 2.1. As shown in the figure, WSN middleware can be broadly classified into pro-active middleware and middleware enabling pro-active applications [5]. Pro-active middleware refers to middleware that pro-actively changes WSN functionality based on current environment and state of the system. Middleware enabling pro-active applications allows applications to guide the system on how to adapt to changes

in the environment and state of the system. Pro-active middleware can be further classified into the following:

2.2.1 Data-centric

Middleware in this category provides the abstraction of the whole sensor network as a virtual distributed database. The idea is to enable utilization of a higher level declarative programming language/interface upon the distributed nodes without having to deal with the network issues. A user dispatches queries using some easy-to-use interface to extract data of interest. The main focus of this type of middleware is on efficient evaluation of query plans using in-network query processing. Such middleware normally only uses predefined static task schedules and normally doesn't handle adaptive task/resource management. Also the approach provides only approximate values and lacks support for real-time applications that need spatio-temporal relationships between events [2].This type of middleware are not expressive enough to implement arbitrary distributed algorithms. Cougar [15] is the first work considering WSN as a database. TinyDB [16] hides the complexity of TinyOS by building a query-processing system on top of it. TinyDB allows users to extract data of interest from the sensor nodes using a SQL-like interface. SINA [17] and DsWare[18] are two other approaches falling into this category of middleware using database abstraction.

2.2.2 Virtual machine

These middleware abstract a WSN as a collection of code interpreters that take care of running programs and scripts. Examples include Mate [19], MagnetOS [20] etc. Mate is a tiny, efficient virtual machine designed and implemented on top of TinyOS [21]. Mate devices code into small capsules of instructions. It defines and implements several frequently used functionalities and provides a single instruc-

tion API for implementing them. Thus Mate allows application developers to write efficient programs with a minimal number of instructions. It provides a (viral) broadcast solution for propagation of programs broken into small capsules. MagnetOS is a power-aware operating system that provides a single system image of ad hoc networks so that the whole network appears as a single, unified Java VM. To achieve performance efficiency, MagnetOS supports object migration to move objects closer to data sources and hence does support code management.

2.2.3 Code Management

This category includes middleware solutions concentrating on providing services for code deployment including allocation and migration of code to sensor nodes. Code allocation includes initial sensor selection for code execution. Code migration involves transferring of code from one node to another effectively allowing to re-program the network. The middleware's efficiency relies on strategies for migrating code to sensor nodes in the vicinity of the phenomenon of interest. Generally, this type of middleware is built on top of a virtual machine that allows remote code execution and migration. Middleware normally handle a trade-off between complexity of the interpreter running on the nodes and complexity of mobile node [22]. Task migration mainly uses mobile code (or mobile agents) moving the code to data sources to allow local data processing. Agilla [23] is an example of mobile-agent based middleware concentrating mainly on code management aspects. Sensorware [24] uses TCL scripts as mobile code that gets transfered through the system to facilitate migration. Middleware of this type suffers from the following issues: 1) Complexity of specification and application development, 2) High resource consumption in terms of computation and communication costs for code migration.

Even though most of the above middleware approaches have some support for chang-

ing WSN functionality based on system state, they do not allow the application to change system behavior directly. Next we classify middleware that allow applications to adapt to current system state and environment.

2.2.4 Macro-programming

Macro-programming involves programming the sensor network as a whole rather then implementing low-level software for individual nodes. The nodal behavior is automatically generated from a high-level global program. This relieves developers from dealing with low-level concerns like distributed code generation, remote data access and management and inter-node program flow coordination at each network node [25, 26]. Middleware in this category normally utilize a concept of neighborhood to allow development of localized algorithms. As nodal behaviors are automatically generated, such middleware providing system level abstraction is easier to use. But they do suffer from high communication and computational overhead with less flexibility in implementation of energy efficient algorithms as compared to programming individual nodes. Kairos[25] focuses on providing the necessary notions and concepts to design, develop, and implement a macroprogramming model on WSN. Kairos enables programmers to choose either tight or loose synchronization based on their needs. Abstract Regions[26] provide a suite of general purpose communication primitives for WSN that handles addressing, in-network data aggregation and data sharing among network's local regions. It aims at achieving energy efficiency by reducing radio communication with use of higher localized computation. Hood[27] also uses a local neighborhood concept similar to Abstract Regions. Hood shows that local behaviors can be implemented adequately by using the neighborhood abstraction.

2.2.5 Code Management/Application Scheduling

Impala [6] is an example of a middleware enabling pro-active applications with focus on code management and application scheduling. The principal goal of Impala is to allow reliable and adaptive code management and code upgrades for long-running WSN applications with infrequent code updates. Impala is designed for the Zebranet project providing wildlife tracking technology. Impala is built on top of an event-based programming model. As for adaptation, it provides a mechanism for dynamically querying operating parameters and checking them against a set of application specified rules to determine if an application switch is required. Application switching is based on a finite state machine model, where transitions are defined in the form of heuristics, and can cope with device failures. However, Impala has been specifically targeted to scenarios where all nodes are mobile and act as peers. Also Impala is being destined to run on linux based hand-held pocket PCs and hence has limited applications.

2.2.6 Data/Resource Management

Middleware solutions in this category advocate need for pro-active adaptation of resources. They handle trade-off between applications QoS requirements and WSN resources. They allow applications to affect WSN resources by varying settings over time to meet applications' goals.

MiLAN [5] (Middleware Linking Applications and Networks) focuses on a cross-layer architecture that extends into the network protocol stack and allows applications to change the network. MiLAN obtains QoS requirements and sensor configurations from the application in the form of specialized graphs, monitors network conditions and based on requirements varies sensor and network configurations to optimize resource utilization and maximize network lifetime. From the specialized graphs, MiLAN de-

termines an application feasible set (F_A), each element of which is a set of sensors that can satisfy QoS requirements of each application specified variable. MiLAN's network plugin next chooses a network feasible set (F_N) that can be supported by the network based on network properties and current state. F_A and F_N is next combined to derive an overall feasible set F, i.e. $F = F_A \cap F_N$.

A cluster-based architecture is described in [1] using virtual-machine abstraction to separate application semantics from underlying physical infrastructure. In this framework, each cluster consisting of a set of spatially adjacent sensor nodes around target phenomena forms a basic functional unit. The framework consisted of two layers: 1) resource management layer-residing only at the cluster head and 2) cluster layer distributed among sensor nodes. The cluster layer is responsible for forming a cluster from a group of sensor nodes and then distributing commands issued from cluster head into the cluster. The resource management layer is responsible for allocation and adaptation of resources so that an application's QoS requirements can be satisfied. Thus all nodes in a cluster cooperate together to create an adaptive resource management layer. TinyCubus [7] is also an adaptive cross-layer framework for TinyOS based sensor networks. TinyCubus consists of a configuration engine that distributes components to sensor nodes based on their role and installs components dynamically, and a data management framework that aims at providing adaptation of system and data management components at runtime based on application requirements and system parameters.

Even though the issue of enabling adaptive resource management for WSN applications is addressed by some mentioned middleware solutions, they require complete knowledge of the system. Adaptation at runtime, for most of them, is very expensive in terms of resource requirements. Most of the approaches required a centralized control to make adaptation decisions and effect global system behavior. Adaptation by

code management on individual sensor nodes is very resource intensive and fragile. Hence, such a centralized system for resource management will not scale with network size. Task scheduling over individual sensor nodes is also not addressed by most of the above work. We have focused on the adaptive data/resource management problem for design of our WSN middleware solution and hence our work (DReL- Distributed Reinforcement Learning based middleware) falls in the category of *Data/Resource Management* as shown in Figure 2.1. We have used a reinforcement learning based bottom-up approach which is inherently distributed, autonomous and adaptive to dynamic changes prevailing in WSN. The rest of the chapters in this dissertation provide detailed description of our middleware framework.

2.3 Resource Management in Wireless Sensor Networks

In this section we provide related work focusing on the resource management problem in WSN. This work can be divided into the following categories:

2.3.1 Rule/Predicate logic based

Here a set of rules are pre-programmed on individual nodes. A rule is fired if all conditions/parameters included in the rule predicate evaluates true and this may result in task adaptation. Work in this category includes [28, 6, 29]. This is a simple technique but requires that all state conditions be known in advance, when adaptation might be necessary. Also it can get very complex with a large number of nodes and high dynamics, when the system state is changing at a high rate. Also the effect of rule based adaptation at local sensor nodes on global WSN system/application is not considered. Generic Role Assignment (GRA) scheme [28] considers sensor nodes in the WSN system taking on certain roles in the network based on their properties such that requirements of a given role are satisfied. GRA uses rules to

encode node behavior and implements an algorithm to specify a set of roles that a node can assign to itself depending on its local and neighboring nodes' state. GRA provides abstraction to allow specification of roles and rules for their assignment using a configuration language. Impala [6] uses rules to implement its application adaptor component. Application adaptor in Impala follows a finite state machine where each state represents an application and transition between states is governed by rules represented as parametric expressions. A transition is made from one application to another if that rule is satisfied. Adaptor also maintains a set of application system parameters to check for transition rules. FACTS [29] uses abstraction of facts, rules and functions for their middleware architecture where all information ranging from sensor readings to state variables is represented as facts. These facts are stored in a fact repository and processed by rules. Facts, rules and functions are local to each node of the sensor network and each node runs its own rule engine. Facts are also been used as medium of information transmission across nodes.

2.3.2 Constraint-satisfaction based

Here, the problem is defined in terms of constraint-satisfaction and sometimes it is reduced to linear programming with the objective function consisting of optimization parameters under given constraints (application requirements). Research work in [30, 31, 32] are examples of using such a constraint-satisfaction based approach. Due to the complexity of the system, it is not always possible to reduce our resource adaptation problem into a linear programming problem without making unreasonable assumptions. Also it doesn not always allow the distribution, making it impractical in large scale WSNs. A constraint-guided software reconfiguration approach is proposed in [30] where adaptation is performed by updating software components on TinyOS motes based on constraints. Here system parameters are expressed as formal

constraints on QoS parameters that are measured at runtime. Each mote monitors its local QoS parameters and transmits them to the base station. Base station components utilize these parameters to decide whether an adaptation is necessary or not. If adaptation is required, QoS parameters along with associated constraints are fed to their model generator to finally output a new software configuration for affected motes, which is then pushed down to those motes. Krishnamachari et el. [31] modeled various configuration tasks of multi-hop wireless networks as distributed constraint specification problems. These configuration tasks include partitioning the network into coordinating cliques, hamiltonian cycle formation and conflict-free channel scheduling. Authors mapped out the connection between critical power thresholds in wireless networks and the work on constraint satisfaction and show that the average problem complexity can be reduced by tuning the transmission power of individual nodes. A distributed constraint optimization algorithm called *Adopt* [32] is proposed for general multi-agent systems. Modi et el. propose a systematic formalization of the distributed resource allocation problem and a general solution strategy that maps a formal model of resource allocation into a problem solving paradigm called distributed constraint-based reasoning. Adopt uses localized asynchronous communication and makes local decisions based on conservative cost estimates rather than global certainty which results in a polynomial-space algorithm.

2.3.3 Agent negotiation/Auction based

Research in this category ([33, 34, 35]) mainly includes multi-agent systems with agents being able to negotiate with each other in order to determine the best allocation. The system consists of one or more mediators responsible for negotiations. Although this approach can lead to efficient resource management, communication and computational resources required for these negotiations may be sometimes prohibitive

for their implementation on a resource constrained WSN system. A center-based algorithm called *Mediation* is proposed in [34] to address the task assignment problem in distributed sensor networks. Mediation implements an iterative hill-climbing search in a subset of the solution space by making and sending successive proposals to the agents of a group. Group members provide responses to the mediator based on the context of the proposal and the mediator uses these responses to find a satisfactory solution to the task assignment problem. The mediation algorithm is further extended to address a) changes during the decision-making process, b) negotiation over tasks whose utilities are not necessarily additive and c) monitoring and adapting a solution in a non-stationary environment. A distributed resource allocation based on dynamic coalition formation and coalition strategy learning is presented in [35]. In this scheme, agents attempt to operate autonomously with incomplete information about their potential collaborators. Synchronization of actions across multiple agents is achieved by forming coalitions via multiple 1-to-1 negotiations. However, because of uncertain properties of WSN, coaltions formed will be suboptimal and satisficing. To allow adaptation to environment dynamics, each agent is capable of multiple levels of learning. This includes case-based learning to learn how-to negotiate better and reinforcement learning to learn how-to form a better coalition. In our work, we also utilize multiple-level of learning but it is mainly based on reinforcements as a result of local actions and global outcome. Forming an explicit coalition warrants excessive communication and computation overhead and is not feasible on resource constrained sensor nodes and hence we do not require any explicit coalition formation.

2.3.4 Utility/Market based

Here the purpose is to define a utility function mapping optimization parameters over a number of participating nodes to a real value and maximize these functions

under given constraints. The problem of solving this utility function can further take the form of a linear programming problem which can then be addressed by distributed or centralized approach.

Byers et el. [36] were the first to introduce utility functions to define global objectives of WSN applications along with a cost model for energy consumption. Rather than adopting a best-effort model, authors used a model where a node makes heuristic assessments based on mostly locally available information. The model is further driven by objective functions that maximize the utility of sensor networks over their lifetime. Although the simplistic model presented by Byers et el. was significant, routing algorithms and heuristics presented by them were very primitive and cannot be applied to real-world WSN applications. Our approach of allowing individual nodes to maximize their utility functions and representation of cost model is inspired by their work.

A utility based sensor network design using techniques from mechanism design and game theory is presented in [37]. Sadagopan et el. focused on continuous data gathering applications that construct a spanning-tree rooted at the data sink, with two global objectives, namely load balance and energy balance. They have attempted to address the same problem that we are interested in, i.e. designing local utility functions for sensor nodes so that each node *selfishly* optimizing its local utility function leads to optimization of a desired global objective function. But they have concentrated on theoretical aspects of developing a game-theory based algorithm for constructing a load-balanced spanning tree in the network and their work does not address issues in development of a generic framework for sensor network management which is our focus in this dissertation. Mainland et al. [38] proposed a market based approach called *Self-Organizing Resource Allocation (SORA)* for resource allocation in sensor networks. Each sensor node in SORA acts as an agent that tries to maximize it's payment by undertaking a set of actions. Each action can result in some energy

consumption and can also produce some goods with an associated price. Agents receive feedback on their actions in the form of payment which drives their behavior. Their work is closest to our approach as they also use reinforcement learning by individual nodes for the task scheduling problem. But they employ a simple heuristic model for learning utilities and do not consider any state based learning, resulting in a reactive system with only one state. We claim that exploiting learned utility values along with the state knowledge can result in a more efficient and responsive system. Also they don't provide any mechanisms to ensure desired global behavior of the WSN application. Our results in Chapter 3 and 4 show comparison of DIRL with SORA and validate our claim.

Even though each of the approaches mentioned above can provide efficient resource management, they suffer from some pitfalls as described. Most of them require a centralized mechanism which is not scalable and robust for many real world WSN applications. None of the techniques described in previous sub-sections try to address uncertainty which is inherent in dynamic networks. Furthermore, most of them require a careful implementation of algorithms on a case-by-case basis which may be quite difficult in sensor networks. Therefore, a generic framework that can enable a large set of applications with autonomous adaptation and minimum communication overhead is required. We address the issue of designing such a resource management framework in this dissertation.

2.4 Sparse WSNs

Solutions for adaptive resource management and energy-efficient data collection for sparse WSNs have already been considered in the literature. However, in most cases these issues have been considered separately.

In the context of opportunistic networks, the well known message ferrying approach was proposed in [39]. Power management has been addressed by [40], where a general framework for energy conservation is introduced. The proposed solution, which can also exploit knowledge about the mobility pattern of the MDC, is evaluated in terms of energy efficiency and delivery performance. However, as the proposed solution is devised for opportunistic networks, it cannot be used without being redesigned in the sparse WSN scenario considered in this work.

Many solutions have also been conceived specifically for WSNs. While many papers focus on the mobility of the MDC [41, 42], some works actually address the problem of energy efficient data collection from the sensor nodes' standpoint. For example, [43] considers a periodic wakeup scheme for discovery and a stop-and-wait protocol for data transfer. A stop-and-wait protocol for data transfer is also used in [44], where the MDC is assumed to be controllable. A different solution is investigated in [45], under the assumption that the MDC has a completely predictable mobility.

A performance evaluation of data collection jointly considering both discovery and data transfer is presented in [46, 47, 48]. The authors consider a periodic wakeup scheme for discovery and a selective retransmission scheme for reliable data transfer. In addition, they provide a very simple characterization of the energy efficiency on the basis of how predictable the mobility pattern of the MDC is.

Approaches such as [41, 42, 44] are not applicable when the MDCs are part of the environment – e.g., when they are buses or cabs. On the other hand, research works focusing on the performance analysis of the data-collection process [45, 43, 49, 46, 47, 48] rely on the assumption that the operating parameters are chosen prior to deployment, and do not change with time. Clearly, these approaches lack flexibility, as they require an *a priori* characterization of some network parameters, e.g. the mobility

pattern of the MDC, the duration of the contacts and the message generation rate. In addition, the chosen parameters cannot adapt to changing operating conditions.

An adaptive data collection protocol has been considered in [50]. However, the corresponding analysis is limited only to the scenario where the sensor can transfer all buffered data in a single contact. In addition, the impact of MDC mobility is not considered at all. In this work, instead, we provide an adaptive strategy suitable to data collection in WSNs, even when there is only a little knowledge on the mobility pattern of the MDC.

Knowledge-based approaches to data collection for WSNs with mobile elements have been proposed in [51, 52]. In [51] the WSN is assumed to be rather dense, so that nodes can organize into clusters. Within each cluster, a specific node operates as a proxy, i.e., it collects data from the other nodes in the cluster and relays them to the MDC. After detecting the presence of the MDC in their proximity, proxies initiate a reinforcement-based routing process so that messages are relayed to the destination while it traverses the network. In contrast, [52] exploits reinforcement learning for discovery purposes, in the different context of sparse WSNs where nodes operate as peers. Specifically, nodes scan for neighbors and use the number of encounters as a reward. The reward is mapped to a time-based domain using context of time of the day. Then, sensor nodes perform discovery according to the likelyhood of the other peers to be in contact, according to their energy budget. Although both [51] and [52] use reinforcement learning for data collection, they do not specifically address the problem of sparse WSNs with MDCs. In fact, the approach in [51] is more focused on routing, and assumes that the network is dense enough to form clusters of nodes. On the other hand, the solution in [52] is suitable to sparse WSNs. However, it considers sensor nodes as mobile peers. Instead, we are considering here WSNs where nodes are static, except for a number of MDCs.

The TinyLime [53] middleware has been proposed for the specific scenario of sparse WSNs. TinyLime is based on a tuple space model, an implementation of the distributed shared memory paradigm for distributed computing. The original tuple space model is extended to the scenario where multiple MDCs collect data from sensor nodes which are not densely deployed in the network. To this end, no assumption is made on network connectivity since nodes can even be isolated from each other. TinyLime provides mechanisms to perform data aggregation and tune the activity of nodes in order to save energy. However, the focus of TinyLime is on their proposed programming abstraction rather than on adaptation and resource management.

2.5 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine-learning and is concerned with determining an optimum policy that maps states of the world to the actions that an agent should take in those states so as to maximize a numerical reward signal [54]. The agent receives a numerical reward as a consequence of it's action which provides a reinforcement signal. The agent tries out different actions in order to learn what actions yield the most reward. An action is selected either based on past experiences (exploitation) or using exploration. RL is very useful for interactive/online learning in dynamic uncertain environments.

In this work, we use Q-learning [9] which is a form of model-free reinforcement learning. Q-learning uses a single data structure, a utility look-up table $Q(s,t)$ across states s and tasks t. The utility of performing task t in a state s is defined as the expected value of the sum of the immediate reward r and discounted utility of resulting state s' after executing task t, i.e.

$$Q(s,t) = E[r + \gamma e(s')|s,t]$$

where $e(s')$ = Maximum $Q(s', t)$ over all tasks t. Note that the expected value above is conditional upon being in state s and performing task t. As 'Q-learning' is done online, the above equation cannot be applied directly as stored utility values may not have converged yet to final values. Hence, in practice, Q-learning is used with incremental step updates as given by the following:

$$Q(s,t) = (1-\alpha)Q(s,t) + \alpha(r + \gamma e(s'))$$

Here α is a learning-rate parameter between 0 and 1. It controls the rate at which an agent tries to learn by giving more (α close to 1) or less (α close to 0) weight to the previously learned utility value. This means setting α equal to 1 will make the agent ignore all previously learned utilities resulting in single-shot learning. γ is a discount-factor and also varies from 0 to 1. The higher the value, the greater the agent relies on future reward instead of merely immediate reward.

An important aspect an RL system is the trade-off between exploration and exploitation. Exploration deals with trying out some random actions which may not have higher utility in search of better rewarding actions, while exploitation tries to use the learned utility to maximize the agent's reward. Most of the RL systems use exploration with a certain probability ε, which can be a constant value (mostly around 0.1 to 0.5) or can be derived using some other heuristics like starting with a high value and gradually decreasing, for example using the Boltzmann equation [54].

A COIN [11, 10, 55] is a large multi-agent system with a well-defined 'world utility' function which rates the behavior of the entire system and there is little or no centralized control. Each agent in the MAS is *selfish* and runs a RL algorithm. The system's global behavior hence is the collective effect of individual agents each modifying their behavior using an RL algorithm. COIN theory addresses the following

design problem:

Given the individual agents are maximizing their own (local) utility functions (e.g. Q-learning), how to design these local utility functions to ensure optimum world utility? Or how to ensure that agents do not frustrate other agents resulting in lower world-utility?

The RL algorithms at each agent that aim to optimize their local utilities are called microlearners. The learning algorithms that update the agents' utility functions are called macrolearners. COIN uses game-theory concepts to devise a methodology for designing/updating local utility functions at each agent so that system will approach near-optimal values of the world utility. COIN theory proves that a collective system which is factored and has higher learnability, eventually reaches a Nash Equilibrium point when all nodes are fully rational in optimizing their utility functions. This Nash Equilibrium point is also the Pareto optimal point of the system. We will address concepts used by COIN as well as its application to WSN in Chapter 4.

2.6 Directed Diffusion

Directed diffusion [12, 13] has been popular among researchers in wireless sensor networks (WSNs). The data-centric nature as well as the ability to perform effective task and data dissemination in WSNs using only localized interactions among neighboring nodes, as facilitated by directed diffusion, has been widely accepted [4, 56].

Directed diffusion is mainly a network-layer communication paradigm for data dissemination from source to sink using interest (task description) gradients setup during interest propagation. It is a data-centric approach where a sink broadcasts its interest to all other sensors as shown in Figure 2.2(a). Task description consists of a list of attribute-value pairs specifying applications variables of interest. Each sensor node stores the interest entry in its cache. Each entry contains gradient fields,

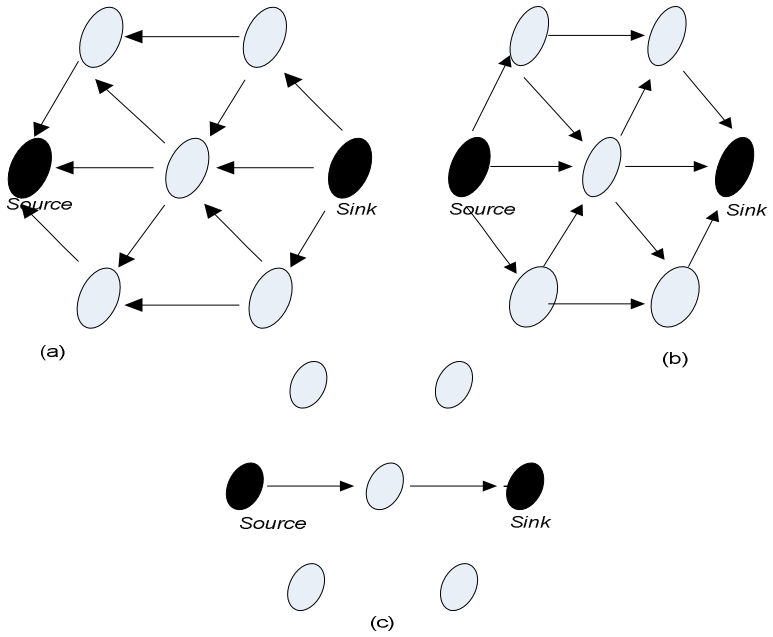


Figure 2.2. Directed Diffusion (a) publishing of interest, (b) exploratory data broadcast to sink and (c) Data dissemination across reinforced path.

one per each neighbor from which the interest was received. As interest propagates throughout the sensor network, a gradient is setup from a source to a sink sensor node as shown in Figure 2.2(b). A node capable of providing the data acts as a source and begins to send exploratory data towards sinks using multiple gradient paths. The lowest delay gradient path is then reinforced by the sink. Data dissemination from source to destination then uses this reinforced path as shown in Figure 2.2(c).

This original form of directed diffusion is found to suffer from large resource overheads [57] because of interest as well as exploratory data flooding throughout the network. Two other types of routing protocols were proposed in [57]: (a) Push diffusion where the sinks become passive, with interest information kept local while sources become active and exploratory data is sent throughout the network (without interest gradients). (b) One-phase pull diffusion where interests are diffused similar

to the original (also called two-phase pull diffusion) protocol, but instead of sending exploratory data towards all gradients, each sensor node directly sends data over the preferred gradient. Both the above protocols are designed to improve performance of different classes of applications, i.e. push diffusion in the presence of many sources and sinks (nodes cross-subscribed to each other) with occasional data publication while one-phase pull in case of many sources publishing large amount of data with fewer sinks. One-phase pull however assumes symmetric communication links between source and sink, which is not true for most WSNs. Chapter 6 describes our DReL framework to generalize benefits offered by all these protocols (without their disadvantages) in the form of single unified interface to application developers and can be applied to a variety of application classes.

2.7 Summary

In this chapter, we provided background and literature review on various topics associated with the work in this dissertation. We have discussed challenges, requirements and design principles for middleware frameworks for wireless sensor networks. We have provided detailed classification of middleware frameworks available to date and how our work compares with them. We have next concentrated on resource management aspects for WSN and provided a classification of work in this area. We have also given a brief background on reinforcement learning and Q-learning as well as multi-agent systems. We briefly mentioned COIN theory and its applicability to learning in co-operative multi-agent systems. We have also described directed-diffusion which we use as a communication paradigm for task, data and reward distribution. These techniques provide the basis for our framework and are further discussed in the remaining chapters.

CHAPTER 3

ADAPTIVE RESOURCE MANAGEMENT USING DISTRIBUTED INDEPENDENT REINFORCEMENT LEARNING

In this chapter, we present a simple reinforcement learning based framework to address the issue of dynamic resource adaptation in wireless sensor networks (WSNs). WSNs can be modeled as a multi-agent system (MAS) with each sensor represented as a goal-oriented agent. Standard reinforcement learning (RL) techniques (e.g. Q-learning) can be applied directly to MAS. In MAS reinforcement learning can take any of the following two forms [58]: a) Independent Learners- where the learning algorithm is applied in a classic sense ignoring the presence of other agents; and b) Joint Action Learners (JAL) - where agents try to learn in conjunction with other agents by coordination through game-theoretic approaches and considering their joint actions. Claus and Boutilier [58] have shown that 'even though JAL have much more information at their disposal, they do not perform much differently from independent learners in straightforward application of Q-learning to MAS'. Use of JAL involves much more communication and processing overhead and may be an over-kill for WSNs. Hence our focus in this chapter is limited to independent learners where each sensor node acts as an individual interacting with the rest of the environment without consideration of other nodes. All other nodes are considered part of the environment.

The main idea of DIRL is to allow each individual sensor node to self-schedule its tasks and allocate its resources by learning their usefulness (utility) in any given state while honoring application defined constraints and maximizing total amount of reward over time. The advantage of using independent learning is that no commu-

nication is required for co-ordination between sensor nodes and each node selfishly tries to maximize its own rewards. Global optimization parameter that application is interested in can be specified in terms of individual task rewards at a sensor node. Thus, if an application needs to optimize energy usage, each task's reward function can be a combination of task outputs as well as energy consumed for performing that task.

3.1 Assumptions

We make the following assumptions in designing our model:

1. Each node is independent and as such actions of one node doesn't affect other nodes in the system.
2. Each node is capable of performing only one task at a time.
3. A node is only allocated with a set of tasks it's capable of executing, i.e. initial task allocation depending on sensor node type (in case of heterogeneous network) is done apriori.

Assumption 1 above is quite restrictive as many WSN applications may have sensor nodes tightly coupled and may directly affect each other. We use this constraint to lay out the initial DIRL framework requiring no communication overhead and then remove this constraint in next chapter where we address global optimization and effect of node's action on system wide utility. Assumption 2 means that sensor node is single threaded as is true with many real-world sensor modules available today. This is not a restriction as our approach can be utilized in environments supporting multi-threaded nodes. We address and remove the constraint noted by Assumption 3 in Chapter 6 where we design a complete middleware framework with support for task distribution and task allocation.

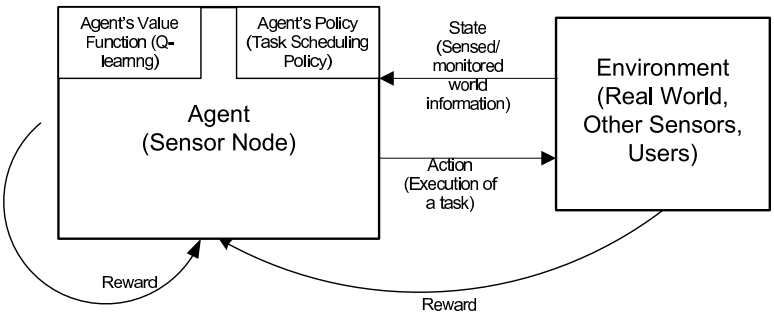


Figure 3.1. Elements of RL applied to WSN.

3.2 Mapping RL elements to resource management problem

In order to apply RL to our resource adaptation problem, we will need to define our problem in terms of the elements of RL. The elements of RL system and their mapping to our problem are as follows:

- *Agent*: As mentioned earlier each sensor node corresponds to an agent in multi-agent reinforcement learning (MARL).
- *Environment*: The world surrounding the sensor node, with which it interacts with.
- *Action*: An agent's action in this context is the application task to schedule. Application is deployed on a sensor node in the form of a set of tasks that a node can perform and each node schedules one task during each time cycle. For example an agent may have the following set of actions: transmit, receive, sample, alarm, actuate, aggregate etc.
- *State*: A set of application defined variables and system variables constitute a node's state. For example, one can include system variables like number of neighboring nodes, residual energy, mobility, capability for outbound/inbound communications etc. Application specific variables such as sensor readings, signal strength etc., may also be part of the state. Number of states in a

system can grow exponentially with increase in state variables and most of the time it is not practical to enumerate all possible states in advance. DIRL uses weighted hamming distance as a method to classify group of similar states and thus reducing number of system states that it needs to keep track of.

- *Policy*: An agent's policy determines what action it will take in a particular state. In our case, this policy determines which task to execute in the perceived sensor state. We have defined policy that consists of predicates as well as exploitation/exploration strategy, as will be discussed later in this section.
- *Reward function*: It provides a mapping of agent state and corresponding action to a reward (typically, a real number) that contributes to the utility. This is how an agent guided through its learning process with an objective to contribute to the goal. Each agent's goal is then to maximize total reward over time. In DIRL, this reward function needs to map application defined optimization parameter (e.g. energy usage, bandwidth utilization, network lifetime etc.) into a numerical reward, as this is the goal that each agent tries to achieve. Each task in DIRL implements a simple reward function which gives the amount of reward (positive or negative) obtained during each execution of that task. For example, if the task is receive, then its reward will be a function of number of messages actually received during its execution as well as amount of resources consumed that it is trying to optimize. This reward function can be as simple as returning a pair of constant values or may be complex considering various optimization constraints.
- *Value function*: This is an important aspect of any RL system. It defines what is good for an agent over long run including current as well as possible future states and not just immediately as described by reward function. But essentially, it is built upon reward function values over time and hence its quality totally

depends on reward function. We use Q-learning, a form of RL that has intrinsic value function defined.

Figure 3.1 illustrates the above elements along with interactions among them.

3.3 DIRL Framework

DIRL is based on Q-learning [9], a form of model-free reinforcement learning. Q-learning is quite simple, demands minimal computational resources and doesn't require a model of the environment in order to operate. Hence it is ideal for implementation on resource-constrained sensor nodes. Along with the simplicity, it also supports state-based learning allowing sensor node to quickly adapt when node moves from one state to other. Each sensor node in DIRL performs task and resource management by executing a Q-learning based algorithm.

As outlined in Section 2.5 Q-learning uses a single data structure, an utility look-up table $Q(s,t)$ across states s and tasks t, which stores knowledge acquired by the agent over time in the form of numerical utilities. For online Q-learning, utility table is updated incrementally using the following:

$$Q(s,t) = (1-\alpha)Q(s,t) + \alpha(r + \gamma e(s')) \tag{3.1}$$

where, $\alpha \in 0,1$ is a learning-rate parameter that control's agent's learning rate. $\gamma \in 0,1$ is a discount-factor.

In DIRL, we have used a simple heuristic where exploration probability at any point of time is given by

$$\varepsilon = minimum(\varepsilon_{max}, \varepsilon_{min} + (S_{max} - S)/S_{max}) - (iii)$$

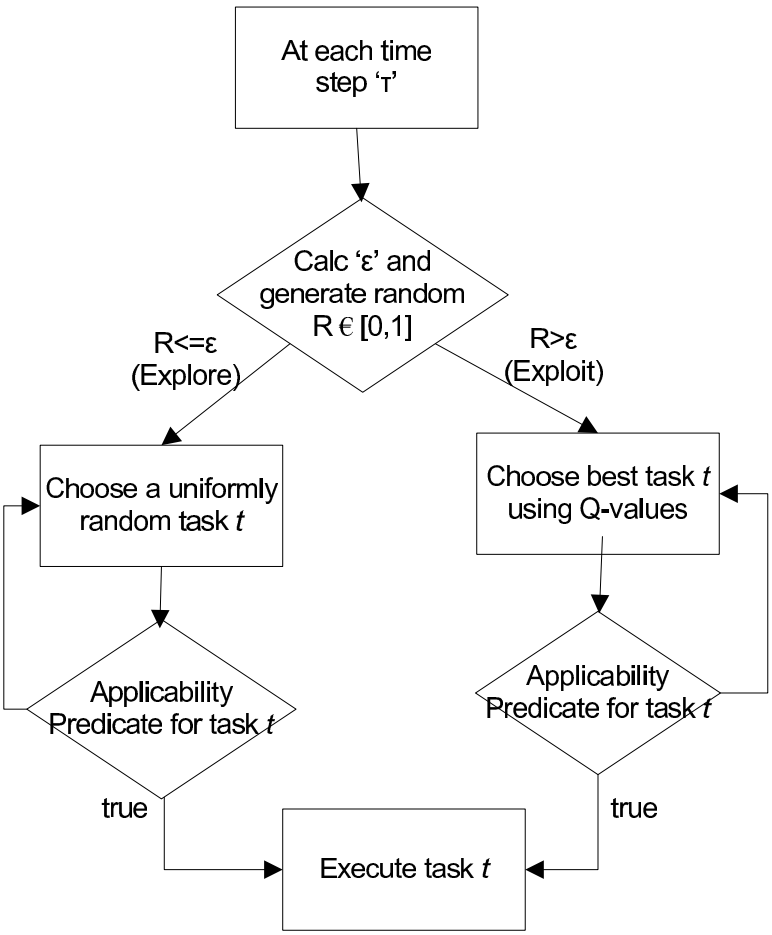


Figure 3.2. Task Scheduling in DIRL.

where ε_{max} and ε_{min} define upper and lower boundary for exploration factor respectively, while S_{max} represents maximum number of states (as obtained from application) that DIRL will try to map and S represents current number of states already known. Thus the above heuristic allows initial exploration with a higher rate and gradually decreasing over time as DIRL is able to map/discover more states. Note that some minimum exploration is required at all times to allow a node to dynamically reconfigure in case of environmental changes.

In DIRL, each node chooses a task to execute at each time cycle either by exploitation or exploration. But, all tasks may not be executable at all times. For

example, *aggregate* task may not be able to execute if there are no readings available to aggregate. Also DIRL needs to honor certain application constraints like latency, quality of readings etc., while scheduling tasks. In order to achieve this, DIRL allows each task to be associated with an applicability predicate that needs to be evaluated true in order for that task to be executed. Thus a task is chosen for execution only if its applicability predicate returns true. Again with *aggregate* task, if application has constraints on maximum latency of a reading, then one can include this constraint in applicability predicate for aggregate task. Figure 3.2 shows the flow diagram of task scheduling using our exploration/exploitation policy along with task's applicability predicate at a particular time-step τ.

Another important characteristic of a RL system is how it handles temporal and structural credit assignment problem [59, 60]. Temporal credit assignment is the problem of propagating reward backwards in time while structural reward is related to propagating reward spatially across states in order to define notion of similar states. Q-learning and its extensions provide support for temporal credit assignment problem dealing with delayed reward [61]. It is important to resolve structural credit assignment problem as otherwise each node will end up with enormous state-space to work with, which is not practical for wireless sensor networks. DIRL uses simple weighted hamming distance between two states in order to resolve structural credit assignment problem. While defining a state representation in the form of system and application variables, application also specifies how much weight each variable carries. This weight specifies to DIRL, how much change in that variable is important for defining state difference. Thus if an application's state representation consists of variables $V_1, V_2, .., V_n$ with corresponding weights $W_1, W_2, .., W_n$, then DIRL uses this

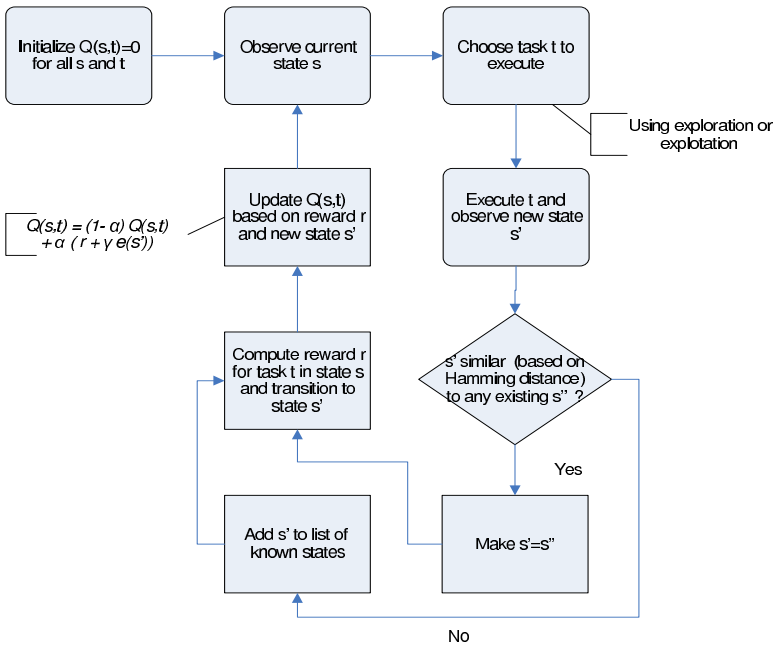


Figure 3.3. DIRL Algorihtm performed by each node.

information to determine if two given states s_1 and s_2 are similar or not, by calculating hamming distance between them as follows:

$$H(s_1 - s_2) = W_1 * |(V_1(s_1) - V1(s_2))| + W_2 * |(V_2(s_1) - V_2(s_2))| + \cdot\cdot$$
$$+ W_n * |(V_n(s_1) - V_n(s_2))|$$

If this hamming distance is less then a threshold value then two states s_1 and s_2 are considered to be similar and has only one entry in Q data-structure.

DIRL needs the following as inputs from the application:

- Set of tasks that application needs to perform in some priority order. Note here that priority will be important only until Q-values are not established or if the two tasks have similar Q-values.

- Associated with each task, an applicability predicate incorporating any application specific constraints and reward functions guiding towards optimization goal.
- A state representation consisting of a combination of system and application variables, along with their weight that will be used in determining weighted hamming distance to aggregate similar states.
- Maximum number of states that DIRL should try to explore. This gives an upper bound on number of states in the system and DIRL will not try to identify any more states beyond this number. If the need arises, one can tune hamming distance threshold to accommodate new states into existing set of similar states.

Once the above information is available, DIRL performs the algorithm given in Figure 3.3.

3.3.1 DIRL Framework using WoLF algorithm (DIRLWoLF)

In previous section, DIRL framework was presented based on Q-learning as underlying learning algorithm. We claim that our approach of applying reinforcement learning to resource management in WSNs is generic and can utilize any learning algorithm. To validate this claim, we also implement DIRL on top of Win Or Learn Fast (WoLF) [62] algorithm. WoLF uses game-theoretic concepts to properly apply reinforcement learning to multi-agent systems. In Q-learning, each agent is acting independently ignoring the presence of other agents. But in most WSN applications, agents (sensor nodes) require to interact with other agents and hence they need to adapt to other agents' behavior. Since other agents also learn and adapt their behavior, the optimal set of actions for each agent are changing with other agents' adaptation. Bowling and Veloso in [62] show that even though agents employing

Q-learning are rational, they may not converge to an equilibrium policy in multi-agent settings. They proposed WoLF principle which utilizes a variable learning rate approach to achieve both rationality and convergence in multi-agent environment.

The principle idea of WoLF is to learn quickly when losing (when doing worser than expected) and cautiously when winning. This helps in convergence by giving more time for other agents to adapt to changes in any agent i's policy which may appear beneficial. At the same time agent i is allowed to adapt quickly when other agents' policy changes that is harmful. WoLF PHC (Policy Hill Climbing) uses two learning rates $\delta_l > \delta_w$. The agent uses one of these parameter based on whether it is currently winning (δ_w) or losing (δ_l). Along with current expected utilities (Q table) as in Q-learning, WoLF also maintains current average policy and updates the average policy based on above learning rate $\delta \in (0, 1]$. With $\delta = 1$, WoLF becomes equivalent to Q-learning. Thus at the end of each time-step after executing action t, average policy $\pi(s, t)$ is updated as follows:

$$\pi(s, t) = \pi(s, t) + \delta_{st} \tag{3.2}$$

Losing or winning of an agent is determined by comparing current expected utility (Q value) of taking an action with an expected utility of average policy. For further details about WoLF algorithm, please refer to [62].

3.4 Application using DIRL: Object Tracking

The DIRL design has been motivated by the need for generalization so that various classes of WSN applications can be built on top of it. Applications which require autonomous adaptation in dynamic environments benefit the most from DIRL. We will next show how an object tracking application can be implemented using DIRL

framework described in the previous section. We have chosen this application as it is widely used in WSN literature concerned with middleware developments [33, 38, 26] and can help with comparative analysis.

Tasks involved in object tracking application along with the corresponding reward function and applicability predicate are defined below:

- *sample*- Take a sensor reading which in this case is signal strength of any target object.

  Reward Function: $(noOfSensedEvents * expectedPrice) - energySpent$

  Applicability Predicate: $remainingEnergy > threshold$

- *transmit (Tx)*- Transmit a message to next hop towards the base-stations.

  Reward Function: $(noOfMsgsTransmitted * expectedPrice) - energySpent$

  Applicability Predicate: $noOfOutboundMessages > 0$

- *receive (Rx)*- Turn radio to receive mode to listen for incoming messages.

  Reward Function: $(noOfMsgsReceived * expectedPrice) - energySpent$

  Applicability Predicate: always

- *aggregate*- Aggregate two or more local and remote same target readings into single reading (we used simple *last value* as our aggregation function).

  Reward Function: $(noOfSamplesAggregated * expectedPrice) - energySpent$

  Applicability Predicate: $noOfSamples > 1$ and $timeFromLastReporting < ncycles$

- *sleep*- Put CPU and radio in sleep mode to minimize battery consumption.

  Reward Function: $expectedPrice - energySpent$

  Applicability Predicate: always

Table 3.1 shows expected price set for these tasks. Here we want to optimize energy usage among all sensor nodes as can be seen from the reward function which penalizes each task with the amount of energy consumed. The separation of expected

Table 3.1. Expected price for tasks in object tracking application

Name	**Expected Price**
Aggregate	0.2
Tx	0.1
Rx	0.2
Sample	0.05
Sleep	0.001

price from reward function allows dynamic external tuning of reward function and hence the overall system behavior by changing the expected price part (only) of the reward function over-the-air [38]. But changing reward function may also invalidate learned utilities (Q-values) and hence Q-values may not converge to correct equilibrium. Refer [9] for more details on convergence of Q-values. But it is possible to discard learned utilities and start over again whenever reward function is updated. Also note that expected price and reward functions are designed in order to reward node only if task execution results in success. Thus if a node schedules task receive, then node will get positive reward only if one or more messages are received in that time step, otherwise it will receive penalty proportional to energy consumed during the time step. Applicability predicates are quite simple and self-explanatory. Here, we don't want a node to perform sampling if its energy is below a certain threshold, to ensure the node's availability for routing messages when required. This can be easily implemented in applicability predicate for the sample task as shown above.

Our state representation consisted of the following variables and their weights: have one or more neighbors (1.0), successful in recent sampling (1.0), successful in recent receive (1.0), signal strength (or quality of reading) (0.1). We used threshold hamming distance of 1.0 and maximum number of states was set to 5.

We would like to point out here that it is also possible to apply Q-learning technique hierarchically in order to schedule sub-tasks for any given task. For an example for the sample task above, consider a sensor node with radar having three heads [33] and at any time it can activate only one of them to detect target. These three heads can be considered as three sub-tasks for *sample* task. Thus DIRL can be used to learn utilities of activating each head in different state and thus determining which one will be the best in a given state. Similarly *transmit* task may need to choose one neighbor out of n next hop neighbors towards base-station. DIRL by applying Q-learning to sub-tasks can learn which neighbor provides highest utility for successful transmission of message to base-station.

3.5 Simulation and Analysis

To demonstrate feasibility of the proposed DIRL scheme as well as the working of our object tracking application, we use J-Sim [63] as our simulation software. J-Sim has support for simulation of wireless sensor networks [64].

3.5.1 Simulation Setup

Figure 3.4 demonstrates a simple simulation scenario consisting of a total of 5 nodes: one base station, three sensor nodes and one object (target) node. Target object is initially closer to node 2 and then as time progresses it moves away from node 2 and finally towards node 3.

We assume here that the network topology is known (is determined outside of DIRL) and that node 1 is acting as an intermittent node that connects both nodes 2 and 3 to the base-station. Note here that each node has no information of what tasks are better for them and they try to learn over time utility of their tasks. Thus node 2 doesn't know that it needs to sample (as object is nearby) and node 1 doesn't know

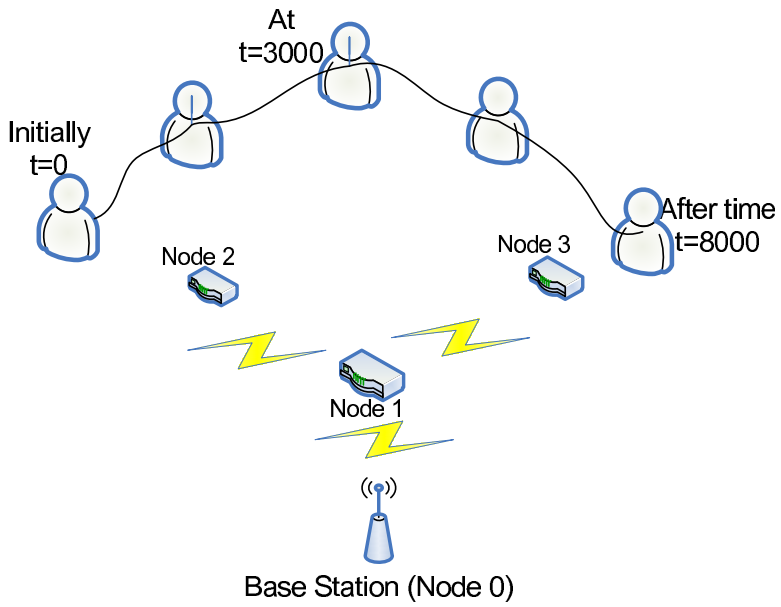


Figure 3.4. Simulation scenario for object tracking.

that it needs to act as a router in this scenario. Figure 3.5 shows task executions for 3 sensor nodes. Node 2 immediately learns that it is getting paid to sample (as target is nearby), while node 1 after some time learns that it is getting paid to listen and route messages. In the middle of simulation time, as target is out of reach of all sensor nodes, they all are sleeping as none of the tasks are getting rewarded. Again towards the end, node 3 starts sampling while node 1 acting as router once again. This shows that nodes effectively learn what task they are supposed to perform in order to achieve their goal. DIRL can help resolve many task scheduling and task synchronization related problems prevalent in WSN.

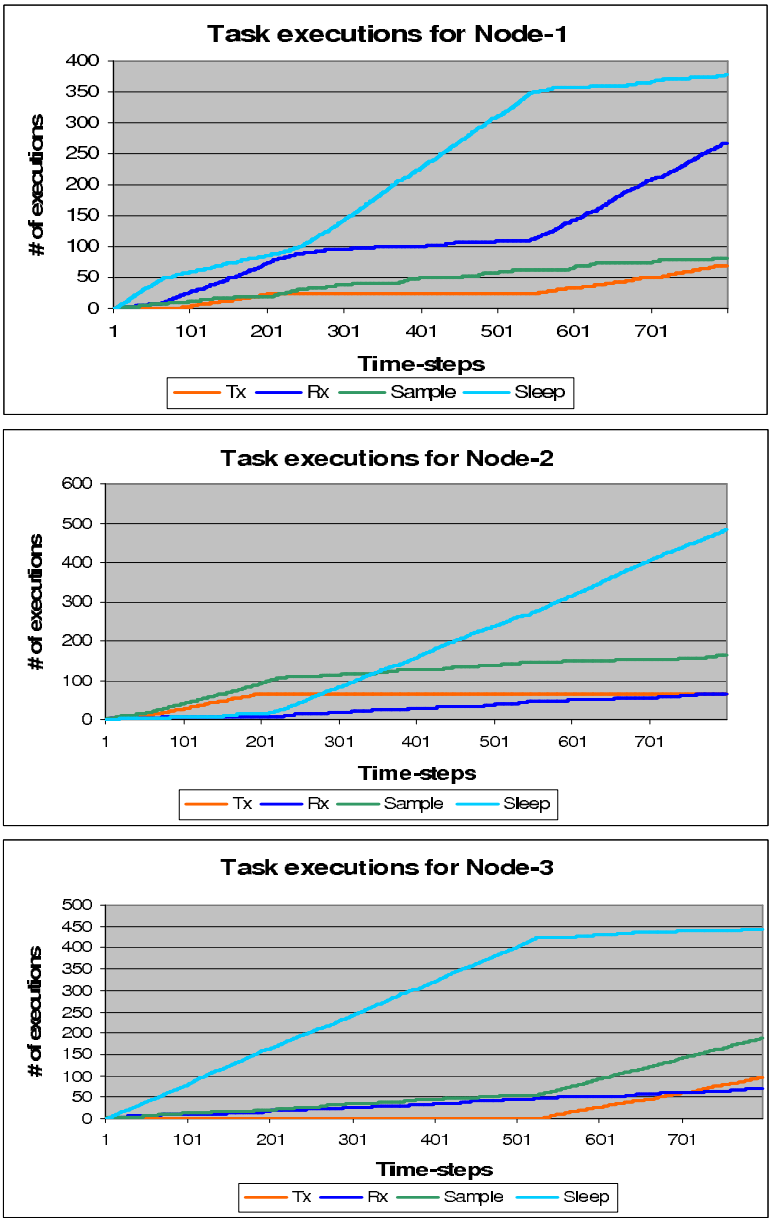


Figure 3.5. Task Executions for 3 sensor nodes.

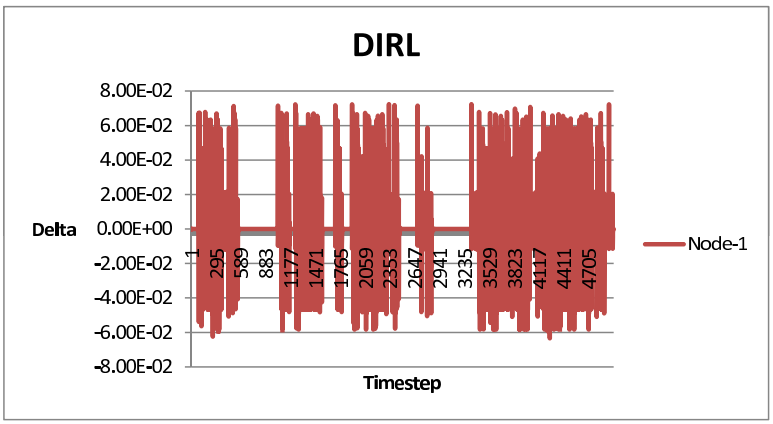


Figure 3.6. Convergence of Q-learning (DIRL).

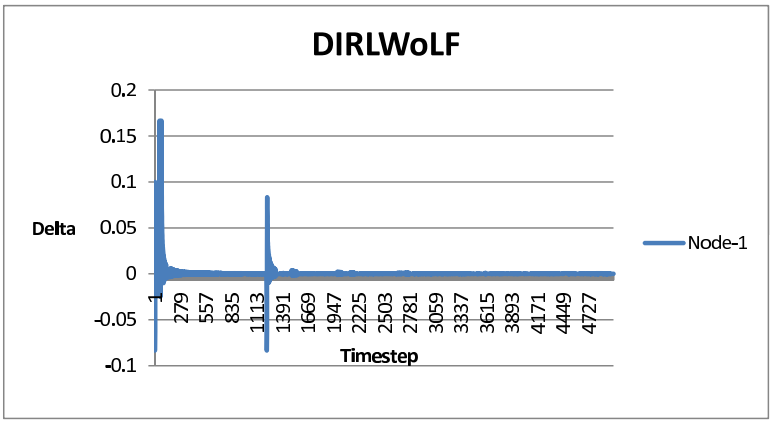


Figure 3.7. Convergence of WoLF (DIRLWoLF).

3.5.2 Convergence of Reinforcement Learning Algorithms

Convergence is a very important property for reinforcement learning algorithms as ability to converge ensures that an algorithm will stabilize to an equilibrium policy and hence provides demarcation of learning phase. In our framework, as we are interested in mainly real-time adaptiveness of sensor nodes and not much of the equilibrium policy, convergence is not much of concern. Though, it is important to analyze system's behavior in terms of convergence as some applications may require some sort of equilibrium guarantees.

Figures 3.6 and 3.7 show convergence behavior of Q-learning (DIRL) and WoLF-Policy Hill Climbing (DIRLWoLF) respectively. For DIRLWoLF, the plot is depicting δ which is the value used to update average policy using equation 3.2. For DIRL, δ is the difference between Q-value after and before the update, thus at the end of time-step τ, δ can be given as

$$\delta_{st} = Q(s,t)_{\tau} - Q(s,t)_{\tau-1} \tag{3.3}$$

For convergence, δ value for each state/action pair should converge. But as can be seen in Figure 3.6 Q-learners always oscillate whenever node has any activity even if same state has been explored and learned earlier. In case of DIRLWoLF (Figure 3.7, it does some initial oscillation but quickly builds up average policy and converges for each new state. Hence WoLF has better convergence compared to Q-learning. This means in steady-state, WoLF can potentially provide more predictable results compared to Q-learning.

3.5.3 Performance Analysis

In this section we present comparative performance analysis of DIRL and DIRL-WoLF against three other schemes as follows:

1. *SIMPLE*: Each node performs a simple scheduling algorithm without trying to adapt or conserve energy by sleeping. Each node is always active and available to perform *Sample* or *Rx* or *Tx* tasks. As nodes are always active, this scheme should provide the best tracking of the target object but at the cost of worst energy efficiency and provides an upper bound to energy usage.
2. *RANDOM*: Each node executes an available task at any given state which is randomly choosen from uniform distribution. This scheme provides a base line

Table 3.2. Parameters used for simulation

Parameter	**Value**
Minimum exploration (ϵ_{min})	0.05
Maximum exploration (ϵ_{max})	0.3
Discount Factor:Q-learning(γ)	0.5
Learning Rate:Q-learning (α)	0.5
Delta Winning:WoLF (δ_w)	0.4
Delta Lossing:WoLF (δ_l)	0.8
Time-step (τ) sec	10
Energy Sample (J)	8.41 x 10^{-5}
Energy Route/RX+TX (J)	8.42 x 10^{-3}
Energy Sleep (J)	8.0 x 10^{-6}

to compare reinforcement learning based algorithms which should be always better than random task execution.

3. *SORA*: This scheme uses a simple heuristic based reinforcement learning as described in [38].

We present results for a scenario with 10 sensor nodes each of which is up to two hops away from base-station and random target movement over a grid of size 500 x 500m. Parameters used for simulation are given in Table 3.2. All results are average of 10 runs and are shown with confidence interval of 95%.

As the goal of our framework is defined in terms of rewards, one way to measure its performance is using metric of total reward over time. Figure 3.8 shows total reward over time for all the five algorithms studied. Note here that total reward is captured by adding individual rewards of all nodes at each time-step. We can see here that DIRL outperforms all other algorithms in terms of maximizing individual node reward immediately followed by SORA. DIRLWoLF has little lower individual reward compared to DIRL and SORA. DIRL and SORA have higher learnability and are faster to respond/adapt to changes, while DIRLWoLF uses average policy which is

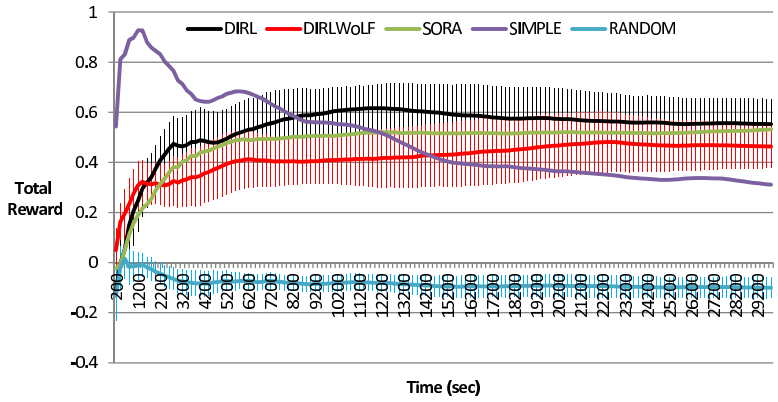


Figure 3.8. Total of rewards over time for random target movement.

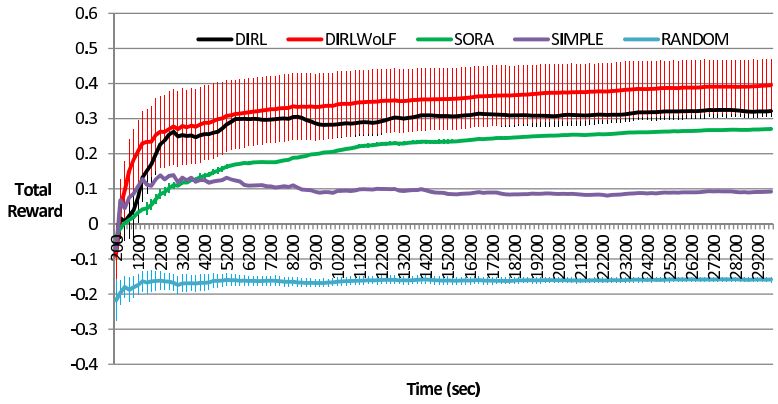


Figure 3.9. Total of rewards over time for stationary target.

cautiously updated and hence slower to adapt. SIMPLE and RANDOM as expected earn the lowest rewards. Figure 3.9 gives total reward for a different scenario where target is stationary and hence global system state is not changing that often. In this case, DIRLWoLF does perform better compared to DIRL and SORA. This can be mainly attributed to better applicability of DIRLWoLF in multi-agent settings where policy learned by each node is dependent on adaptation done over time by other nodes and are not independent as assumed by DIRL. If given enough learning time, DIRLWoLF helps improve total reward by utilizing variable learning rate and allowing more time for each agent to adapt to other agents' change in policy.

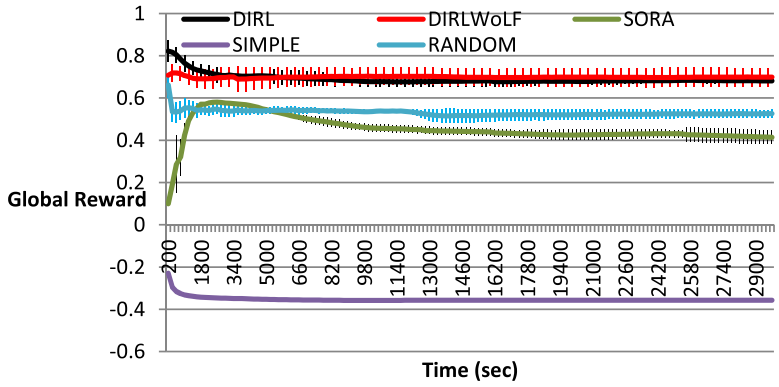


Figure 3.10. Global system-wide reward over time.

In WSN applications, goal is to maximize a system-wide optimization metric. Thus, if we measure performance in terms global reward achieved by entire system, this will give better picture of how system as a whole is performing while individual nodes are attempting to selfishly improve their rewards. Global reward is measured as a function of number of distinct tracking events received by sink and quality (signal-strength) of tracked objects. If there are redundant events for same target for same time-step, only one will contribute towards global reward. Figure 3.10 presents such global reward with respect to time for all five schemes. As we can see, in this case DIRL and DIRLWoLF outperforms all other schemes. Global reward achieved by SORA settles at a lower value (after initial spike) compared to that of DIRL and DIRLWoLF as later ones can exploit state-based learning and quickly adapt to current state for which the utilities are already available. Again as expected SIMPLE has the lowest global reward.

Figure 3.11 compares the five schemes in terms of activity ratio. Activity ratio is defined as a ratio of number of time-steps sensor nodes were active (e.g. executing Sample, Rx or Tx tasks) to total number of time-steps. Thus an activity ratio of 1 denotes that all nodes in system are always active, using maximum energy and

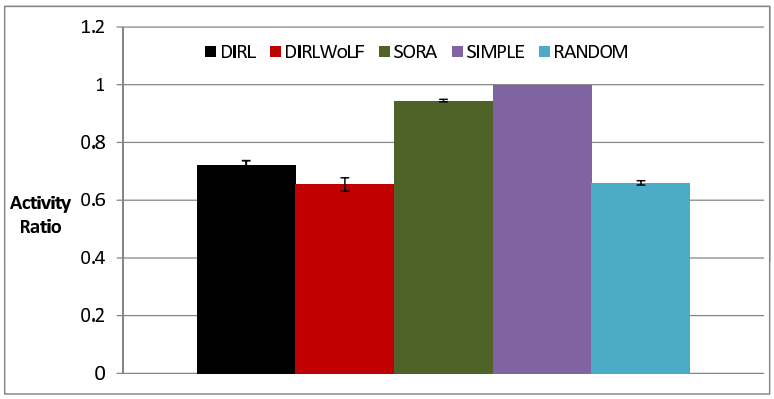


Figure 3.11. Activity ratio of all nodes.

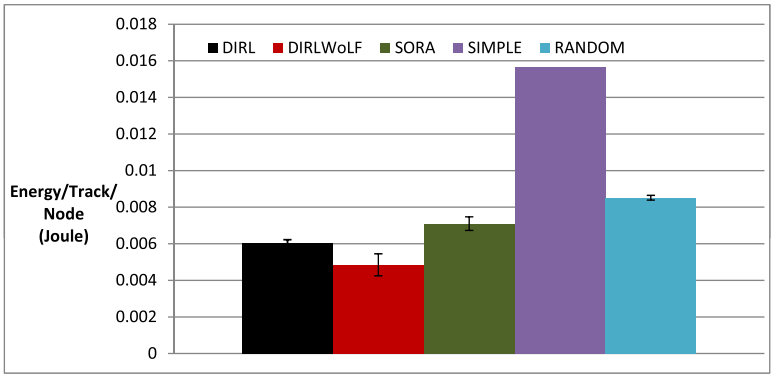


Figure 3.12. Energy consumption per track per node.

don't try to conserve energy by sleeping. DIRL and DIRLWoLF have lowest activity ratio as they manage to conserve energy by adaptively scheduling *Sleep* task when execution of other tasks are not useful and do not generate any reward. SORA on the other hand has a very high activity ratio and is not able to conserve energy as it is more greedy and is short-sighted as it looks at only the current state. Under SIMPLE scheme, all nodes are always active and hence has activity ratio of 1.

Energy consumption per tracking event per node is plotted in Figure 3.12. Energy per tracking event with respect to object tracking application provides a better metric to allow comparison of energy efficiency of DIRL against other existing schemes. Lower energy consumption denotes higher energy efficiency. DIRLWoLF

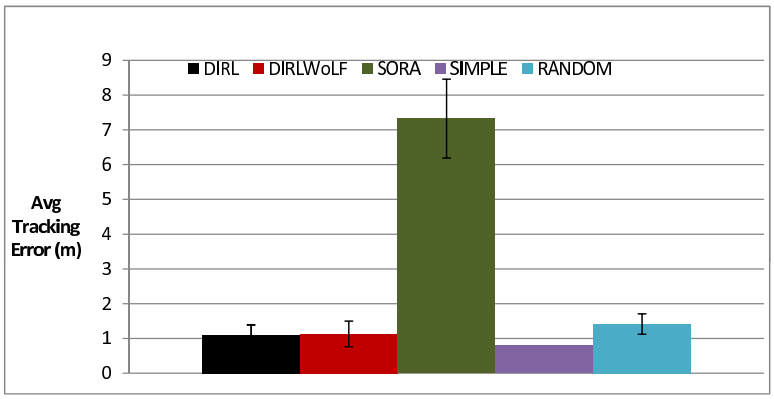


Figure 3.13. Average tracking error.

uses only 0.005 Joule per event per node and hence has highest energy efficiency compared to all schemes studied, followed by DIRL. In this case also DIRL outperforms SORA. SIMPLE provides worst case value for energy efficiency with highest energy usage as all nodes are always active.

Finally, average tracking error is given in Figure 3.13 calculated over all target readings obtained at base-station for one simulation run. Both DIRL and DIRLWoLF has low tracking error and compares to that of SIMPLE scheme which provides lowest possible value for tracking error but using three times more energy[1]. Thus, even though DIRL provides highest energy efficiency, it doesn't compromise functionality i.e. accuracy in tracking the target. SORA has the worst tracking error as tracking events are missing many times and then received in burst resulting in poor object tracking.

3.6 Conclusion

In this chapter, we advocate the use of reinforcement learning for task adaptation and scheduling in wireless sensor networks. We have presented distributed

[1]As all nodes are active, they are always able to track target whenever in range.

independent reinforcement learning (DIRL) framework based on simple but effective and proven Q-learning technique to enable development of autonomous and adaptive WSN applications. Our framework has intrinsic support to handle dynamics and uncertainty prevailing in WSNs. Dynamism is handled by intelligent exploration throughout system lifetime while uncertainty is tackled by probabilistic actions based on learned utility values. We have also exemplified our generic framework with an object tracking application and shown how such application can be implemented on top of DIRL. We present performance analysis of DIRL and results of simulation studies demonstrate its superiority over other approaches.

CHAPTER 4

ENSURING GLOBAL OPTIMIZATION WITH MULTI-TIER REINFORCEMENT LEARNING

The main advantage of using independent learning in DIRL, as presented in last chapter, is that no communication is required for co-ordination between sensor nodes and each node selfishly tries to maximize its own rewards. This works fine when each node in WSN application is acting on its own and does not need to co-operate or compete with other nodes. In other words, if all nodes are acting independently and their actions do not affect others, then any increase in a node's utility cannot decrease any other node's utility and hence will always increase world (system-wide) utility which is merely sum of all nodes utilities over all times. As we will see in the next section, such a system is sub-world factored and will eventually attain a Pareto-optimal point and hence leads towards our system-wide optimization goal.

Most of the real-world WSN applications need some sort of co-operation among sensor nodes and hence nodes cannot work independently. In this case, increase in utility of an individual node may result in reduction of other nodes' utilities and hence may not increase world utility. It is also possible that such system can lead to the Tragedy of the Commons (TOC) phenomenon or Braes' Paradox [10], wherein individual's selfishness leads to significantly lower global utility. Such phenomena can be avoided by carefully designing agent's utility functions as well as constraints under which agent performs task selection. In other words, we need to make sure that individual's utility is aligned' with the global (or world) utility, i.e. any increase in agent's private utility because of its action will also result in increase of world utility.

A real-world example of this would be human economy [11], wherein each individual tries to maximize his private utility in the form of income, career advancement etc. If there are no constraints over how individuals can operate and maximize their utility, he can operate at cross purposes to other human beings and thereby leading to the downfall of the global utility (say GDP). Here government regulations act as necessary constraints and modifications to human utility function in order to ensure its alignment to global utility. Regulations are designed such that any increase in human utility will also cause increase in global growth of the economy.

In our framework, we do not have to deal with virtues of trust and honesty among sensor nodes as the design of utility functions of all nodes in the system is in our hand and hence they can be coded' to be honest. Nevertheless, the problem of attaining sub-optimal system wide utility is still required to be addressed, even if nodes are honest. This is mainly because of imperfect/partial knowledge that each node has for the rest of the system. This partial knowledge of one node can cause it to perform an action which may not benefit the overall system. Hence, key to achieving higher global utility lies in the design of private (individual node) as well as global utility functions such that they are aligned. In this chapter, we investigate the adaptation of COIN theory to address this critical problem in sensor systems.

Another drawback of using DIRL in real-world application is determining reward functions/price settings. It is difficult to investigate different aspects of system dynamics and choose reward settings for each resource/task. Also these settings need to be changed on the fly whenever overall system state and/or application requirements change. Hence, hand-tuning of reward-settings is not acceptable. COIN based macro-learning can help learn the right settings for these rewards (private utilities) for individual nodes and no domain expertise is required to set them. Macro-learning

can update micro-learners utility function as and when application state/requirement changes and always steer them toward global optimization goal.

In the next section, we present further details about COIN theory and how we have adapted it to the problem of resource management in sensor systems in order to steer system towards higher global utility and to provide some type of guarantee towards achievement of system goal.

4.1 Concepts of COIN theory in WSN Resource Management Context

The problem of resource management in WSN is setup in terms of COIN in order to apply its concepts. Consider a WSN consisting of N nodes, evolving across a set of discrete, consecutive time steps $\tau \in \{1, 2, 3, ..\}$. Let $\xi_{n\tau}$ be an element of a vector space $Z_{n\tau}$, representing state of a node n in our WSN at a particular time step τ. Here state of a node consists of not only its application and system variables, but also node's actions that are directly visible to the outside world (including other nodes in the system). Following this convention, $\xi_\tau \in Z_\tau$ denotes a global state of our system, combining actions/variables from all nodes, at a particular time step τ, while $\xi \in Z$ is the vector of global state at all times. ξ can also be referred as world-line [11] in space Z over all time steps. The goal of our framework is then to determine an optimal world-line ξ by maximizing some system-wide global utility function of ξ i.e., $G(\xi)$ and then steering system along that world-line. Each node n in our framework is trying to maximize its private utility function say $q_n(\xi)$. Thus, we need to show here how and under what conditions, the framework can determine optimal ξ, given each node trying to maximize its private utility function $q_n(\xi)$ (Q-learning value function).

The salient features of COIN theory [11, 55] along with their applicability to the resource management problem in WSN are described below:

- A *collective* is defined as a multi-agent system wherein each agent is adaptively trying to maximize its own private utility function, while at the same time there is system-wide performance criteria defined to rate the behavior of the entire system. Thus, our system of wireless sensor networks comprising individual sensor nodes adaptively trying to maximize their utility function, in order to achieve system-wide goal is a collective.
- Most of the collective system focuses on *forward* problem of how the localized attributes induce a global behavior and thereby determining system performance. COIN on other hand addresses the *inverse* problem of designing a system to induce behavior that maximizes world utility. This is done by designing either private utility functions or incentives to private utility functions. Likewise, our objective is to design reward/utility functions used by individual sensor nodes in the WSN so that system-wide utility can be maximized.
- *Subworlds* are sets making up an exhaustive partition of agents. For each subworld, w, all agents in that subworld share the same subworld utility function $g_w(\xi)$ as their local utility functions. Accordingly, consider each subworld to be a set of agents that collectively have the most effect on each other. In this situation, by and large, agents cannot work at cross-purposes, since all agents that affect each other substantially share the same local utility. WSN applications mainly being data-centric, a chosen subworld is a set of sensor nodes involved in a data stream i.e. from data-source to data-sink. For example, all nodes involved in a particular data stream from data sensors, to aggregators and collectors will be part of single sub-world as they all have immediate and considerably high effect on each-other and hence share single utility function. This sub-world definition suggests that sub-world formation is dynamic and can change along with the state of the system. Figure 4.1 marks three such

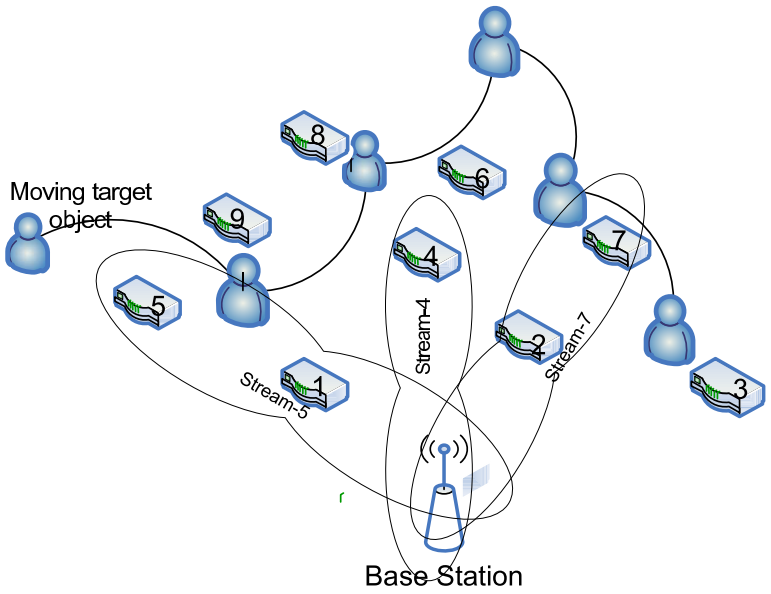


Figure 4.1. Data-stream subworlds in WSN for object tracking application.

data-stream subworlds in a sensor network comprising of 10 nodes in an object tracking application.

- Associated with sub-worlds is the concept of a (perfectly) *constraint-aligned* system. That is a system in which any change to the state of the agents in sub-world w_i at time τ will have no effect on the states of agents of sub-world w_j $(i \neq j)$ at times greater than τ. Intuitively, a system is constraint-aligned if no two agents in separate sub-worlds affect each other, so that the rationale behind the use of sub-worlds holds. Most of the real-world systems are not perfectly constraint-aligned and same is the case with WSN. A change of state in a node involved in one data-stream (and hence one sub-world) may affect state of other node in near-by data stream. But the effect here will be probably much less compared to the effect on a node in the same data-stream. Nevertheless, this is an assumption that needs to be experimentally validated.

- A subworld-factored system is one where for each sub-world w considered by itself, a change at time τ to the states of the agents in that sub-world, when propagated across time, results in an increased value for $g_w(\xi)$ if and only if it results in an increase for $G(\xi)$. Mathematically, a system is subworld-factored if the following holds true for all pair of states ξ and $\acute{\xi}$ that differ only for subworld w:

  $g_w(\xi) \geq g_w(\acute{\xi}) \Longleftrightarrow G(\xi) \geq G(\acute{\xi})$

  For a subworld-factored system, the side effects on the rest of the system of w's increasing its own utility do not end up decreasing world utility. In our problem, if we model a system where each sensor node, as a subworld, selfishly tries to maximize its private utility, then system will not be subworld-factored (assuming sensor nodes are not totally independent). This is because action of one node may be harmful to other more critical nodes and hence may reduce world utility. However if we model a system where each data-stream is a subworld trying to maximize its utility using appropriate subworld utility function (e.g. Wonderful Life Utility Function [11]) $g_w(\xi)$ will not reduce world utility because of relative independence with other data-streams for the period of sub-world's existence. Hence such a system can be considered as subworld factored.

Another requirement of application of COIN theory is to have private utility functions with higher learnability. Learnability is a measure of how well an RL-algorithm can learn to optimize the utility function. For example, learnability of utility functions for team game (where reward of each node is same as that for all nodes in the system at any time step), is much less than those of self-only utility functions. Thus, learnability of a utility function will be high if it is easy to interpret effect of node's action in the reward obtained. In DIRL, each node uses self-only utility functions where it gets immediate feed-back on the action taken and hence enjoys higher learnability.

On the other hand in COIN, all the nodes in a data-stream subworld share the same utility and hence learnability of each node is lower than that of self-only utility functions, but considerably larger than those in a team game. This is because number of nodes in single data-stream is just a fraction of total number of nodes in WSN. Wonderful Life utility function plays an important role in designing a subworld-factored system because of the fact that a constraint-aligned system with wonderful life (WL) subworld-utilities is subworld-factored. If $CL_w(\xi)$ is defined as vector ξ modified by clamping the states of all agents in subworld w across all time to a null vector (or say 0), then WL utility of w is:

$$g_w(\xi) = G(\xi) - CL_w(\xi) \tag{4.1}$$

This definition of WL utility is same as setting WL utility to world utility when considering that subworld w had never existed. Thus a subworld's utility will be high only if that subworld contribution has also increased world utility. We are using WL utility for macro-learning among sensor nodes which in turn is used to set DIRL's private utility functions i.e. $q_n(\xi)$.

COIN theory proves that a collective system which is subworld-factored and has higher learnability, eventually reaches a Nash Equilibrium point where all nodes are fully rational in optimizing their utility functions [11, 55]. COIN theory shows that this Nash Equilibrium point is also the Pareto optimal point of the system. From the above description, we can see that a resource management framework using a model of data-stream subworlds with Wonderful Life (WL) subworld utility function is subworld-factored and also has high learnability. Hence such a system will also eventually reach Nash Equilibrium which is also the Pareto optimal point and hence will avoid upsets like Tragedy of Commons (TOC).

We will next describe how such a data-stream subworld scheme including macro-learning and settings of private utilities can be introduced to DIRL based resource management framework.

4.2 Resource Management Framework using Two-Tier Learning

Our design goal is to create a system using a bottom-up approach where each sensor node is responsible for task selection, rather than top-down approach (where some central entity dictates nodes what task to execute) used by many other middleware solutions [1, 5, 7]. The main advantages of bottom-up approach are pro-active and real-time adaptation, no centralized processing requirement for task allocation and minimal communication overhead. But principal challenge of bottom-up approach is how to make sure that system is actually meeting the global application goals and is not just acting randomly or creating chaos. We resolve this issue by using two-layer learning: micro-learning as used by individual nodes to self-schedule their tasks and macro-learning as used by each data-stream subworld to steer the system towards application goal by setting/updating rewards for micro-learners.

As mentioned earlier, the goal of resource management framework is to determine best allocation of task to sensors/resources so that application defined optimization goals, such as energy savings, network lifetime longevity, bandwidth preservation etc., can be achieved while simultaneously honoring application's QoS metrics. QoS may be defined in terms of quality of measured variables (sensed and processed data) or other application constraints like latency, reliability etc. Hence, the system's global utility function $G(\xi)$ should be a function of success towards achieving optimization goal as well as application's QoS metrics which in-turn are based on variables of interest to the application. Also $G(\xi)$, which represents global utility over all time, can

be expressed here as a sum of rewards $\sum_\tau R_\tau(\xi_\tau)$, where R_τ is global reward and ξ_τ is global state at time-step τ. Thus, $R\tau$ are temporal translations of one another, i.e.

$$G(\xi) = \sum_\tau R_\tau(\xi_\tau) \tag{4.2}$$

As global utility function may take many forms and is application specific, we allow application to define $R_\tau(\xi_\tau)$ given the current state of the system as represented by measured variables (data) and optimization parameters. All optimization parameters are represented in the form of running-sum of numerical rewards attached to each data packet. It is also possible to provide generic implementation of $R_\tau(\xi_\tau)$ based on QoS requirements specification and total of rewards from all data-streams. Each micro-learner uses Q-learning as in DIRL and hence their private utility function $q_n(\xi)$ is a Q-learning value function as given by equation 3.1. Macro-learners on the other hand use COIN based wonderful life utility function. All sensor nodes that are part of one data-stream create a subworld and hence will share same utility (reward-for single time unit). From 4.1 and 4.2, wonderful life reward of each agent (node) part of subworld w at time-step τ is given by:

$$g_{w\tau}(\xi_\tau) = R_\tau(\xi_\tau) - CL_w(\xi_\tau) \tag{4.3}$$

In this case, $CL_w(\xi_\tau)$ is the world reward $R_\tau(\xi_\tau)$ after removing all data values that have been reported by data-stream (subworld) w. This nulls out the effect of data-stream w on the world reward, but considers the actual contribution towards improving world reward $R_\tau(\xi_\tau)$. Suppose data-stream w is providing values of a data variable which by itself is not significant, but has higher effect on overall global application goal, w gets higher reward as required. On the other hand if w is contributing to

redundant information provided by data-stream $\acute{w}$ with higher reward, the wonderful life reward of w will be lower as desired, hence discouraging its use. This reward value is used to update reward function of micro-learners for the task they executed for data-stream w. In order to manage resources using this COIN based framework, we made following extensions to the application input for DIRL described in Section 3.3:

- Instead of hand-tuning expected prices associated with reward function of each task (which we found very difficult to do given various system dynamics), in this scheme, expected price is set by macro-learner using WL utility of its sub-world. Again we have used numerical price here to allow macro-learner to update node's private utility functions without incorporating new code.
- Application also provides a global reward function $R\tau(\xi\tau)$ which returns global reward for a time-step τ, given the current state of the system as represented by measured variables (data) and optimization parameters.

Each sensor node in the system has two agents: 1) micro-learner which is self-contained, trying to maximize its private utility using local information only and 2) macro-learner which is part of a subworld containing other sensor nodes linked in a data-stream and sharing same utilities. Once application input is available, system enters into Initialization phase. Note here that initially there are no learned utilities available with either micro-learners or with macro-learners and hence system needs to go through some sort of initialization. Initialization is possible using any of the following three options:

- *Self-exploration and system learning*: Reinforcement learning based system always uses some balance of exploration (trying out some random actions in search of better rewarding actions) and exploitation (choosing actions based on build utilities) to build up its knowledge base. During initialization phase, the rate

of exploration will need to be higher compared to that of exploitation. Hence decisions made by the system during this phase will be more random and may not lead towards a system goal. A given WSN system and the application using it may be flexible enough to allow such initial self-learning phase and thereby building utilities over time. In this type of system, self-learning will be the viable choice.

- *Using domain knowledge*: It is possible to incorporate domain expertise to provide effective and faster initialization phase. This can be done in combination with self-learning for micro-learners. Domain knowledge can be provided to the system in the form of initial expected price (utility values set by macro-learners) for each task. This is similar to providing initial estimate of task's worth for given sensor node. These values can also be determined using application model simulation using self-exploration as described in first option above and then fed to the deployed system. As mentioned earlier, micro-learners have very high learnability and hence can build their utilities quickly.
- *Employing an available sensor-selection scheme*: We can also utilize any of the various sensor-selection techniques as published in related work e.g., MidFusion [65], MiLAN [5] etc. Results from these techniques can be used to initialize macro-learners with expected price for individual sensor nodes.

After initialization, micro-learners and macro-learners have a knowledge base to start decision making and the system is then considered to be in Normal operating phase. During normal phase, micro-learner tries to maximize its private utility function by a modified version of DIRL algorithm as given in Figure 4.2. Micro-learner uses either exploration or exploitation for task selection at each time step τ based on exploration factor and uses hamming distance between two states as a criterion to distinguish separate states allowing reducing state space of the sensor node as done in DIRL.

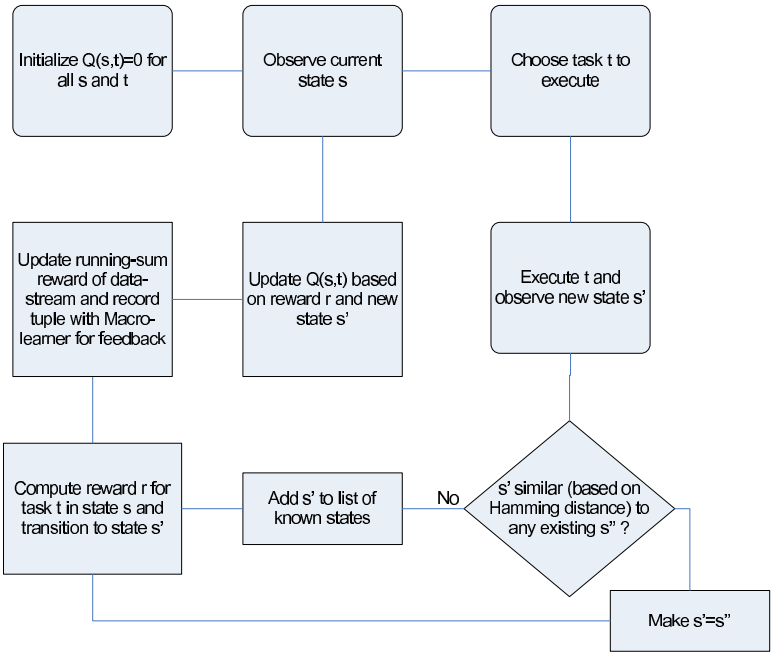


Figure 4.2. Algorithm performed by Micro-learner of each node.

Micro-learner gets an immediate reward after task execution at each time-step which it uses to update its Q-learning value function as well as to update a running sum of reward ($r_{w\tau}$) on the data-packet that it acted on. As the reward obtained is a function of application's optimization goal (e.g. minimizing energy usage), $r_{w\tau}$ will be a measure of how well each data-stream is performing towards application's goal. Micro-learner also updates a running sum of cost ($c_{w\tau}$) as obtained from task's cost function. A tuple consisting of chosen task ID and data-packet ID at time step τ is recorded with macro-learner so that macro-learner can provide future feedback on this task execution.

Data-sink (base-station/controller) is responsible for determining wonderful-life reward $g_{w\tau}(\xi_\tau)$ (given by 4.3) for time-step τ for each data-stream. Here global reward $R_\tau(\xi_\tau)$ is calculated using application specific or generic global reward function given data streams output (measured variables) and total of running-sum of rewards as

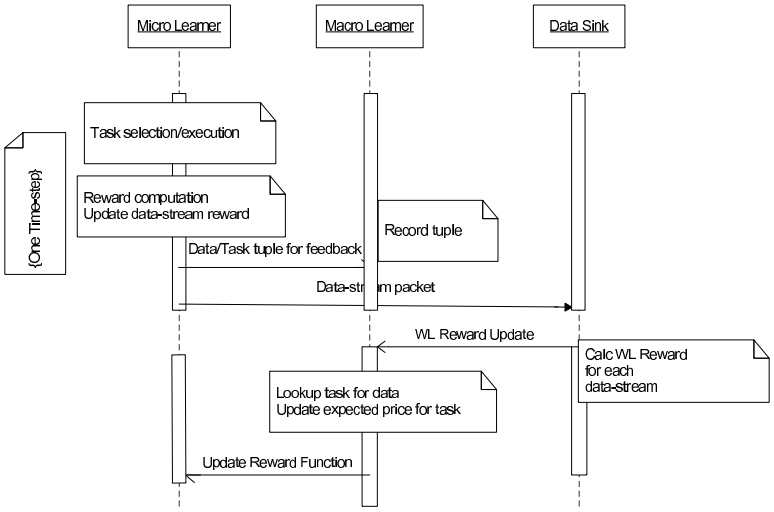


Figure 4.3. Sequence Diagram for Macro-learning process.

well as cost from all data-streams (acting as a measure of optimization parameters). $CL_w(\xi_\tau)$ is also calculated using same global reward function but using all data-streams other then w and discarding reward obtained from w. The difference between WL reward and running sum of local utility based reward of data-stream given by δ_w, is next determined for each data-stream w using equation 4.4.

$$\delta_w = g_{w\tau}(\xi_\tau) - r_{w\tau} \tag{4.4}$$

This value δ_w is then passed down to participating nodes in w along the reverse path of the data-stream. As each macro-learner gets its reward from data-sink, it may arrive after few time-steps. Hence macro-learner maintains a recent history of tuples recorded by micro-learner (a tuple for each time-step), until it receives reward for that time-step. Received reward is matched with recent history using packet id and is used to update utility value (expected price of micro-learner) for the associated

task. Update to expected price e_p for a task is done using the learning rate α as used for Q-learning in chapter 3. Following equation is used for the update.

$$e_p = (1 - \alpha)e_p + \alpha\delta_w \tag{4.5}$$

This process and associated interactions are illustrated in Figure 4.3.

The value of δ_w may need to be transmitted to each node participating in the data-stream w and may be a costly operation. To minimize this, we introduced the optimization to publish WL reward only if it's significant as described below:

$$\begin{aligned} &Negative \quad Reinforcement \quad if \quad \delta < -m < 0 \\ &Positive \quad Reinforcement \quad if \quad \delta > m > 0 \\ &No \quad Reinforcement \quad if \quad m < \delta < m \end{aligned} \tag{4.6}$$

Here, m is determined empirically based on application's reward and cost functions to adjust the rate of WL reward updates. Value of m can be varied to tune the effect of macro-learning on the system and as such, a very high value of m may turn off macro-learning all-together. If both $q_n(\xi)$ and $g_w(\xi)$ are correlated (i.e. generate reward value using same scale/metrics), it is possible to design our global reward function (using reward and cost values of data-stream) so that $\delta_w \to 0$, as our system approaches a steady state. For example, if the value of $g_{w\tau}(\xi\tau)$ is high for a data stream w, this will result in high utility for participating node $q_n(\xi)$. This will further increase for data-stream reward $r_{w\tau}$ for subsequent data collection and thereby reducing the value of δ_w. When $\delta_w < m$, no reinforcements are sent out and system converges for current global state. These ensures that WL updates to entire data-stream will be required only when there is a global state change resulting in a need for adaptation

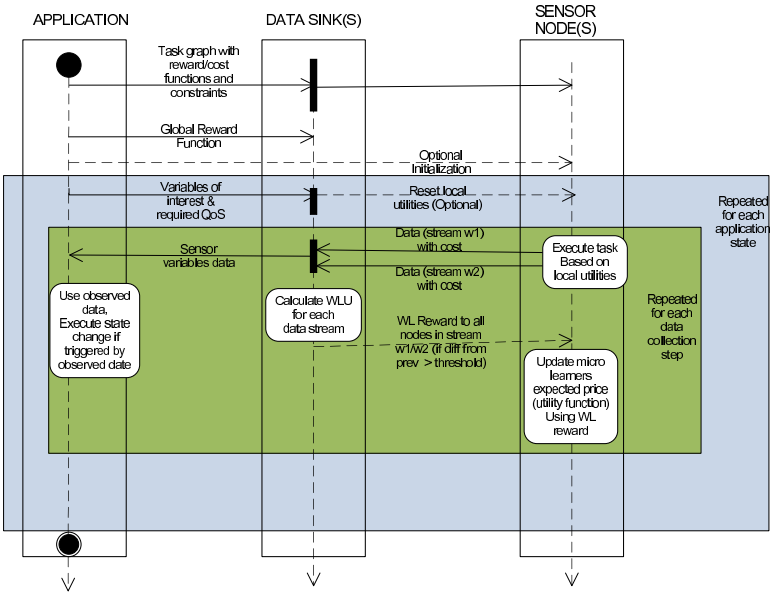


Figure 4.4. Overview of activities between application and WSN using two-tier reinforcement learning.

and learning the new optimum task scheduling strategy. Updates will decrease and eventually diminish as system approaches a steady state.

Computation and transmission of reinforcements for macro-learners by sink can be deferred to a reinforcement window of N time-steps instead of doing at every time-step τ. Thus sink collects data for a window of N time-steps and computes stream reward based on all accumulated data in that window. This allows not only efficient communication but also prevents reward fluctuation by computing global reward over larger time interval.

Figure 4.4 gives a high-level overview of our framework and interactions with the application. As part of application deployment, task graph and associated application constraints as well as reward and cost functions are dispersed onto the nodes of the sensor network. Application also provides global reward function to applicable data sinks. Figure 4.4 also shows optional initialization of sensor node's local utilities. Application can provide variables of interest with their required QoS at any state change.

From this point onwards, each node takes on the responsibility of self-scheduling their tasks and allocating resources based on learned local utilities. All data-streams are evaluated for WL reward at the end of each time-step. WL reward is next distributed to all the nodes in respective data-stream (if significant as determined by equation 4.6). Sensor nodes participating in data-stream updates their local-utility functions based on the global WL reward.

The simple set of micro-learner and macro-learner provides each sensor node the capability to self-schedule tasks, while making sure that the overall system is being steered towards its optimization goal. This scheme allows a sensor node to self-adapt to system dynamics and uncertainty inherent in the WSN. Micro-learner adapts to local state changes (e.g. low battery, neighbor change, nearby target etc) immediately while on the other hand macro-learner provides adaptation at the global level with change in global state or application requirement (e.g. change in QoS of data variables, addition/removal of sensor nodes, etc.). Any such global change ensues a change in wonderful-life utility of macro-learner which changes utilities learned by micro-learner in that direction. The changing reward function of micro-learners may invalidate learned utilities (Q-values) and hence Q-values may not converge faster to equilibrium [9]. But it is possible to discard learned utilities and start over again whenever reward function is updated. Also we are more interested in adaptation as well as convergence of the system as a whole rather than convergence of individual micro-learners.

4.3 Implementing Real-world Applications

Our design of resource management framework has been motivated by the need for generalization so that various classes of WSN applications can be built on top of it. Applications which require autonomous adaptation in dynamic environments

Table 4.1. Implementation of WSN Applications

	Object Tracking	**Intrusion Detection**	**Health Monitoring**
Variables	signal strength (SS), area coverage, target(s) position/track	Security threat confidence level, area coverage	Heart rate, respiratory rate, blood pressure
QoS Requirements	SS greater than threshold and coverage and delay less than threshold	Confidence level greater than threshold and k-coverage	Quality of measured variables greater than threshold
Types of Sensors	Acoustic, seismic, video	Motion detectors, Biometric-reader, RFID, video	ECG, blood pressure, blood flow, EMG etc.
Global Reward Function ($R_\tau(\xi_\tau)$)	f(SS, coverage, tracking delay)-total cost of data acquisition	f(threat confidence level, coverage)-total cost of data acquisition	f(quality of measured variables)-total cost of data acquisition

benefit the most from our framework. We will next show how some of the real-world WSN applications can be easily implemented over our two-tier RL based resource management framework. Variables of interest, QoS requirements, involved sensor types and global reward function for studied applications are given in table 4.1.

4.3.1 Object Tracking

Object tracking application can be deployed on top of heterogeneous WSN to track one or more objects of interest. This may be to provide surveillance for environmental monitoring or for a battlefield. Depending on the usage, application may have minimum sensor coverage area and lifetime requirements. Application may not need redundant information provided by overlapping sensors. Also tasks performed by each sensor node (e.g. sampling, routing etc) can be tuned based on current state of the system, e.g. presence of objects of interest. Hence, by means

of efficient and continuous adaptive resource management over time, it's possible to allow sensors to preserve energy while still meeting application's requirements. Tasks involved in an object tracking application have already been described earlier in section 3.4 along with their reward function. These tasks include transmit, recieve, sample and sleep. We want to optimize energy usage among all sensor nodes as can be seen from the reward function which penalizes each task with the amount of energy consumed. The cost function is ratio of amount of energy utilized for the task execution to node's available energy. This cost function allows quoting low cost for task execution by higher energy node compare to that of a low-energy node. Note that expected price and reward functions are designed in order to reward a node only if task execution results in success. Thus if a node schedules task receive, then node will get positive reward only if one or more messages are received in that time step, otherwise it will receive penalty proportional to energy consumed during the time step.

4.3.2 Intrusion Detection

In intrusion detection application, variety of sensors e.g. motion detectors, RFID, video etc., may be utilized for detecting an intruder; each providing different level of confidence and can have different cost of data acquisition. Large number and types of these sensors may be deployed in a given WSN for redundancy as well as flexibility of choosing the required sensory information. Sensory information required by the application depends on the state, for an example, on detecting high probability of intrusion; application may want to utilize video cameras for highest quality of information. Hence again by properly managing sensor resources, it is possible to optimize their usage while meeting the requirements of application. High level tasks

and reward functions for intrusion detection application are similar to that of object tracking application and are detailed below:

- *sense*- Sense the environment for present of any intruder (either using motion, RFID or video sensor)

  Reward Function: $f(confidenceLevel) * expectedPrice) - energySpent$
- *transmit (Tx)*- Transmit a message to next hop towards the sink

  Reward Function: $(noOfMsgsTransmitted * expectedPrice) - energySpent$
- *receive (Rx)*- Turn radio to receive mode to listen for incoming messages.

  Reward Function: $(noOfMsgsReceived * expectedPrice) - energySpent$
- *sleep*- Put CPU and radio in sleep mode to minimize battery consumption.

  Reward Function: $expectedPrice - energySpent$

4.3.3 Health Monitoring

For health monitoring application also a large variety of sensors can be used each providing one or more health-related variables with different quality as well as cost. For an example, heart rate can be measure by ECG, blood pressure monitor or blood flow monitor [5]. But accuracy and quality of each of them is different and so does the cost of obtaining heart rate. Intelligent resource management can help in choosing less costly sensors in say normal health state while triggering expensive but highly accurate sensor in abnormal state. Input from such health monitoring application as required by our resource management framework is described in Table 4.1. Tasks for health monitoring application are similar to that of intrusion detection with *sense* task responsible for measuring related sensed variable (e.g. blood pressure, heart rate etc) and it's reward function given as $f(qualityOfMeasuredVariable) * expectedPrice) - energySpent$.

4.4 Simulation Setup

We have again chosen object/entity tracking application for the analysis of our framework. All simulations are done using J-Sim [63]. Performance of our two tier learning approach (marked as COIN) is compared against four other schemes:

- *DIRL*- where there is no macro-learning involved and system consists only of individual micro-learners as described in chapter 3.
- *RANDOM*- where each node performs a task randomly chosen from uniform distribution each time,
- *SORA*- which uses simple heuristic based reinforcement learning (micro-learning only)
- *SIMPLE*- where each node performs a simple scheduling algorithm without trying to adapt or conserve energy by sleeping. As nodes are always active, this scheme provides the best tracking of the target object but at the cost of worst energy efficiency and provides an upper bound on energy usage. This scheme will always have activity ratio of 1.
- *ORACLE*- which is an idealistic scheme that assumes each node (magically) knows exactly what task to perform and there is no overhead involved of any kind for managing the system. Thus, this idealistic scheme provides us with a lower bound on energy usage and again best tracking efficiency and accuracy. Again this scheme is hypothetical and is used only for comparison of our performance metrics.

The following performance metrics are considered for our analysis:

- *Activity Ratio*: Activity ratio is defined as number of active tasks (Sample, TX and RX) to total number of tasks executed in a WSN system (including Sleep). A scheme with better resource management should have lower activity ratio as it should allow system to conserve energy by executing Sleep tasks whenever

Table 4.2. Parameters used for simulation

Component	**Parameter**	**Value**
Micro-learner	Minimum exploration (ϵ_{min})	0.05
	Maximum exploration (ϵ_{max})	0.3
	Discount Factor(γ)	0.5
	Learning Rate (α)	0.5
	Time-step (τ) sec	10
Macro-learner	Minimum WL reward (m)	0.25
	Reinforcement Window (N) time-steps	10
Radio	Energy Sample J	8.41 x 10^{-5}
	Energy Route/RX+TX J	8.42 x 10^{-3}
	Energy Sleep J	8.0 x 10^{-6}

possible. A system where all nodes are always active will have activity ratio of 1.

- *Energy usage per track per node*: This metric is heavily used in literature [33, 12] for measurement of energy efficiency for object tracking application. Hence it is useful in comparison of energy efficiency of our approach against other existing schemes. Lower energy usage per tracking event dictates better energy efficiency.
- *Average Track Error*: This metric captures how well the actual tracking is done. It is important for any scheme to achieve acceptable performance in terms of average track error irrespective of the system optimization. This is an average of difference between a target's tracked (perceived) position and the actual position.
- *Global Reward*: As optimization goal of our framework is defined in the form of global reward function $G(\xi)$, one way to measure its performance is using metric of average global reward over time. Global reward function is defined based on target events that are reported to sink and amount of energy consumed (i.e. cost of acquisition) as given in Table 4.1.

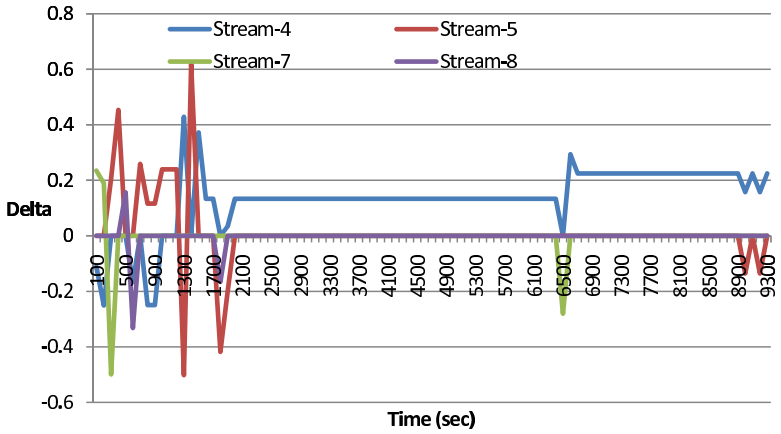


Figure 4.5. Convergence of COIN based two-tier learning scheme.

4.5 Simulation Results

Simulation is performed under variey of network and target scenarios. All results are averaged over 10 simulation runs with different target speed and movement over the grid. Table 4.2 summarizes other important simulation parameters.

4.5.1 Analysis of reinforcement learning

This section evaluates our scheme in terms of convergence to an equilibrium state and global reward. Results are shown for a 10 node scenario with a single target and a grid of 300x300m. Figure 4.5 shows the convergence behavior of our two-tier COIN based learning scheme based on one simulation run [1]. The target in this test was stationary and within range of multiple sensing nodes. Y-axis in the figure represents δ_w (Delta) for different data-streams as given by Equation 4.4. Some streams in this 10 node scenario have been identified in Figure 4.1. At the beginning, multiple streams (stream 4,5,7 and 8) report target's tracking events to sink. The value of δ_w for each stream osciallates into positive and negative territory. This roughly corresponds to the initialization phase described in section 4.2. After

[1]This behavior is verified to be similar for multiple runs

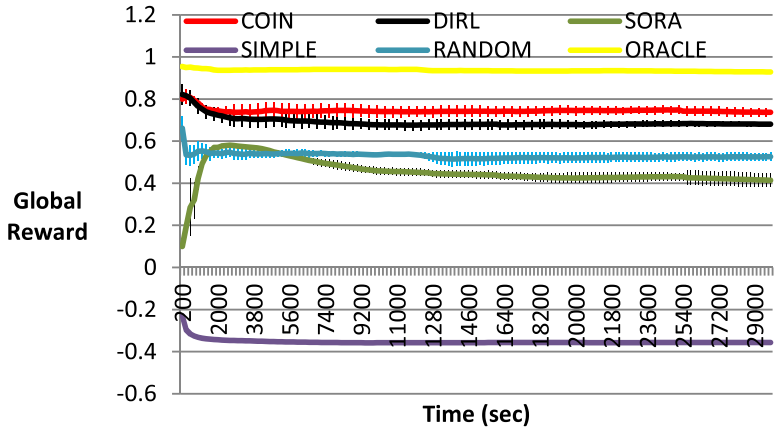


Figure 4.6. Global reward over time for a 10 node scenario.

about 2000 sec, δ_w value stabilizes and in this particular case, system has effectively chosen stream 4 and other streams reporting redundant target data were turned off to preserve energy using reinforcement. At this stage, as $\delta_w \in (-m, m)$ for all streams, no more global reinforcement needs to be sent out and corresponds to an equilibrium state. When the state of system/application changes, we may see some variations in δ_w again until system reaches a new equilirium. This behavior shows that our scheme of two-tier learning is effective in selecting best stream and quickly attains an equilibrium state.

Global reward metric as defined in previous sub-section is plotted for all five schemes in Figure 4.6 over simulation time. Target was moving along with grid with a random movement and a speed of 3.6 kmph. As shown, COIN is able to achieve highest global reward among all studied schemes and is closest to ORACLE. This shows that COIN is able to manage system resources appropriately at all times balancing the cost of acquisition and reward from received data. DIRL is also able to achieve overall higher global reward and stands next to COIN. SORA shows bursts in global reward value initially, but settles down at value lower than DIRL and RANDOM. This is caused by receiving a burst of data all at-once initially (when nodes are able to schedule the right

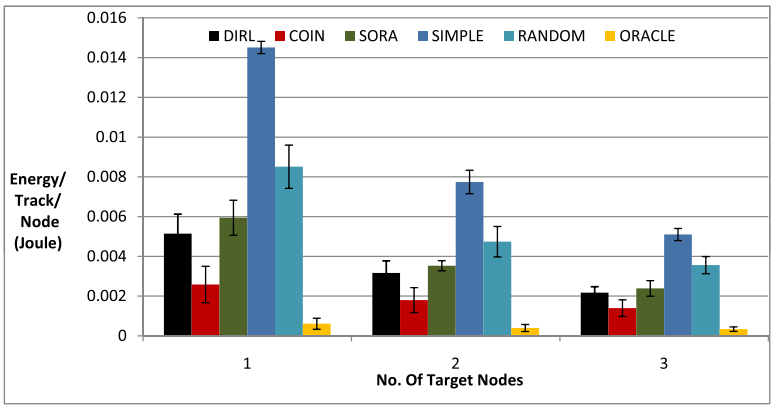


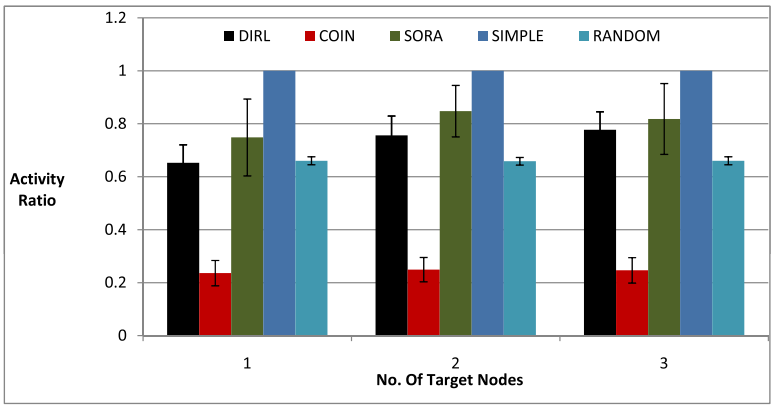
Figure 4.7. Energy consumption per track per node over increasing number of target nodes.

task) and then not being able to track the object at all time. As DIRL and SORA are based on micro-learning only, sensor nodes try to just maximize their personal reward at all times and no consideration is given to system wide performance. Hence system wastes sizable energy in redundant as well as unnecessary sensing and processing. As expected SIMPLE has the lowest reward as SIMPLE results in maximum energy consumption.

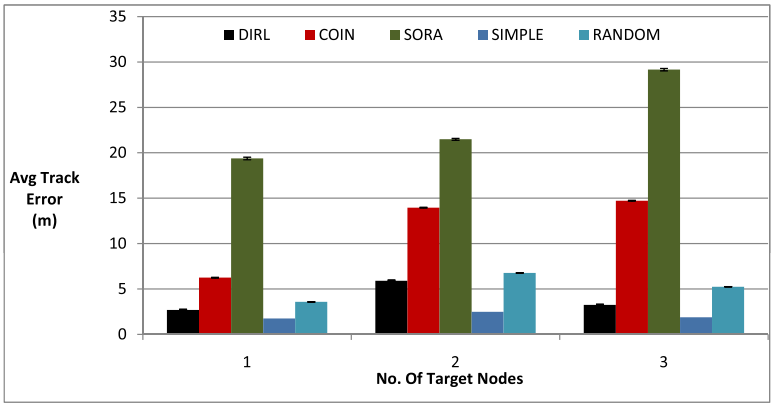
4.5.2 Varying number of target nodes

In this scenario, 10 sensor nodes are placed over 250x250m grid with each sensor upto two hops away and number of target objects is varied from 1 to 3. All target objects are placed randomly over the grid and have random movement at a speed of 3.6 kmph.

Energy consumption per tracking event per node is displayed in Figure 4.7. COIN is closest to ORACLE for all cases and uses just about 0.0025 Joule of energy per track for single target object which is less than half of SORA and DIRL. SIMPLE scheme doesn't try to conserve any energy and hence gives upper-bound to energy consumption which is more than 6 times compared to COIN. As number of target



(a)



(b)

Figure 4.8. (a) Activity ratio, (b) Average tracking error; over increasing number of target nodes.

nodes increases, number of tracking events also increases and hence energy consumption per track decreases for all schemes. Though COIN continues to outperform all other schemes.

Figure 4.8(a) presents activity ratio of all sensor nodes in the system for multiple target objects. COIN maintains a very low activity ratio of just above 0.2 even when number of target increases. This shows effectiveness of COIN in resource management over all scenarios. Activity ratio for DIRL and SORA increases with number of targets. RANDOM has activity ratio of about 1/3 (0.66) which is equal to probability of randomly scheduling SLEEP out of three tasks from a uniform distribution. As all

nodes are always active, SIMPLE scheme has activity ratio of 1. Figure 4.8(b) shows average track error. If an object is not successfully tracked by a scheme all the time (as nearby nodes may be sleeping to conserve energy), that scheme will have higher tracking error. SIMPLE scheme has the lowest tracking error mainly because it has activity ratio of 1 and nodes are always active. Despite the lowest activity ratio, COIN still manages to perform within close limits to DIRL and TEAM. But DIRL does perform little better than COIN in terms of tracking error in general while consuming more energy. This is caused by negative feedbacks received by data-streams that may be non-optimistic at time-step τ (effectively asking them to conserve energy), but can be critical at time-step $\tau + k$, if target object moves towards their direction. This issue can be prevented by using more sophisticated global reward function which uses object's track to predict its future direction and provide reward accordingly. We are currently investigating design of such tracking scheme.

4.5.3 Varying number of sensor nodes

We will present our results for 3 different scenarios consisting of 5, 10 and 15 nodes respectively, with sensors up to two hops away from base-station over a grid size ranging from 200x200m to 500x500m. There is a single target object moving in the grid with a random movement and a constant speed of 3.6 kmph over the simulation run of 12000 seconds.

Figure 4.9 plots the energy consumption per track per node with increasing number of sensor nodes in the system for all studied algorithms to understand how they scale. SIMPLE and RANDOM have highest energy usage and that also increases with increase in number of sensor nodes. Plot shows that difference in energy consumption between COIN, DIRL and SORA is insignificant with 5 sensor nodes, with

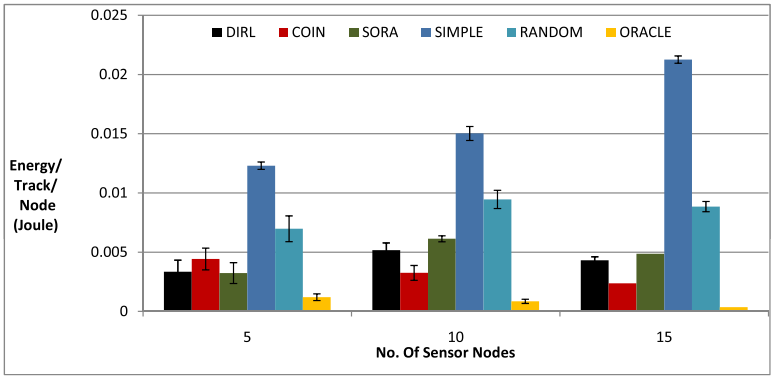
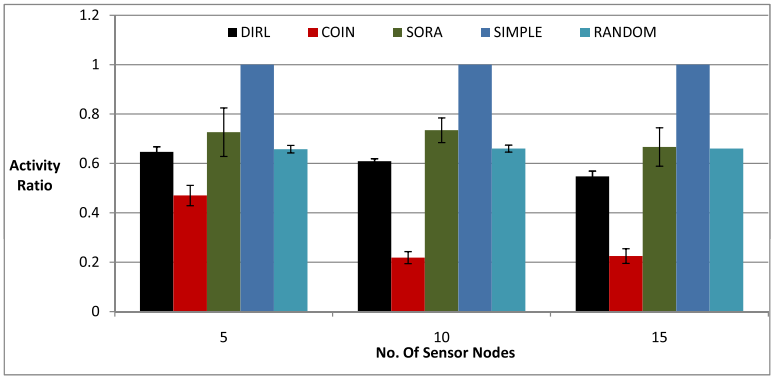


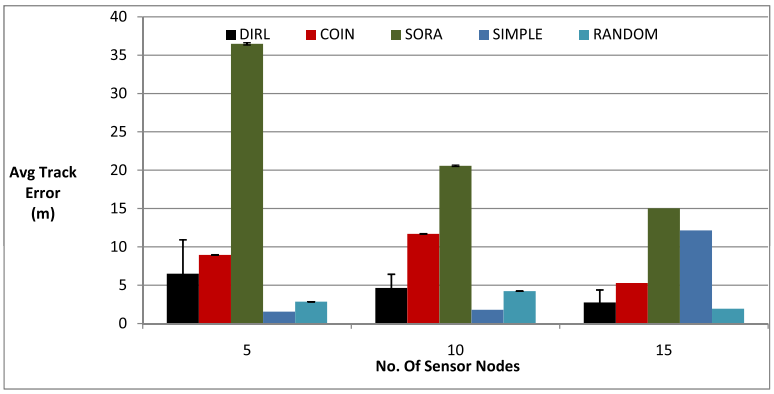
Figure 4.9. Energy consumption per track per node over increasing number of sensor nodes.

DIRL using minimal energy. This is because with less sensor nodes, most of them have to be relatively active and hence difficult to conserve energy by keeping activity ratio low. But as the number of nodes increases, the difference in energy consumption between these schemes is huge. While energy consumption for all schemes increases, COIN is able to lower energy usage with increase in sensor nodes. This shows that COIN is very effective in managing system resources as redundancy and overall size of WSN increases. This is mainly achieved by providing global feedback to sensor nodes in associated data-stream and thereby reducing the redundancy in data collection as well as re-enforcing the more favorable data-streams.

Figure 4.10(a) compares performance of studied algorithms in terms of activity ratio. Our two-tier reinforcement learning scheme (COIN) has the lowest activity ratio closely followed by DIRL, and hence resulting in the lowest energy consumption. As mentioned earlier, with increasing number of sensor nodes, COIN's overall activity ratio decreases. Figure 4.10(b) shows average tracking error for each scheme. In this case DIRL outperforms COIN while SORA has the worst performance as it has a bursty behavior and is not able to track target at all times. Tracking efficiency of DIRL is better than COIN particularly when system changes are very dynamic (only



(a)



(b)

Figure 4.10. (a) Coin activity ratio, (b) Average tracking error; over increasing number of sensor nodes.

few time-steps), as by the time global knowledge is learned and applied, system may have moved to a different state, e.g. a very fast and randomly moving target in a field.

4.6 Conclusion

We presented a scheme for resource-management in WSN using a bottom-up approach where each sensor node is responsible for task selection instead of top-down approach conventionally used by other middleware solutions. This bottom-up approach using reinforcement learning allows development of autonomous WSN appli-

cations with real-time adaptation, minimal or no centralized processing requirement for task allocation and minimal communication overhead. In order to ensure that system is actually meeting the global application goals and is not just acting randomly, we used two-tier learning: micro-learning as used by individual nodes to self-schedule their tasks 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. We used COIN theory to enable macro-learning that can steer system towards application's global goal. Simulation results show that two-tier learning as used here can significantly improve overall performance compared to micro-learner alone or other traditional schemes. Application of COIN theory for setting micro-learners utilities, guarantees eventual achievement of Pareto optimal point and avoids system getting trapped in TOC or other related phenomenon.

CHAPTER 5

RESOURCE-AWARE DATA ACCUMULATION IN SPARSE WIRELESS SENSOR NETWORKS

In previous chapters we have considered resource management only in traditional dense wireless sensor networks (WSN) where sensors use multi-hop communication to send collected data to a sink. There are applications that do not have such a dense network with sensors connected to form a multi-hop path or applications don't need such fine-grained sensing. Examples of such application include monitoring applications in farms, battle-fields, urban traffic and the environment. For such applications, it is possible to consider a *sparse wireless sensor network* where the density of nodes is so low that they cannot communicate with each other through multi-hop paths. *Mobile data collectors* (MDCs) are utilized in sparse WSN to make communication possible between source and sink nodes. MDCs are not resource constrainted and are responsible for collecting data from individual nodes by visiting them in some predetermined or dynamic fashion and optionally disseminating collected data. An MDC can serve either as a Mobile Sink (MS), a mobile node which is also the endpoint of data collection, or as a Mobile Relay (MR), which carries data from sensors to a sink node or an infra-structured AP. In either role, the MDC moves throughout the WSN, and in most cases it is autonomous.

Sparse WSNs with MDCs have many advantages when compared to traditional dense WSNs. First, costs are reduced, since fewer nodes can be deployed, as there is no need for a connected network. Second, as data is collected directly by the MDC from sensor nodes, reliability is improved as a result of less congestion and

collisions. Finally, data collection by MDC can extend the WSN lifetime, as the energy consumption is spread more uniformly in the network with respect to dense (static) WSN, where the nodes close to the sink are usually more loaded than the others.

Although MDCs are not power constrainted and can help extending WSN lifetime, individual sensor nodes sensing data are significantly resource constrained and hence energy efficiency for sensor nodes is critical. To address energy constraints, sensor nodes normally adopt duty cycling where each sensor node is awake for only small fraction of time while conserving energy by sleeping at other times. In order for data collection process to be successful, a sensor node needs to be active when MDC is in its communication range and needs to detect MDC's presence. Thus, contact detection and energy conservation are two major challenges for data collection in sparse WSN. Energy efficiency can be achieved by adaptively tuning the duty cycle to MDC's arrival pattern. If a node can execute at a higher duty cycle when probability of MDC being in communication range is high, contact detection ratio can be significantly improved. This requires an efficient MDC discovery process that can learn MDC's mobility pattern given high level of uncertainty and use that knowledge to adaptively tune sensornode's duty cycle.

In this chapter, we address the problem of the MDC discovery by exploiting DIRL. We propose an resource aware data accumulation (RADA) framework based on DIRL that learns MDCs mobility pattern over time and utilizes this knowledge for autonomous tuning of node's duty cycle. Reinforcement learning is very useful for interactive/online learning in dynamic uncertain environments as found in sparse WSNs with MDCs. DIRL is quite simple, demands minimal computational resources and doesn't require a model of the environment in order to operate. Hence it is ideal for implementation on resource-constrained sensor nodes. RADA classifies duty-cycle

into three tasks viz. very high, low and very low duty cycle and learns the usefulness of each duty-cycle at any given state. RADA's state definition allows it to capture time and associated MDCs mobility pattern. By using a generic state definition for reinforcement learning, RADA is able to learn a wide-range of MDC mobility patterns. We show that the proposed solution is adaptive, generic and efficient for resource-aware data collection in sparse WSNs with MDCs.

5.1 System Overview

Before introducing the resource aware data collection framework, it is necessary to present our reference network scenario and the targetted applications for our framework. In this section, we first describe a reference scenario with details on discovery and data transfer phases involved in the data collection process along with the terminology used. Next, we provide brief classification of sparse WSNs applications for which we are designing a generic solution and corresponding MDC mobility scenarios.

5.1.1 Reference network scenario

The reference network scenario is depicted in Figure 5.1(a). We assume that the network is sparse so that, at any time, the MDC can communicate with at most one sensor node and also a sensor node communicates with only one MDC at any instance of time, even if multiple MDCs are present.

In the discussion below, we consider data collection process between the MDC and an arbitrary static node N_i, $i \in \{1, N\}$, where N is number of nodes in the sparse WSN. Data collection takes place only during a *contact*, i.e., when sensor node N_i and the MDC are within communication range of each other. The area within the communication range of a sensor node N_i is called *contact area*, and the overall time

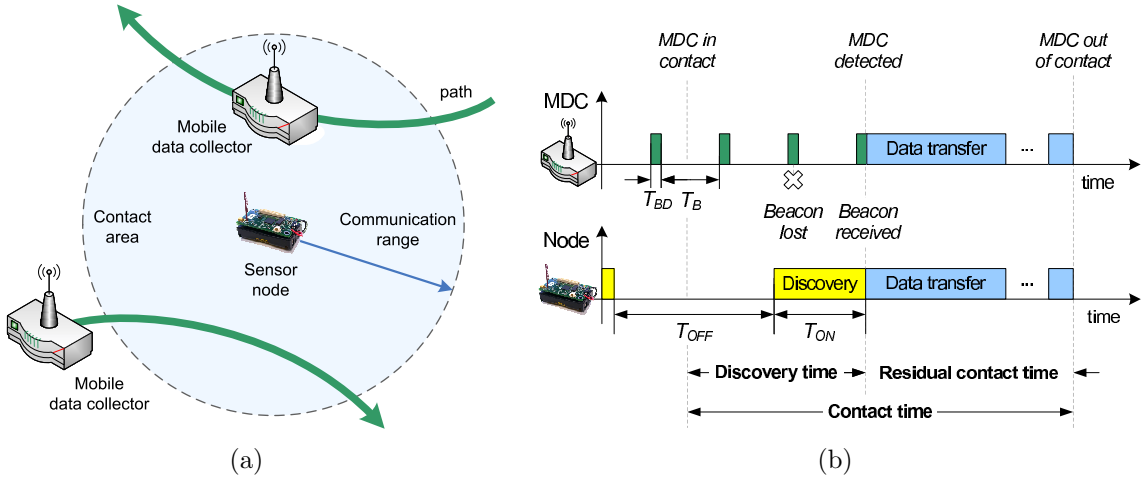


Figure 5.1. Reference scenario (a) and an example of contact (b).

spent by the MDC inside the contact area is called *contact time*. During a contact, messages exchanged between the MDC and a sensor node experience a certain message loss, denoted by $p(t)$. We also assume that the MDC mobility is not controllable. We further define *tour* (and denote it with T) as the smallest time duration after which the mobility pattern repeats [51] and *inter-contact time* as the actual period of time elapsed from the beginning of a contact to the beginning of the subsequent one.

The overall data collection process can be split into three main phases [40]. Figure 5.1(b) shows an example of contact. As MDC arrivals are generally unpredictable, a sensor node enters a discovery phase for the timely detection of the MDC. Successful MDC detection by the sensor node N_i is not immediate, but requires a certain amount of time, called *discovery time*, and denoted as d_T in Figure 5.1(a). Upon detecting the MDC, the node N_i switches from the discovery state to the data transfer state, and starts transmitting data to the MDC. As a result of the discovery process, the node cannot exploit the whole available contact time for data transfer. The portion of the contact time that can be actually used for subsequent data transfer is called *residual contact time* and is referred to as c_T. At the end of the data transfer

phase, the node may switch to the discovery state again in order to detect the next MDC passage. However, if the MDC has a (even partially) predictable mobility, the node can exploit this knowledge to further reduce its energy consumption [40]. In this case, the node can go to sleep until the next expected arrival of the MDC. In any case, the node N_i may be awake also when the MDC is out of reach. The amount of time spent by a node in the discovery state while the MDC has not yet entered the contact area is called *waiting time*.

Similar to [48], we will use an asynchronous discovery protocol and an ARQ-based protocol for data transfer. In detail, the MDC periodically sends special messages called *beacons* to advertise its presence in the surrounding area. The duration of a beacon message is equal to T_{BD}, and subsequent beacons are spaced by a *beacon period*, indicated with T_B. In order to save energy during the discovery phase, the node operates with a duty-cycle δ, whose active time $T_{ON} \geq T_B + T_{BD}$ so that a complete beacon can be received during the active time, provided that node wakes up when the MDC is in the contact area.

The node N_i enters the data transfer phase upon receipt of a beacon from MDC. While in this phase, the node remains always active to exploit the contact as much as possible. On the other hand, the MDC enters the data transfer phase as soon as it receives the first message sent by the sensor node, and stops beacon transmissions. The communication protocol adopted during the data transfer phase is *selective repeat* [66], i.e., a window-based ARQ protocol with selective retransmission, whose window size is assumed to be equal to W messages. Note that the acknowledgement messages in the ARQ scheme are used not only for implementing a retransmission strategy, but also as an indication of the MDC presence in the contact area.

The data transfer phase ends either when the sensor node N_i has no more messages to transmit during a contact, or the MDC is not reachable any more. Node

N_i assumes that the MDC has exited the contact area when it misses N_{ack} consecutive acknowledgments.

5.1.2 Application mobility scenarios

Mobility scenario exhibited by an application depends both on application requirements as well as the choice of MDC. Next we classify various mobility scenarios based on sparse WSN applications.

- *Deterministic*: In this scenario, the MDC arrivals are periodic and the inter-contact time (time between two arrivals) is fixed. A controlled MDC used for data collection will fall in this category. A battle-field surveillance application using controlled airborne or ground vehicles as MDC and an agricultural farm monitoring application using a mobile robot are examples of applications exhibiting deterministic mobility.
- *Gaussian*: Here also MDC arrivals are periodic but the inter-contact time varies and can be considered to follow a normal distribution. This mobility pattern corresponds to the case where the MDC arrivals are rather predictable, but suffer from a certain spread [40]. Example applications include habitat monitoring where a forest patrol makes periodic patrols along certain paths in the forest and battle-field surveillance using a semi-controlled/manual vehicle for data collection.
- *TimeOfDay*: This scenario covers MDC whose arrival patterns are not periodic and depends on time of the day (and/or day of the week). Exact time of arrival may still not be completely deterministic and suffer from a spread like Gaussian mobility. Applications requiring data-collection from single MDC like postal service truck, city tour bus, college/company campus bus etc., can be modeled using this scenario.

- *TimeOfDay-Multiple*: Unlike above mobility scenarios which are constrained to mobility pattern of single MDC, this scenario includes application involving large number of MDCs arrivals depending on time of the day. Applications like city traffic monitoring, national park monitoring etc. can mainly be modeled with this scenario where a number of MDCs arrive with their patterns dependent on time of the day.

From the above classification, it is clear that various applications exhibit distinct mobility patterns and hence any approach to learn these mobility patterns may require unique strategy for efficient discovery and data collection process. We have created a generic framework based on reinforcement-learning that can be utilized for all above applications without requiring distinct algorithms/strategies for each application scenario. Next we describe basic elements involved in our framework.

5.2 Resource Aware Data Accumulation (RADA) strategy

In this section, we define an adaptive strategy based on DIRL for resource-efficient data collection in sparse WSNs. The goal of this strategy is to maximize the number of contact detections and the percentage of data successfully transferred during contacts, while minimizing the energy consumption of sensor nodes. Next we describe the task and state definitions which RADA takes as an input and are the building blocks of an application.

5.2.1 RADA tasks

In the context of the reference scenario already introduced in Section 5.1.1, we have identified three major phases involved in data collection process in sparse WSNs, i.e. discovery, data transfer and sleep. As focus of this work is on the discovery phase, we have defined the following three tasks, each corresponding to a different duty-cyle

used for discovering the MDC. In order to make the derivation of tasks more general, we have defined the actual duty-cycles on the basis of a maximum allowed duty-cycle, denoted as δ_{max}.

- *High Duty-cycle* (HD). The sensor is executing at a high duty-cycle, equal to δ_{max}. Ideally this task should be executed whenever the probability of MDC being in the contact area is high.
- *Low Duty-cycle* (LD). The sensor is executing at a low duty-cycle, equal to $0.5 \cdot \delta_{max}$. Ideally this task should be executed whenever the probability of MDC being in the contact area is low, so that the correspondent energy consumption is very low as well.
- *Very Low Duty-cyle* (VLD). The sensor is executing at a very low duty-cycle, equal to $0.1 \cdot \delta_{max}$. Ideally this task should be executed whenever the probability of MDC being in the contact area is very low, so that the correspondent energy consumption can be considered as almost negligible with respect to the maximum allowed duty-cycle.

Eventhough our task definition only deals with duty-cycle tuning during discovery phase, efficiency of data-transfer phase highly depends on success of discovery phase. For data-transfer to be successful, it is eminent to have MDC discovered as early as possible to maximize the residual contact time. As can be seen from the above-mentioned task definitions, the MDC discovery and successful data transfer process can be maximized while minimizing energy usage if we can adaptively schedule above tasks based on learned probability of MDC being in contact. RADA learns this probability in the form of utilities built by using local rewards. In order to achieve task scheduling and resource management, RADA executes discovery tasks according to the algorithm depicted in Figure 3.3 in Section 3.3.

Each task above may include one or more phases of the data collection process. The number of executions of discovery and sleep phase depends on the current duty cycle. For an example, during high duty cycle (HD) task, number of times discovery phase is executed will be much higher compared to that of VLD task. The data transfer phase is executed only when discovery is successful and an MDC is in contact. In order to manage this, we have introduced a state variable i_c which is true when the MDC is assumed to be in contact with the sensor. In detail, i_c is set to one when the discovery phase ends with success, i.e., a beacon is successfully received by the sensor. i_c is set to zero when the sensor has lost a number N_{ack} of consecutive acknowledgement messages as a result of the data transfer phase, thus assuming that the MDC has exited the contact area. Hence, the data transfer phase can be entered only after the MDC has been detected (i.e., $i_c = 1$), under the constraint that messages in transmission[1] are enough to fill a complete window.

For all tasks scheduled by node, the reward is defined as $r_t = (n_c \cdot e_p - 1) \cdot e_s$, where n_c is the number of contacts encountered while executing that task, e_p is the expected price of task for each contact, and e_s the energy spent. Note that the expected price is chosen as a multiple of the energy spent for that task, so as to allow a symmetric evaluation of the reward function. Thus, for each task, the reward is equal to the expected price e_p minus the energy spent e_s if one MDC is successfully detected. If no MDC is detected, reward is negative (equal to minus e_s).

5.2.2 RADA state definition

The state definition is one of the most important characteristics for application of reinforcement learning algorithms and provides a context to learned utilities.

[1]Messages in transmission can be either buffered messages or messages which have been already transmitted but not yet acknowledged.

Efficient application of reinforcement learning to any problem depends heavily on how well the state definition captures the context of what needs to be learnt. As such, different mobility scenarios as described in Section 5.1.2 may require different state definition as their requirement of learning context is different. For example, deterministic and Gaussian mobility scenarios require learning of inter-contact time between MDC arrivals in order to predict next arrival of MDC, while time-of-day scenario need to learn MDC's arrival pattern in terms of hour of the day and is not dependent on actual inter-contact time. In order to support all mobility scenarios, we have created a generic state definition involving context parameters for all cases. This is possible because of usage of weighted hamming distance in RADA for state distinction. Hence, weights of state parameters can be set at runtime to allow creation of appropriate learning context required for current application scenario.

In order to learn MDC's arrival pattern, it is necessary to introduce a temporal characterization in the state representation of the sensor nodes. On the basis of the concept of tour, we split the time (as perceived by a sensor) into a number of intervals called *time domains*, whose duration is denoted as T_d. More specifically, each task is scheduled for one time domain, at the end of which utilities are updated and the sensor node evaluates the new state. Following are the state parameters used in our generic data collection framework.

- *icp*. Inter-contact period as observed and recorded by the sensor node.
- i_r. A boolean denoting whether MDC's arrival is in time bounded range or not and is set as below:

  i_r= true, if $(icp - t_lc) < T_d$ and false otherwise,

  where t_lc denotes time elapsed since last MDC contact.
- *tod*. Time-of-day value of this state (based on time at which node observed this state).

More state variables can be added if additional learning context is required. For example if day of week is also important along with time of the day, one more state variable representing day of the week can be added to the above set. Note here that with addition of each variable, state space (based on their weight) and the storage and computational requirements at sensor node will grow. Hence it is necessary to evaluate the effectiveness of additional variables in terms of overall performance before adding them. By default time-of-day variable has zero weight meaning that it will not have any contribution to the learning context (for deterministic and Gaussian mobility scenarios). Support for different mobility scenarios based on setting of state parameter weights allows tackling of change in application mobility scenario at runtime by merely tuning the weights. Further, setting/tuning of weights can be faciliated by MDC based on MDC's mobility configurations.

As all statistics regarding the mobility pattern of the MDC are estimated by sensor nodes, we added some filtering techniques to avoid misinterpretation of context. For instance, the sensor might perceive a single actual MDC contact as multiple observed contacts. To this end, we implemented a simple timeout technique, so that the sensor considers successful reception of a beacon message as a new contact only when a certain time has elapsed since the preceding contact detection. Similarly, non-detection of actual contact would result in incorrect learning of the MDC mobility pattern. For this reason, we maintained a short history (size 5) of contacts (in terms of the inter contact time), and used minimum inter-contact time across this history for state evaluation.

5.2.3 Exploration/Exploitation

DIRL uses the classic exploration and exploitation strategy used in most approaches based on reinforcement learning to obtain the utilities of the individual

tasks. In this chapter, the original DIRL exploration 3.3 policy has been improvised by considering a mobility-aware exploration probability based on number of contacts. More specifically, exploration factor ϵ is given by

$$\epsilon = \epsilon_{min} + \max\left(0, k \cdot (c_{max} - c)/c_{max}\right)$$

where ϵ_{max} and ϵ_{min} define upper and lower boundaries for the exploration factor, respectively; c_{max} represents the maximum number of contacts (as obtained from the application) after which a steady state condition is likely to be reached, while c represents current number of detected contacts; finally, k is a constant that can be tuned to control the descending rate to the minimum exploration probability. Therefore, the heuristic presented above allows initial exploration with a higher rate and gradually decreases over time as DIRL is able to detect up to c_{max} contacts. Note that some minimum exploration is always required, so as to allow a sensor node to dynamically reconfigure in case of environmental changes.

5.3 Simulation setup

To evaluate the performance of our RADA framework we developed a discrete event simulator written in Java. In our analysis we considered the following performance metrics.

- *Discovery ratio* is defined as the average of the ratio between the number of contacts correctly detected by the sensor and the total number of contacts. It is a measure of the discovery efficiency of considered scheme.
- *Residual Contact Ratio* is the ratio of residual contact time to total contact time. This metric captures the effect of discovery phase on data-transfer phase. If MDC can be detected early enough, the residual contact time will be high

and hence increases chances of successful data-transfer. Thus, residual contact ratio is a measure of efficiency of data transfer phase.

- *Activity ratio* is the ratio between the active time and the total time spent during discovery[2]. A low activity ratio indicates low energy consumption as node tries to save energy by using a low duty cycle.
- *Energy efficiency* is energy spent by the sensor per each correctly detected MDC contact. It is computed as ratio of total energy consumed and number of MDC contacts detected. Energy consumption here includes energy spent in discovery as well as data transfer phase.

The former two metrics relate to performance in terms of energy whereas the latter two relate to cost. Thus, an optimal scheme will have highest discovery ratio and residual contact time while minimizing activity ratio and energy utlization per detected MDC contact. As for the energy expenditure, we implemented a simple model that characterizes the radio, while we do not address the energy expenditure of the CPU, since it is almost negligible. Specifically, the energy expenditure of the radio is calculated as $P_{state} \cdot T_{state}$, where P_{state} and T_{state} denote respectively, the power consumption of the radio and the amount of time spent in a given state, i.e., receive, transmit and sleep. We assume that the energy consumption of the radio during idle periods, i.e., when it is monitoring the channel, is the same as in the receive state. As for message loss, we used the model considered in [48, 50] and based on experimental data measured in a real testbed in the same scenario [67].

In order to compare the performance of RADA with other approaches, we also considered the following schemes.

[2]In this metric we do not consider the activity due to data transfer. Hence, the activity ratio is a measure of the average duty-cycle which derives from the executions of the different discovery tasks.

- *FixedHD*. Sensor node executes High Duty-cycle (HD) task at all time steps. This scheme gives an upper bound on performance that other learning based schemes can achieve but at the cost of higher energy consumption.
- *FixedOD*. Sensor node executes a duty-cycle equal to average optimal duty-cycle determined by RADA at all time steps. This allows making a fair comparison between RADA and fixed duty-cycle approach. Please note here that the average duty cycle is determined by RADA dynamically, based on the scenario and, in practice it is not possible to derive it in advance and this approach is considered only for comparison.
- *Simple RL*. Sensor nodes use the adaptive node discovery algorithm proposed in [52].
- *Oracle*. Sensor nodes have perfect knowledge on MDC contacts, so they do not perform discovery at all. They start transmitting data as soon as the MDC is in the contact area and stop transmitting when there are no more data or the MDC is out of contact.

As for the mobility pattern of the MDC, we considered all the scenarios described in 5.1.2 for our analysis. The first three mobility scenarios i.e. Deterministic, Gaussian and TimeofDay applies to applications using single MDC while the last scenario (TimeofDay-Multiple) is for applications involving large number of MDCs. State parameter weigths are set as per the requirement of application scenarios and are depicted in Table 5.1.

To derive confidence intervals we used the replication method with a 95% confidence level. In all experiments we performed 10 replicas, each consisting of at least 1000 MDC passages. In the following, we assume a MICA2 series mote [68] as the static sensor node, and use the related parameters for power consumption. We will assume that the radio is operating at a link speed of 19.6 kbps bitrate. All other simu-

Table 5.1. State parameter weights used for simulation

Scenario(s)	**Parameter**	**Weight**
Default values	Inter-contact period (*icp*)	0.005
	MDC In Range (i_r)	1.0
	Time-of-day (*tod*)	0.0
Deterministic/Gaussian	Inter-contact period (*icp*)	0.005
	MDC In Range (i_r)	1.0
	Time-of-day (*tod*)	0.0
Time-of-day	Inter-contact period (*icp*)	0.0
	MDC In Range (i_r)	1.0
	Time-of-day (*tod*)	1.0
Multiple MDCs Time-of-day	Inter-contact period (*icp*)	0.0
	MDC In Range (i_r)	0.0
	Time-of-day (*tod*)	1.0

lation parameters, chosen according to the methodology used in [48], are summarized in Table 5.2.

5.4 Simulation results

The performance evaluation of our RADA framework is divided into multiple parts. For the sake of clarity, in the following we will consider a single MDC (except in case of Multiple MDCs Time-of-day scenario where multiple MDCs are considered) which collects data from a single sensor node.

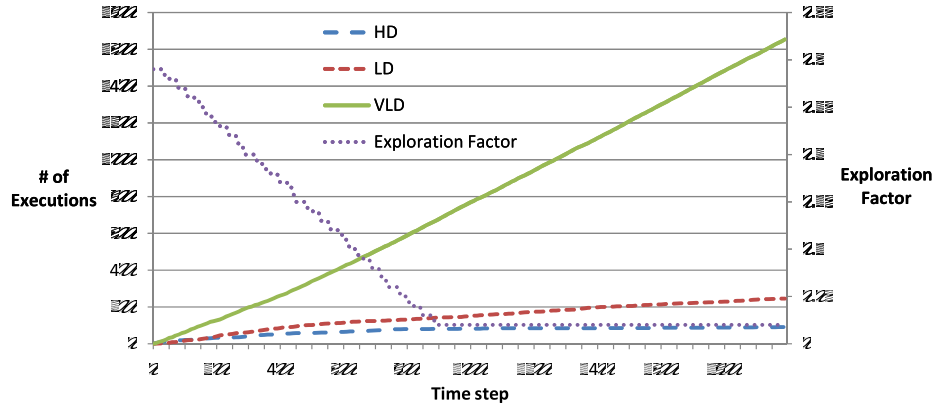
5.4.1 Analysis of reinforcement learning

In this section, we evaluate adaptive task scheduling capabilities of our framework depicting the effectiveness of reinforcement learning in RADA framework. Figures 5.2(a) and 5.2(b) show number of executions for different tasks for MDC speeds of 3.6 km/h and 40 km/h respectively over time-steps during initial phase. Overall simulation was carried on for 18000 time units and was using deterministic mobility

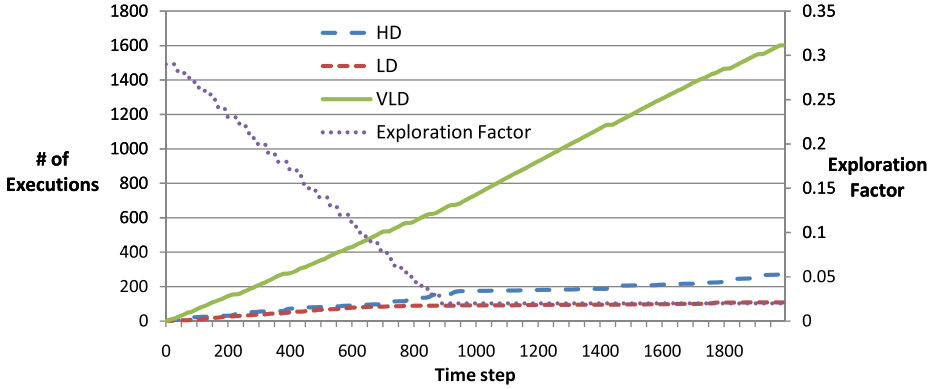
Table 5.2. Parameters used for simulation

Parameter	**Value**
Maximum Duty-cycle (δ_{max})	3.0
Minimum exploration (ϵ_{min})	0.02
Maximum exploration (ϵ_{max})	0.3
Descending rate (k)	0.2
Maximum contacts (c_{max})	10
Initial time domain duration (T_d)	100 s
Message generation interval	10 s
Expected price (e_p) multiplier	10
Beacon period (T_B)	100 ms
Spread over mean for Gaussian mobility	30 s
Beacon duration (T_{BD})	10 ms
Window size (W)	16
Consecutive lost acks (N_{ack})	5
Message payload size	24 bytes
Frame size	36 bytes
Radio transmit power (0 dBm)	49.5 mW
Radio receive/idle power	28.8 mW
Radio sleep power	0.6 μW

scenario. Figures also show variation in exploration factor over time in this phase. Number of executions of VLD (with minimal duty-cycle) is higher almost immediately after startup compared to other tasks. This is because VLD consumes least energy and hence will receive maximum reward in absense of MDC. The slope of VLD task increases further with time as the exploration factor decreases, while the slopes of HD and LD tasks decrease with exploration. When exploration factor is high, HD and LD tasks will still get scheduled eventhough their utility in any given state is less and hence those tasks also have higher initial slope. Number of executions for HD and LD tasks are significantly less compared to that of VLD task. System reaches a steady state at around 900 time steps where exploration factor approaches minimum



(a)

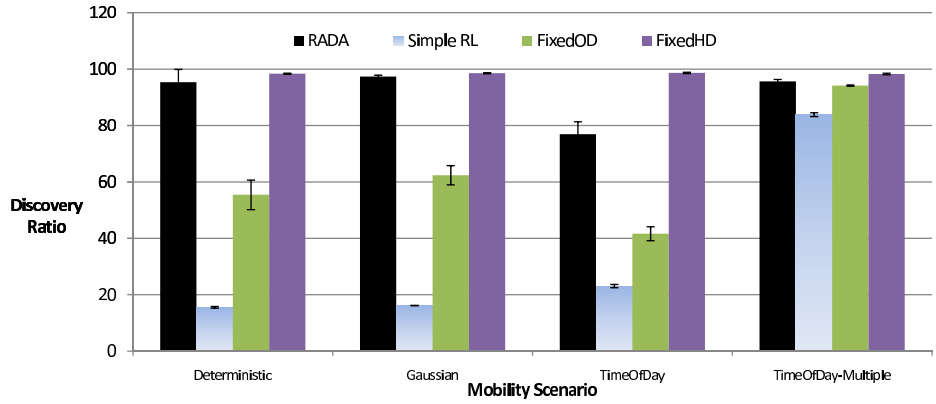


(b)

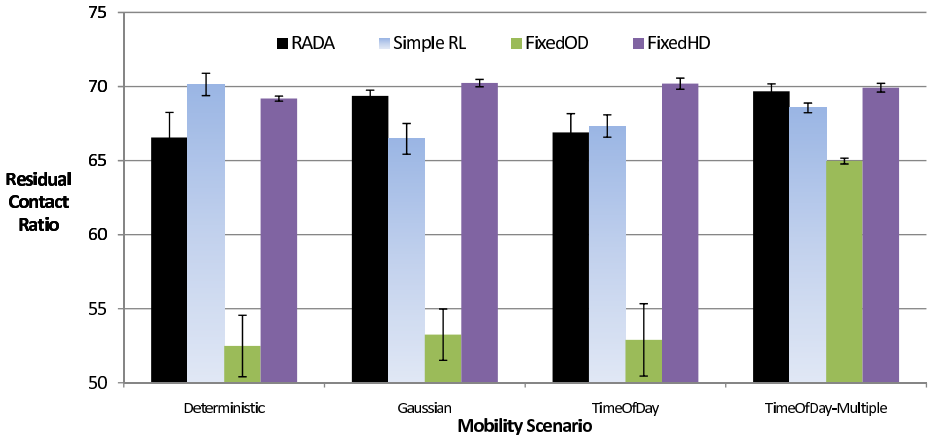
Figure 5.2. Number of task executions and exploration factor over time for MDC speed of 3.5 km/h(a) and 40 km/h(b).

value after which the system mainly uses its learned utilities to choose task at each time-step.

Reinforcement learning allows node to determine when it is most appropriate to execute a HD or LD task and when it should preserve energy by executing VLD based on learned utilities in different node states. This learning is also apparent if we compare plots for MDC speeds of 3.5 km/h and 40 km/h. At a slow speed of 3.5 km/h, a node can discover MDC even with lower duty-cycle (LD task) and it is not necessary for node to spend additional energy on HD. On the other hand, at a high speed of 40 km/h, MDC's discovery requires HD as use of LD may result



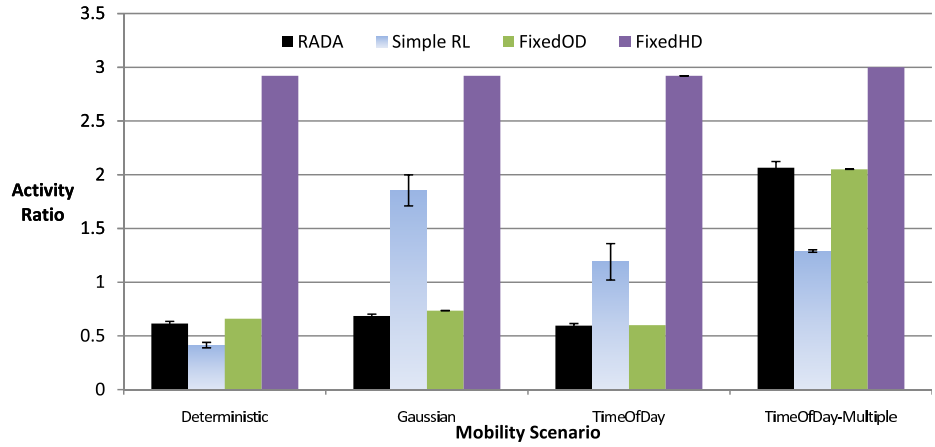
(a)



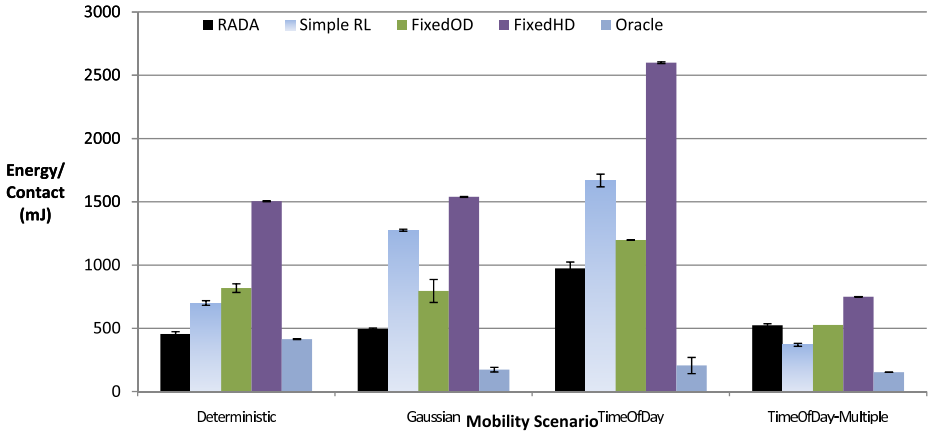
(b)

Figure 5.3. Comparison of schemes under different mobility scenarios: (a) Discovery ratio and (b) Residual contact ratio.

in large number of missed contacts. RADA framework enables a node to learn the appropriate task in both cases by using reinforcement learning. As can be seen from 5.2(a) and 5.2(b), node uses HD at higher speed and LD at lower speed to preserve energy. Thus, reinforcement learning enables node to optimize task scheduling based on current operating environment.



(a)



(b)

Figure 5.4. Comparison of schemes under different mobility scenarios: (a) Activity ratio and (b) Energy consumption per contact.

5.4.2 Analysis with varying application mobility scenarios

In this section we evaluate the performance of the RADA framework for all application mobility scenarios described in 5.1.2. An inter-contact of 1800 s is used unless specified otherwise. All other parameters are as specified in Table 5.2.

Performance of RADA in terms of activity ratio, discovery ratio, energy efficiency and residual contact, is compared with those of other schemes described in Section 5.3. The study considers different mobility scenarios of the MDC at a fixed speed 20 Km/h.

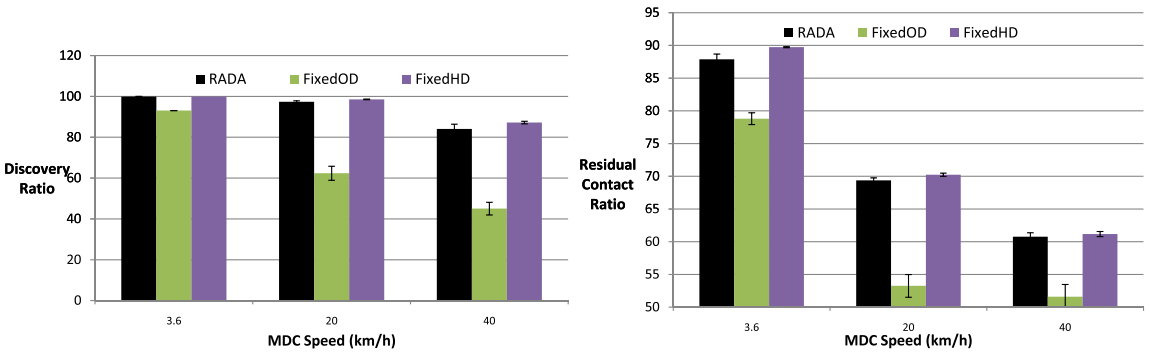
Discovery ratio is an important criteria as activity ratio does not take into account missed contacts when nodes are asleep. To this end we considered the discovery ratio (see Figure 5.3(a)). Activity ratio for different mobility scenarios is given in Figure 5.4(a). RADA consistently shows lowest activity ratio and is able to conserve maximum energy compared to other schemes under mobility scenarios considered. This is because RADA is able to learn and adapt to MDC's mobility pattern for all scenarios and thereby tuning its duty-cycle efficiently. RADA enables a node to operate at lower duty-cycle without compromising discovery efficiency. As can be seen in Figure 5.3(a) dicovery ratio of RADA is very close to that of FixedHD which operates on average at three times higher duty-cycle than RADA. All other schemes either use a high activity ratio or do not provide appropriate discovery efficiency. Even though FixedOD operates on same duty-cycle as RADA, RADA's discovery ratio is almost twice that of FixedOD for most scenarios. As expected, performance of the scheme proposed by Simple RL [52] is closer to that of RADA only in case of TimeOfDay-Multiple mobility scenario. This is because the scheme proposed by Dyo et al., addresses applications with TimeOfDay-Mulitple mobility scenario and cannot be applied efficiently to other application scenarios. RADA on the other hand is suited to a wide range of applications due to its flexbile state based learning approach. Activity and discovery ratio for Oracle are not shown as Oracle doesn't perform discovery and hence its activity ratio will always be zero while discovery ratio will be 100%.

Figure 5.3(b) shows residual contact time which is a measure of efficiency of message transfer phase. Residual contact time of RADA on average is in the same vicinity as FixedHD (slightly lower in some cases), but is considerably higher than FixedOD. This shows that RADA helps node to operate at lower duty-cycle without compromising message transfer efficiency.

Energy efficiency is the most important metric that characterizes the joint effect of activity and discovery ratios. We measure energy efficiency in terms of energy consumption per successful MDC contact detection. Thus the scheme performing at lower duty-cycle while maximizing MDC discovery will have lowest energy consumption per contact or highest energy efficiency. Energy also includes energy spent during data-transfer phase. Observing Figure 5.4(b), it can be concluded that RADA outperforms all other schemes for all mobility scenarios with an exception of TimeOfDay-Multiple scenario. For Deterministic and Gaussian scenarios, RADA's performance is similar to Oracle which is the ideal scheme not performing any discovery (hence mainly represents energy spent for message transfer during contact). This observation proves that RADA substantially reduces node's energy consumption in discovery phase for these scenarios. For TimeOfDay scenario also RADA performs better compared to other schemes. The scheme proposed by Dyo et al. performs better in terms of energy efficiency for the TimeOfDay-Multiple scenario. But the difference in energy consumption for different schemes is not significant in TimeOfDay-Multiple scenario. This is because the number of contacts in TimeOfDay-Multiple scenario is very high (due to presence of multiple MDCs) and hence energy consumption in message transfer dominates over energy consumption in discovery process. Hence, there is not much distinction between different schemes as message transfer phase is same for all of them.

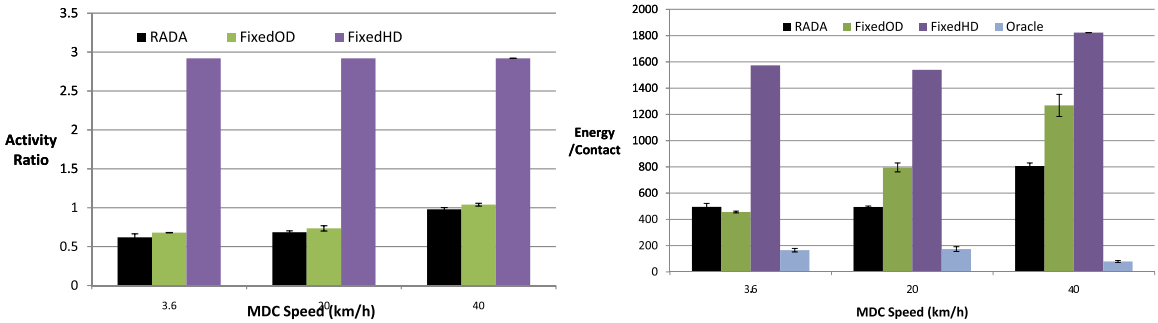
5.4.3 Performance under varying speeds

The effect of different MDC speeds on peformance of our adaptive framework is studied next. Performance of all discovery schemes are compared in terms of activity ratio, discovery ratio and energy efficiency for three MDC speed, 3.5, 20 and 40



(a) (b)

Figure 5.5. Performance of schemes with varying MDC speeds: (a)Discovery ratio and (b)Residual contact ratio.



(a) (b)

Figure 5.6. Performance of schemes with varying MDC speeds:(a)Activity ratio and (b)Energy consumption per contact.

km/h. For this analysis, Gaussian (with 30 s spread over mean) mobility scenario with varying speeds is used.

Figure 5.5(a) provides discovery ratio for different MDC speeds. As expected, when mobility is low, almost all contacts are detected, independent of the discovery scheme [3]. The situation is different, however, when the speed is high (i.e., 20 or 40 km/h). In this case the two schemes RADA and FixedHD clearly get better results than other approaches. RADA provides discovery efficiency equivalent to

[3]The scheme proposed by Dyo et el.[52] is not considered in this discussion as it works only with TimeOfDay-Multiple scenario

FixedHD while operating at no more than one-third of VHD duty-cycle. FixedOD's performance degrades significantly with increase in speed compared to RADA even though both executes same duty cycle. As expected, residual contact ratio (and thus contact period available for data transfer) decreases significantly with increase in MDC speed for all schemes. Here also RADA is closest to FixedHD than any other scheme and hence provides better opportunity for data transfer.

At lower speeds, the MDC is expected to be detected even at lower duty-cycles due to high contact period. Hence node should be able to conserve energy by operating at lower duty-cycle when MDC speed is less. Figure 5.6(a) shows RADA executing this behavior very well. At 3.5 km/h speed, it uses little over 0.5% of duty-cycle but then adapts when MDC speed increases to higher duty cycle of 1.0% at a speed of 40 km/h. This illustrates the effectiveness of RADA in adapting to its environment to optimize energy consumption. FixedHD operates at fixed duty-cycle and hence there is no change in duty-cycle with speed.

The results for energy efficiency (in terms of energy consumption per MDC contact) are provided in Figure 5.6(b) (results for Oracle scheme are also shown as a reference). As a general trend, the energy expenditure per contact increases with MDC speed. However, for Oracle energy consumption decreases with increase in MDC speed. This is because Oracle's energy consumption only includes data transfer phase and with increase in MDC speed, contact period will decrease, resulting in smaller data transfer phase. For all other schemes, discovery ratio decreases at higher speed and hence energy utlization per detected contact increases with increase in speed. Figure 5.5(b) shows residual contact ratio with varying speed.

In conclusion, proposed solution using RADA can be effectively used in a wide range of application scenarios, even when the contact time is short and the uncertainty

Table 5.3. Effect of time domain duration on RADA performance

Scenario	**Time domain duration T_d**	**Energy consumption /contact (mJ)**
Deterministic	0.5%icp	1064.52
	5.0%icp	655.75
	25.0%icp	751.44
	Auto-tuning (Initial value=0.5%icp)	647.93
	Auto-tuning(Initial value=25.0%icp)	673.30
Gaussian	0.5%icp	1205.61
	5.0%icp	643.15
	25.0%icp	793.05
	Auto-tuning(Initial value=0.5%icp)	630.06
	Auto-tuning(Initial value=25.0%icp)	666.13

on MDC arrivals is high. Thus RADA results in effective resource allocation while, at the same time, exhibiting very good performance.

5.4.4 Automatic tuning of time domain

As our time-step for reinforcement learning is equal to T_d and as value of i_r is dependent on time domain, the performance of RADA is dependent on appropriate value for time domain duration T_d. To illustrate this fact, we conducted preliminary experiments with different time domain durations as shown in Table 5.3 where time domain duration is set to 0.5%, 5% and 25% of inter-contact period (icp). The performance of RADA is measured in terms of energy consumption per successful MDC detection, and hence lower value denotes higher energy efficiency. We can see from the table that energy consumption is minimal when using time domain duration equivalent of 5% of icp for both deterministic and gaussian mobility. Performance degrades if we try to use high T_d value at 25% and is worse at very low value of 0.5%. RADA utilizes this observation to adaptively tune time-domain duration to around 5% of icp value automatically at runtime. As a result of automatic tuning, framework

becomes somewhat insensitive to initial time domain value set by application and results in optimum results. To prove the insensitivity to initial time domain value, we set initial T_d to worse case values and apply auto-tuning after observing actual *icp* at runtime. The results after automatic tuning are also shown in Table 5.3 where initial time domain duration is set to 0.5% and 25% of *icp* respectively. Now as RADA is tuning time domain duration at runtime automatically based on observed *icp*, its performance is not impacted by the initial value of time-domain duration set by the application and is close to optimal as obtained at 5% of *icp*.

5.5 Conclusion

In this chapter, a novel Resource Aware Data Accumulation (RADA) framework for sparse Wireless Sensor Networks (WSNs) with Mobile Data Collectors (MDCs) is proposed. The problem of energy-efficient MDC discovery has been addressed by exploiting the Distributed Independent Reinforcement Learning (DIRL) framework. Our results show that the RADA framework is highly efficient in terms of low duty cycle, high discovery rate and high energy and data transfer efficiency. Compared to existing solutions, the proposed approach not only performs better, but also can adapt to different operating conditions and mobility patterns characterized by high uncertainty. Automatic tuning of time-domain duration based on MDCs mobility pattern is introduced for optimum performance. The generality of RADA framework allows applicability to wide range of application scenarios. As a result, it can be effectively used in the development of sparse WSN applications.

CHAPTER 6

DESIGN OF DReL MIDDLEWARE

In Chapter 4, we presented a multi-tier reinforcement learning based framework for efficient distributed resource management in WSNs. A middleware framework for WSN also requires an appropriate communication paradigm for task and data dissemination as well as reward distribution. In this chapter, we design and develop a complete middleware solution called DReL (Distributed Reinforcement Learning) that provides an easy-to-use interface to application developers for creating customized applications with specific QoS and optimization requirements.

Additionally, WSN applications are becoming increasingly pervasive, requiring support for multiple heterogeneous applications executing simultaneously on the sensor network infrastructure. Thus, rather than building a WSN infrastructure for each application, a single WSN infrastructure may be utilized by a wide range of applications. In recent years, utility theory has played a significant role in the proliferation of cloud computing allowing ease of development of applications using shared computing infrastructure. Similarly, one can envision use of utility theory to enable rapid development of pervasive applications on top of shared sensing infrastructure. CitySense [8] project is an example of such sensing network infrastructure deployed all over a city to allow development of a variety of WSN applications viz. traffic statistics/monitoring, accident alerting, public safety and crime watch, driver advisory applications, noise and air pollution monitoring etc.

We incorporate concepts from directed diffusion [12, 13] in designing an effective communication mechanisms for our middleware. Directed diffusion is data-centric in

nature and has the ability to perform effective task and data dissemination in WSNs using only localized interactions among neighboring nodes. Nevertheless, there are certain issues in applying directed diffusion for several classes of applications:

- *Scheduling*: Directed diffusion assumes the use of static scheduling of sensor nodes where each sensor node may collect and report data at fixed time intervals or based on some pre-defined schedule. This assumption does not hold good for applications requiring autonomous adaptation, e.g. sensor nodes operate at a very low-duty cycle most of the time, but should adapt to a high-duty cycle on occurrence of events of interest (for example, object tracking). Applying directed diffusion to such applications can be energy inefficient as there is no mechanism for nodes to adapt to prevailing conditions for better resource management.
- *Optimization*:Directed diffusion allows optimization of data-dissemination based on a single metric, i.e. lowest delay path and doesnt incorporate any application specific goals to optimize one or more parameters - longest network lifetime, lowest energy consumption, lowest bandwidth utilization, highest data quality etc. For the development of a generic framework applicable to development of varieties of WSN applications, it is necessary to support a range of application optimization goals.
- *Reinforcement*: Directed diffusion uses reinforcement to determine favorable path from source to sink for data-dissemination, but neither a concrete mechanism nor rules are provided for reinforcement, thereby increasing the burden on the application designer. Therefore, there is a need for a generalized reinforcement learning framework to allow development of simplified applications without indulging into reinforcement mechanism/rules.

The goals of the work are two-fold: to incorporate effective and proven task (interest) and data dissemination techniques of directed diffusion in a resource management framework; and to address above issues by improving efficiency and autonomous adaptation of directed diffusion in sensor selection and scheduling.

6.1 Design Principles

The DReL middleware solution incorporates the following design principles:

1. *Bottom-up approach*: Each sensor node manages its own resources and schedules tasks by learning tasks' utilities over time using mostly local information.
2. *Global optimization*: Overall system is able to achieve global optimization goal or at-the-least does not suffer from Tragedy of Commons (TOC) or similar phenomena, even-though each node self-schedules the task using local information.
3. *Continuous adaptation*: Middleware is able to tackle uncertainty and dynamic nature of WSN by continuous autonomous adaptation. Middleware manages change in application's requirement over time to allow application to adapt to its global state change.
4. *Localized Interactions*: All interactions are confined to neighboring nodes.
5. *Data-centric*: Middleware uses data-centric task and data dissemination using publish-subscribe mechanism. Middleware supports application specific in-stream data processing and filtering.
6. *Generic*: Middleware can support sensor nodes with heterogeneous capabilities and a wide range of WSN applications, e.g. target tracking, environmental monitoring, surveillance, health-monitoring etc.

Directed diffusion helps achieve items 4 and 5 above while the two-tier learning approach discussed in Section 4.2 facilitates realization of items 1, 2, and 3. The goal is to incorporate all of the above principles.

Task Description (Interest Packet)
/* Fixed values only set by sink*/ *taskid* /* Unique task Id */ *attributes* /* Set of <key,type,operator,value> tuples one for each variable sink is interested in */ *timestamp* /* task publication time */ *qosConstraints* /* Set of <key,type,operator,value> tuples representing QoS requirements */ *costParameters* /* Set of optimization parameters with their weight factor */ *refreshInterval* /*How often this interest will be refreshed */
/* Variables that can be updated by any node */ *payment* /* Expected payment/reward for matching data packet (used for setting gradients)*/

Figure 6.1. Task Packet.

6.2 Design Architecture

We utilize data-centric and localized communication paradigm of directed diffusion [12, 13] for implementation of our utility based task, data and reward distribution mechanisms. In directed diffusion, each node uses data-rate as a means for setting up gradients for reverse data path. In DReL, a concept of payment (or utility) is introduced to create gradients as well as guide WSN system towards an application's global optimization goal. An application can inject new task on-demand based on its state and requirements on appropriate sink nodes. Sink initiates the task diffusion process and broadcasts a task description packet to neighboring nodes as in directed diffusion. Payment value set on task description packet can be in any arbitrary unit.

Data Packet
/* Fixed values only set by source node*/ *taskid* /* Task to which data belongs */ *timestamp* /*event generation time */ *attributes* /*Set of <key,type,operator,value> tuples that contains actual data */
/* Variables that can be updated by any node */ *sourceId* /* Id of sending node */ *streamId* /* id of the data stream this packet belongs to */ *cost* /* running sum of cost associated with this data */ *reward* /* reward expected by sending node */

Figure 6.2. Data Packet.

However, use of currency for payment that is related (directly or indirectly) to world currency, can be useful in determining valuation of collected sensor data as well as resources available in the sensor network [69]. For a task to be successfully deployed, initial payment value set by the sink needs to be high enough to cover the cost of all nodes in the possible data path.

We adopt attribute-based naming scheme of directed diffusion for task as well as data descriptions. Attribute-based scheme is very flexible and allows usage of simple but powerful publish-subscribe mechanism and many-to-many communications (i.e. multiple sources and sinks) that are very important to WSN applications. Attributes help associating messages with source, sink and filters (in-network processing elements) via matching. Please refer [70] for more details on attribute matching sup-

port in directed diffusion. Like diffusion, task description (interest) consists of a set of attributes that sink is interested in along with context in the form of (key, type, operator, value) tuples (see Figure 6.1). Operator allows specification of simple constraints on data, e.g. (less than, greater than, equality, inequality etc). These operators can allow sensor selection by matching sources to published interests. Task description can also contain data rate and task validity period if applicable.

Application publishes *task* packet (see Figure 6.1) requesting data of interest along with *payment* it is willing to make for that data. Here *taskid* is a fixed attribute like all other attributes of task description, while *payment* is a variable attribute that changes as the interest packet diffuses from one node to another in the network. Task's payment attribute is used by all nodes while setting up gradients (instead of data rate as used by directed diffusion). This is described in greater detail in later sections. Along with set of data attributes, task packet also consists of a set of constraints that represent an application's QoS metrics. For example, these attributes may dictate such application constraints as 'latency less than x' or 'coverage higher than c' on data dissemination or resource allocation. Unlike interest attributes that mainly map sources to sinks, these constraints are evaluated by intermediate nodes in the data-path using in-stream filters. Finally, task packet also consists of a set of cost optimization parameters with their weight factors used by nodes for determining the cost of participation in the task. These include parameters that WSN system supports e.g. energy, network lifetime, number of hops etc. Weight factor allows application to give relative importance to each parameter while evaluating cost of the task. The sum of all weight factors needs to be equal to 1.

The cost of participation for each node is determined using costParameters specified by task description. For example, a task contains costParameters, (lifetime,0.5) and (numberofHops,0.5), i.e. optimize lifetime and number of hops with equal weight.

Table 6.1. Cost Parameter Description

Parameter	**Description**
lt_i	Available lifetime at node i
lt_{max}	Maximum lifetime supported
$cost_{lt}$	Cost coefficient for Lifetime factor
$weight_{lt}$	Weight for lifetime in task packet
E_i	Available energy at node i
λ	Trigger event inter-arrival rate
K_t	Time spent on one sensed event
K_e	Energy spent on processing of one sensed event
P_s	Power spent to monitor the events
$cost_h$	Cost coefficient for NoOfHops factor
$weight_{nh}$	Weight for no. of hops in task packet

Then node 'i' uses the following cost function for evaluating cost of participation by utilizing application specific cost parameters as shown in Equation 6.1 (parameters described in Table 6.1).

$$cost_{iT} = \max(lt_{max} - lt_i, 0) * cost_{lt} * weight_{lt} + cost_h * weight_{nh} \tag{6.1}$$

Lifetime (lt_i) in above equation is computed using lifetime model presented in [71]. For example, for trigger driven applications, lifetime is approximately given as follows:

$$lt_i = \frac{E_i(1 + \lambda K_t)}{P_s + \lambda K_e} \tag{6.2}$$

Coefficients of different cost parameters in above cost function is determined by using a high-low method. If we fix $lt_{max} - lt_i * cost_{lt} \in [0, 1]$ and $cost_h \in [0, 1]$, as sum of weights of all cost parameters is 1, the total value of $cost_{iT}$ will be in range $[0, 1]$. Using high and low values of 1 and 0 respectively for each term, gives $cost_{lt} = 1/lt_{max}$ while $cost_h = 1$. Note here that lt_{max} can be an approximate value and if a node has

lifetime higher than this value, cost contribution by lifetime parameter will be 0. Each node after receiving a task packet, computes node's expected payment based on application's QoS and cost parameters and deducts that value from task's payment attribute before publishing task again to its neighboring nodes. Thus at each node in the system, payment of a task description packet specifies the amount of payment expected by that node if the sending neighbor is chosen for data dissemination. Node's expected payment for each task (for each neighbor) is stored in a gradient table which is further used by the reinforcement learners for management of tasks and resources as well as acts as gradients for data path in the reverse direction.

Data packet (see Figure 6.2) travels in reverse direction from source node to sink. Along with required data attributes, data packet includes a)*cost*- representing cumulative cost of producing this data packet (actual or estimated); b)*reward*- expected reward at the sending node based on set gradients; and c)*streamId*- identifier of data stream (e.g. source id). These three fields are used by a two-tier reinforcement learning scheme to guide WSN system towards the application's global optimization goal. Figure 6.3 shows basic components of DReL as present in individual sensor nodes and their roles and interactions. Macro-learner layer (routing layer) is responsible for managing macroscopic view and actions of a node by performing task, data and reward distribution while micro-learner is mainly responsible for scheduling node's tasks and other local actions and thereby managing node's local resources. A macro-learner component of a node works in conjunction with macro-learner of other sensor nodes that collectively affect each other (e.g. nodes involved in localized interactions). Each micro-learner uses Q-learning as their independent reinforcement learning algorithm to increase node's individual utilities, while on the other hand, macro-learners use Collective Intelligence (COIN) theory to steer the system towards application's

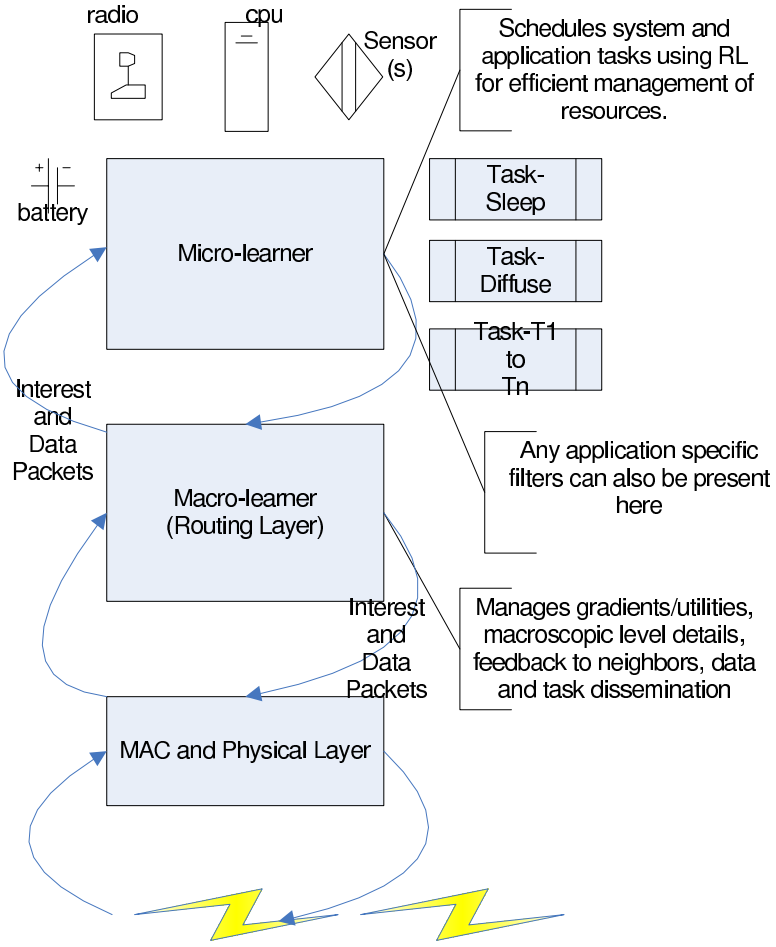


Figure 6.3. DreL components in a sensor node.

global goal. Next we describe DReL in terms of these two components i.e. micro-learner and macro-learner.

6.3 Task Scheduling and Management by Micro-learner

In DReL, each node is responsible for self-scheduling tasks based on current state and utilities using reinforcement learning (RL). This scheduling is performed by micro-learner and uses Q-learning based RL algorithm as described in Chapter 3. Each node has zero or more application specific active tasks and also has two implicit system tasks viz. *Sleep* and *Diffuse*. As the name suggests, *Sleep* task puts

node's radio as well as CPU in SLEEP mode for energy savings while *Diffuse* task is responsible for carrying out task as well as data dissemination process. While acting as a source (executing application's task), a node is capable of carrying on diffusion process as well. Thus in order for successful data and task dissemination process, intermediate nodes (at-least nodes making a complete path from source to sink) should not be executing *Sleep* task. The rest of the nodes in the system should be mainly executing *Sleep* to conserve energy. This requires proper co-ordination among all nodes in the system so that, nodes along the path execute appropriate tasks while allowing energy conservation in the rest of the system. DReL achieves this co-ordination using our two-tier reinforcement learning approach.

Micro-learner of a node inspects task packet to decide whether it is a possible source for the given task or not. A node can be a source if it provides all attributes requested in the task description while matching application's QoS constraints. If node can act as a source, micro learner adds this task to its list of active tasks and makes it available for execution. If this task already exists in node's list of active tasks, micro-learner updates task's source payment value. The payment amount that a micro-learner of node 'i' receives on execution of task T is equal to maximum gradient for task T among all neighbors N, i.e.

$$sourcePayment_{iT} = \arg\max_{n \in N} gradient_{inT} \tag{6.3}$$

After task execution at each time-step, micro-learner computes cost ($cost_{iT}$) based on cost parameters of task T for application specific task (using Equation 6.1) or built-in

cost function for implicit tasks (*Sleep* and *Diffuse*). Micro-learner further computes reward ($reward_{iT}$) from cost and source payment as below:

$$reward_{iT} = (success_T * sourcePayment_{iT}) - cost_{iT} \tag{6.4}$$

From above equation, the reward is positive only if task is successful and otherwise it will be negative (assuming task execution was associated with non-zero cost). An application's source task is considered to be successful if it's able to produce task's attributes with correct match. Similarly, *Diffuse* task is considered to be successful if it receives either data or interest packet during execution. Micro-learner then uses the computed reward values to update its Q-learning value function as well as to set reward ($reward_{iT}$) on the data-packets collected for task T (either generated as a source or received from neighbors). Micro-learner also updates a running sum of cost ($cost_T$) with its own cost ($cost_{iT}$). Each modified packet is then forwarded to macro-learner to carry out the data-dissemination phase. State of micro-learner mainly constitutes the following:

- *Success in recent Diffuse task*: Whether or not there was any successful *Diffuse* task scheduled in the last N time-steps.
- *Success in recent Source task T*: Whether or not there was any successful *Source* task scheduled in the last N time-steps. Here, *Source* task represents any application task T for which this node can act as a source. Note that there will be one such state variable for each deployed task.

This state definition provides micro-learner a way to determine whether it is currently dominant as a source or it is merely participating in a diffusion process. The state definition also allows micro-learner to store all learned utilities for each distinguishable state and quickly adapt to its state change.

1. *task*= Receive task packet from a neighbor n
2. if *task* doesn't exist in task cache then
   2.1 *task_entry*=create a new entry for task
3. if gradient entry doesn't exist in *task_entry* then
   3.1 *gradient_entry*= create a new gradient entry
4. else
   4.1 update *gradient_entry* with the new gradient value /*if different*/
5. if *gradient_entry* is modified and this gradient value is maximum in *task_entry* then
   5.1 Compute cost of participation $cost_{iT}$ /*based on equation (6.1) */
   5.2 Update the *payment* in *task* packet /* with value calculated using equation (6.5) */
   5.3 Set $payable_{iT}$=*payment* for *task_entry* /*this represents the amount node i is expected to pay out to neighbor node */
   5.4 Broadcast *task* packet to all neighbors

Figure 6.4. Psuedo-code for gradient setup phase.

Micro-learner uses either ϵ_{min} or ϵ_{max} as exploration factor based on the following condition: if successful in recent *Diffuse* or *Source* task and previous task is not *Sleep*, use ϵ_{min} else use ϵ_{max}. This exploration policy allows micro-learner to use low exploration rate when it's role is already established and allow to explore more when it is mainly inactive.

6.4 Task and Data Dissemination by Macro-learner

In Chapter 4, we used macro-learning in the context of end-to-end data-stream, i.e. among all nodes collected in a data-path from source to sink. In this context, sink node evaluates each received data-stream and provides reward using Wonderful-life utility function and COIN theory. This scheme may not scale well with increase in network size as each data packet needs to track all participating nodes and reinforcement packet needs to travel to each of them in reverse direction. In DReL, with a goal

to keep all interactions local, macro-learning is used in the context of a local neighborhood only, i.e. each node acts as a sink for inbound data-streams and provides feedback to those neighbors. Thus only nodes involved in a neighborhood will be participating in macro-learning process. Macro-learning is applied at all neighborhoods in a hierarchical fashion along the data-path including the sink and a novel cost and payment based incentive scheme is employed to combine the effect of all nodes. This ensures global applicability of optimization goal. Also use of incentive based system allows easy adaptation to applications state changes by updating payment values.

Macro-learner component of a sensor node maintains a task (interest) cache as in directed diffusion. Each task entry contains a set of gradient entries, one per each neighbor. But unlike directed diffusion where gradient is determined by data rate, each gradient entry consists of expected payment for data dissemination in that neighbor's direction for given task. This expected payment (gradient) for an intermediate node i towards neighbor n for task T is simply equal to payment attribute of the received task description packet, given by, $gradient_{inT} = payment_{inT}$, where $payment_{inT}$ is the amount of payment specified in task description packet as received by node i from neighbor n. Thus each neighbor's gradient suggests expected payment if data is disseminated in its direction. Before forwarding any interest packet to its neighbors, a node deducts its cost of participation from the task's payment value. The payment value set by node with interest forwarded to its neighbors (represents amount payable by this node to neighbor sources) is:

$$payment_{in'T} = \arg\max_{n \in N} gradient_{inT} - cost_{iT} \tag{6.5}$$

```
1. Receive data packet(s) belonging to task T from micro-learner
2. if reward value on data packet ≤ 0 then
    2.1 if Random(0,1) > data_exp then
        /*data_exp=Data Exploration Factor */
        2.1.1 Ignore and drop the packet
    2.2 else
        2.2.1 continue /*exploration*/
3. if gradients are already established for this task T then
    3.1 max_grad_neighbor= Get neighbor with maximum gradient for task T
    3.2 if max_grad_neighbor has gradient ≤0 then
        3.2.1 Drop packet
    3.3 else
        3.3.1 Send packet to max_grad_neighbor
4. else  /* no gradients setup as in case of static deployment */
    4.1 Broadcast packet to all neighbors.
5. if data packet doesn't exist in data cache then
    5.1 Add packet to data cache with the corresponding task and neighbor (inbound) ids.
```

Figure 6.5. Psuedo-code for data dissemination phase.

Here $cost_{iT}$ is the cost computed for task T using task's cost function given by Equation 6.1. Algorithm executed by a node's macro-learner on receiving a task description packet is given in Figure 6.4.

Macro-learner receives collected data packets of a task destined towards task's sink from the micro-learner and employs algorithm shown in Figure 6.5. Macro-learner also uses some exploration for data dissemination in case when data packet has no reward. The exploration allows macro-learner to adjust to dynamic system changes and to find better alternative paths. With our goal to keep all interactions localized, we use macro-learning in the context of a local neighborhood only, i.e. each node acts as a sink for inbound data-streams and provides feedback to neighbors reporting data. Thus only nodes involved in a neighborhood will be participating in macro-learning process and forms a *subworld*. Macro-learning is applied at all

neighborhoods in a hierarchical fashion along the data-path including the sink using our utility based scheme. This ensures global applicability of optimization goal.

In DReL, Wonderful life (WL) reward of each node part of subworld (i.e. data-stream) w at time-step τ involved in task T is given by

$$g_{iwT}(\xi_T) = R_{iT}(\xi_{iT}) - CL_w(\xi_{iT}) \tag{6.6}$$

Here, $R_{iT}(\xi_{iT})$ is the reward in the context of neighborhood at any node i while ξ_{iT} is the corresponding state at time-step τ and $CL_w(\xi_{iT})$ is the reward $R_{iT}(\xi_{iT})$ after clamping w to null or in our case, removing all data values that have been reported by data-stream w. If W is the set of all data-streams received at node i at time-step τ, the following global reward function is used to determine value of R_{iT} for a task T:

$$R_{iT}(\xi_{iT}) = quality_{iWT} * payable_{iT} - \min_{w \in W} cost_{iwT} \tag{6.7}$$

Here, $quality_{iWT} \in [0, 1]$ is an optional quality factor determined by the macro-learner of node i using QoS constraints of task T and all received data-streams W. Quality factor can be useful if application requires to distinguish different data-streams based on quality of data collected and not just based on satisfaction of QoS constraints. Next term denotes the amount payable by node i to its neighbor sources, while the last term on the right hand side of the above equation gives minimum cost of all data-streams in W. Thus reward R_{iT} is computed based on quality of data and lowest cost present in all streams W.

Macro-learner receives data packets from neighboring nodes through the lower MAC layer over wireless channel (Figure 6.3). At the end of N time-steps, macro-learner calculates the WL reward for each data-stream as given by Equation 6.6.

1. Receive data packet(s) for task *T* from a neighboring node *n* belonging to stream *w*
2. if data packet exists in data cache then
    2.1 Ignore and drop the packet.
3. Insert data packet in data cache for stream *w* and sending neighbor *n*
4. Forward the packet to micro-learner
5. if time elapsed from last reinforcement > N time-steps then
    /* compute and send reinforcement */
    5.1 if gradients are already setup then
        5.1.1 Compute g_{iwT} for task *T*
        5.1.2 Compute reinforcements for stream *w* /* using equation (6.8) */
        5.1.3 Send a reinforcement packet to node *n* containing payments for all active streams *w*
    5.2 Reset state of all data-streams /* clears data cache as well as moving average stats */

Figure 6.6. Psuedo-code for reinforcement phase.

Each node already has some gradients set and hence they do get immediate reward based on just their local state (equal to reward value in data packet). Reinforcement from neighbor is only necessary to make sure its utility is aligned with the system as a whole. Macro-learner of receiving node manages the data streams received from all its neighbors and tracks the cumulative moving average of reward and cost (specified in data packets) for each data-stream in the data cache. Data stream entry also consists of current payable amount which is for that stream. At the end of N time-steps, for each data-stream from which it has received one or more packets during last N time-steps, macro-learner sends a reinforcement packet to the neighbor from

which it received those data-streams. If we denote $\delta_w = (payable_{iT} - reward_{iwT})$, reinforcements are computed as per the following:

$$Positive \quad Reinforcement \quad (with \quad payment = payable_{iT}) \quad if \quad \delta_w > m > 0$$

$$Negative \quad Reinforcement \quad (with \quad payment = 0.0) \quad if \quad \delta_w < -m < 0$$

$$No \quad Reinforcement \quad if \quad m < \delta < m$$

Here, $reward_{iwT}$ is the average of data packet reward while $payable_{iT}$ is the amount payable by node i to its neighbors. Thus, reinforcement packet containing payment is sent to source neighbor only if its payable value is significantly different from the average packet reward. This allows us to further reduce the number of reinforcement packets. Here m is a fractional value of task's $payable_{iT}$ (e.g. 10%) and can be used to adjust the rate of WL reward updates. Though a very high value of m may turn off reinforcements altogether and the system may not be able to cope with dynamics. If it is required that reinforcements are sent periodically (e.g. they are acting as data acknowledgements), m should be set to 0.

A reinforcement packet destined to a neighbor n from node i contains a)*taskId*- task identifier; b)*payment*- task's payment value (similar to task description packet); and c)*streamPayments*- consisting of data-streams reinforcements as discussed above for each data-stream received by i from n. On receiving reinforcement packet, macro-learner processes it in the same manner as receiving subsequent interest packet during gradient setup phase. Macro-learner updates sending neighbor's gradient entry with received payment value. If changed gradient is the maximum value gradient then this will also change value of $payable_{iT}$ which is used to reinforce its neighbors. Data-stream payments are used to update the current payable value associated with each

data-stream. Figure 6.6 summarizes macro-learner's algorithm for providing reinforcement to neighboring nodes.

If a node receives reinforcement for a data-stream for which it acts as a source, it uses the data-stream payment ($streamPayments_i$) to update value of expected price for task T i.e. $sourcePayment_{iT}$ as follows.

$$sourcePayment_{iT} = (1 - \alpha) sourcePayment_{iT} + \alpha * streamPayments_i \qquad (6.8)$$

In above equation, α is the learning-rate parameter. This is similar to micro-learner's expected price update (equation 4.5) given in chapter 4.

6.5 Sequence of Interactions

Figure 6.7 gives a high level overview of sequence of interactions taking place between the application and the sensor nodes including the sink and intermediate nodes. As shown in figure, data is diffused from a source node towards a sink through intermediate nodes at each time-step based on data rate and established gradients at each node. At every N time-step, macro-learner on each intermediate node as well as sink evaluates reward for data-streams and provides reinforcement to its immediate neighbors that are actively reporting data. On receiving reinforcement, node updates its gradient for the sending neighbor. If state of the application changes or if there is a change in application requirements at any time, an updated interest/task packet is re-published and all nodes update their task cache and gradients relative to new task definition.

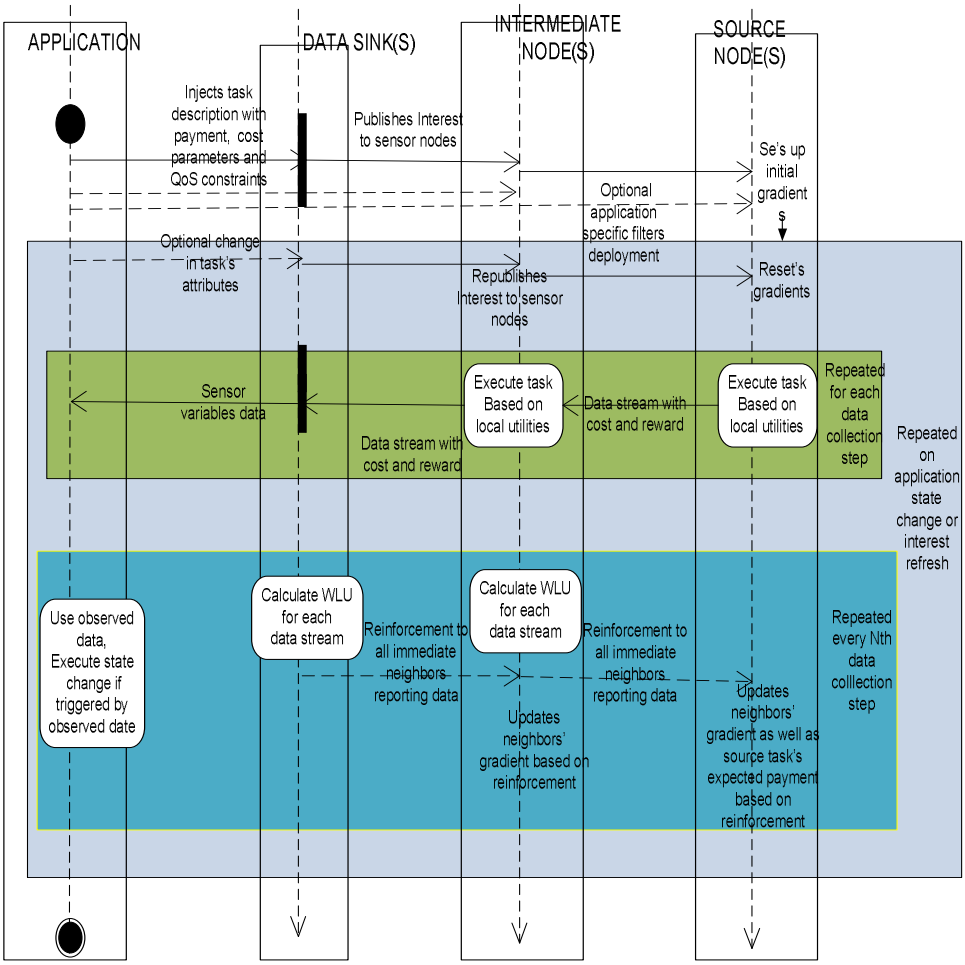


Figure 6.7. Overview of sequence of interactions between application, sink and intermediate and source sensor nodes.

6.6 Discussion

This section describes the incorporation of design principles into the architectural framework. Resource management and task scheduling decisions are taken by micro-learner in each individual node by using only its local information. Macro-learner ensures that individual node increasing its own utility, results in higher global utility and doesn't lead system to Tragedy of Commons or similar undesirable state. Macro-learner achieves this by tuning micro-learner's reward function so that it is always aligned to global utility function. Hence a nodes utility increases only if it

results in an increase in system-wide utility. Reinforcement required for this tuning is implemented using only localized interactions among neighboring nodes. Each node receives reinforcement in the form of payment from immediate neighbor, but the value is still controlled by sink node and global application's goal. Continuous exploration and learning allows system to adapt to uncertainties and dynamic changes in WSN at micro as well as macro levels. Each node is capable of adapting immediately to the local or neighborhood changes and hence resulting in real-time adaptation. Task constraints are verified in-stream near the source and data not matching constraints can be filtered immediately (rather than at sink) saving energy and bandwidth.

Whenever an application's requirement changes (for example with transition of application's state), it's just a matter of publishing changed task description packet (if data or QoS attributes has changed) or changing payment value for a given task. Proposed framework can also support and take advantage of heterogeneity in the system as all payments are based on actual cost of data acquisition. Thus if a node has unlimited energy resource (connected to power supply), its cost of task participation (as given by equation 6.1) will be very low and hence data paths using this node are more favorable. The usage of generic cost and utility functions allow DReL to support heterogeneity in various aspects such as sensing, energy etc. Multiple applications can be supported, as for a WSN each application is a set of tasks and any number of tasks can be deployed (as per the capacity) at a time. If two tasks across applications are identical (same attributes, QoS requirements etc.), all data can be shared across those tasks. If a source can support two tasks that are different, it will schedule a task with highest payment.

Even though our approach of task and data dissemination is based on one-phase pull diffusion [57], it doesn't suffer from issues arising due to asymmetric links prevailing in WSN because of following two reasons: i) each node chooses neighbor

for data dissemination based on cost of acquisition rather than lowest latency path for interest; ii) gradients are updated using reinforcement which is based on data-path (and not interest path) and hence takes care of asymmetric nature of links in WSN.

6.7 Performance Analysis

In this section we present results from simulation based performance analysis of DReL using an object tracking application where the objective is to track a target object in the sensor grid. The results are presented for different scenarios consisting of increasing number of sensor nodes ranging from 7 to 25 with a) constant network grid size (i.e. increasing redundancy and sensor node density) and b) constant node density (i.e. increasing grid size). Sensors can be many hops away from the base-station. All results are averaged over 10 simulation runs and are shown with confidence interval of 95% whenever applicable. Table 6.2 lists values of important parameters used for this simulation. The following metrics are used for performance analysis of DReL:

- *Average dissipated energy* is the ratio of total energy consumed per node per distinct event received by the sink.
- *Lifetime* is studied in terms of when the first node is depleted of its energy (*lifetime1*) and when the last event is received (*lifetime2*). Here *lifetime2* gives an indication of how long a system is able to continue functioning even after some critical nodes running out of energy.
- *Average delay* is the average time interval between the transmission of an event at the source to its reception at the sink computed for all events received at sink.
- *Event delivery ratio* is a ratio of the number of events received by the sink to the number of events the system was supposed to generate (based on data-rate and expected lifetime).

Table 6.2. Parameters used for simulation

Component	**Parameter**	**Value**
Micro-learner	Minimum exploration (ϵ_{min})	0.05
	Maximum exploration (ϵ_{max})	0.25
	Time-step (τ) sec	5
Macro-learner	Reinforcement threshold (m)	0.0
	Reinforcement window (N) time-steps	20
	Data exploration factor ($data_exp$)	0.1
Radio	Transmitting Current TX (amp)	0.02
	Receiving Current RX (amp)	0.01
	Idle Current (amp)	0.0002
	Sleep Current (amp)	10^-5
	Voltage (V)	1.0
Sink	Data-rate (sec)	5.0
	Interest Refresh Period (sec)	10000

6.7.1 Comparison with Directed Diffusion

We first compare performance of DReL with directed diffusion [13]. Figure 6.8 shows average dissipated energy per node per event (with error bar showing standard deviation for 95% confidence interval) against increasing number of sensor nodes. In the plot, *DReL-CD* and *Diffusion-CD* are for scenario with constant sensor density, while plots *DReL* and *Diffusion* are for scenario with constant network grid. As can be seen from the plots, DReL outperforms Diffusion by significant margin for all the tested scenarios in terms of energy expenditure per node per event. Moreover, with increase in number of nodes, DReL is able to further reduce average energy expenditure by managing resources at all times. Energy dissipation for Diffusion grows drastically with increase in number of nodes. This is because of large data broadcasting overhead in diffusion where each node is active and trying to participate in data diffusion process. On the other hand, in DReL only nodes in the data path with the least cost are participating in a data stream, while the rest of the nodes are in sleep

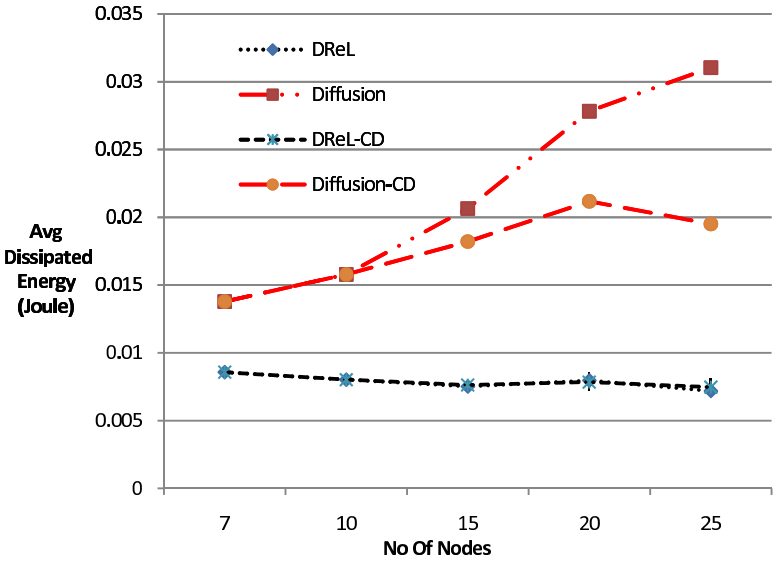


Figure 6.8. Average dissipated energy per node per event for different scenarios.

states. For constant density scenario, increase in energy with number of nodes for diffusion is less compared to that of constant grid scenario as nodes are comparatively sparse and hence data broadcasting overhead is low. For DReL, performance under both constant density and constant network grid scenarios is identical and DReL is able to keep energy dissipation low in all scenarios.

Lifetime1 and Lifetime2 (as defined above) are plotted in Figures 6.9 and 6.10 respectively. As can be seen from the plot, lifetime for DReL increases with addition of more nodes and hence it's able to utilize redundancy for improving the lifetime of the system. On the other hand, lifetime for diffusion reduces significantly with addition of nodes because of overhead involved in the diffusion process and lack of any resource management. After a certain point (20 nodes in these figures), addition of nodes doesn't improve lifetime for DReL - the effect of overhead with dense neighborhood equates/dominates over the benefits of adding extra nodes. Value of liftime2 is comparatively larger than lifetime1 for DReL, i.e. DReL is able to continue providing useful events even after a heavily utilized (most probably critical node

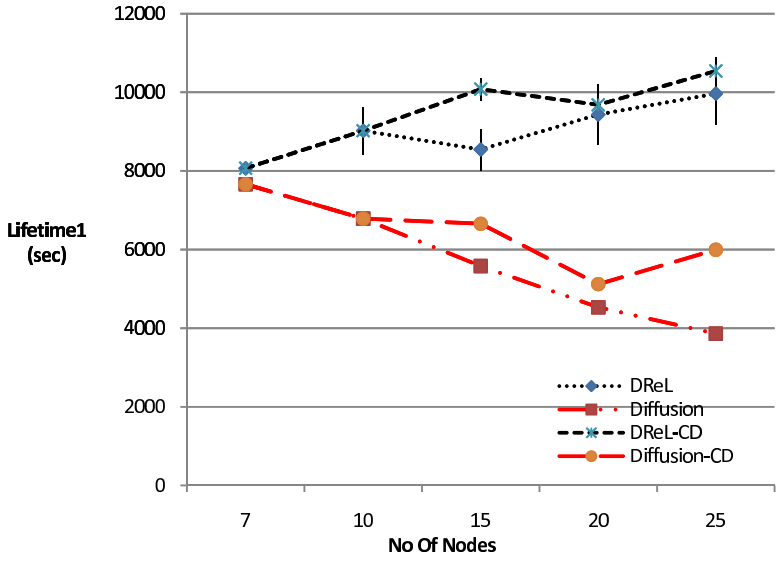


Figure 6.9. Lifetime in terms of energy depletion of first node (lifetime1).

across data path) node dies. As can be seen for most scenarios, lifetime2 for DReL is more than twice as that for original diffusion. Nodes are able to preserve energy in DReL by learning to tune their duty cycle and task execution based on system's state and application's need. When using DReL, increase in lifetime2 with nodes for constant grid scenario is higher compared to constant density scenario. This is because of DReL being able to take advantage of increase in redundant nodes with dense network available in constant grid scenario.

Figure 6.11 shows average delay for DReL and Diffusion for the scenarios of constant grid size and constant density. As can be seen, delay is varying across all scenarios and always increases with number of nodes as expected. On average Diffusion has lower delay compared to DReL for constant density scenario, but the difference is not that significant. The cost parameters used in the experiments for DReL are combination of both lifetime and number of hops and hence increase in delay is attributed to emphasis on lifetime improvement. As shown in next section, for applications requiring data with minimal delay, one can use cost parameter of

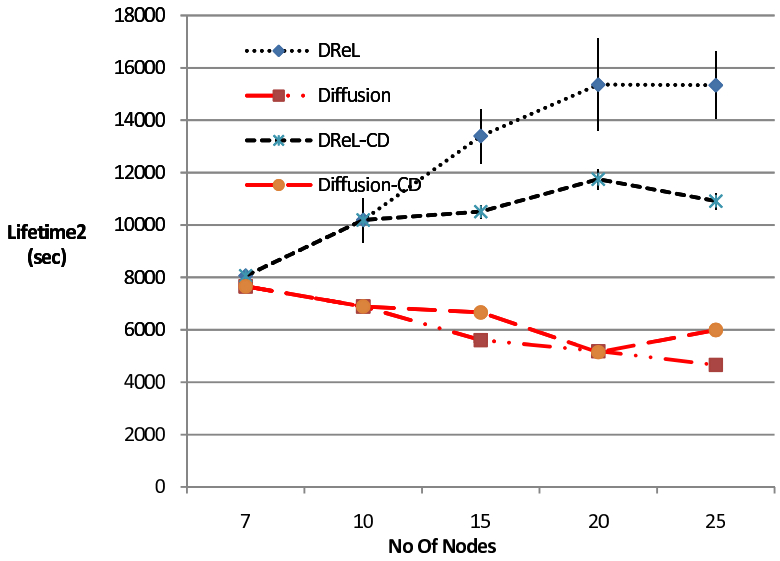


Figure 6.10. Lifetime in terms of last event received (lifetime2).

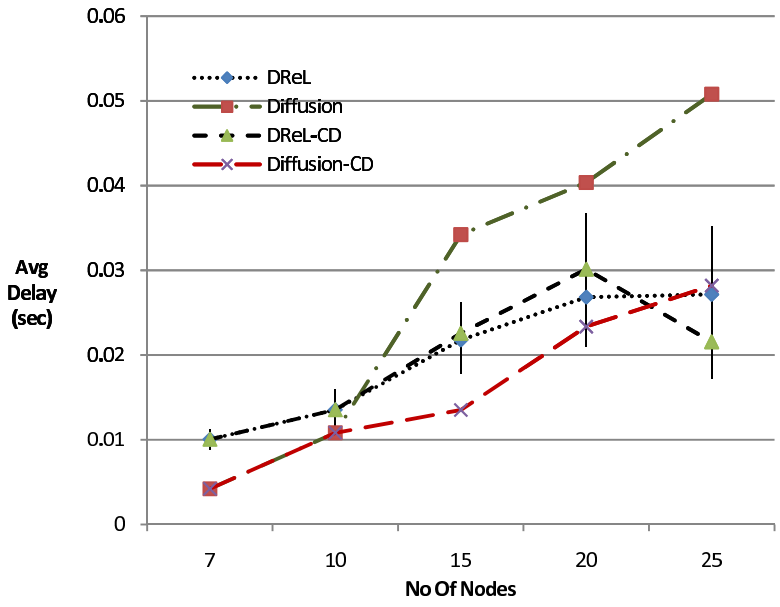


Figure 6.11. Average Delay for constant grid size and constant density scenario.

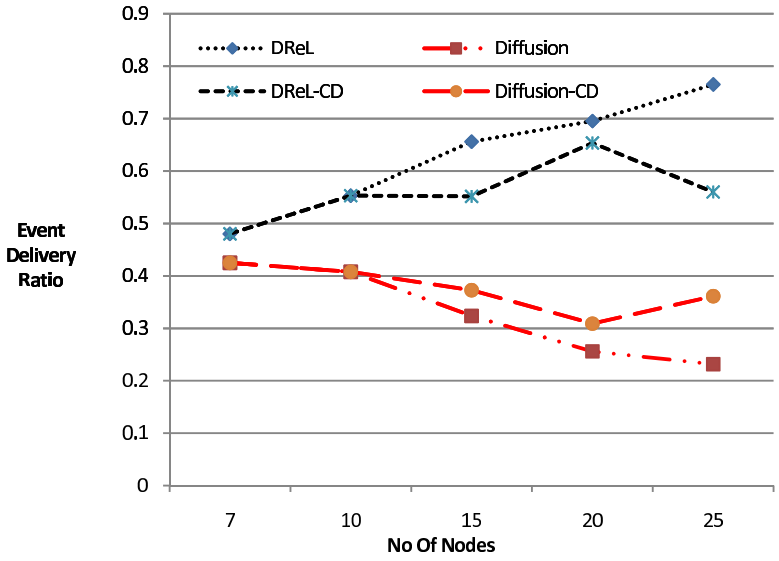


Figure 6.12. Event delivery ratio for constant grid size and constant density scenario.

NoOfHops only. In constant grid scenario, average delay for Diffusion is notably higher compared to DReL. This again shows inability of Diffusion to perform well in dense networks. On the other side, DReL is able to maintain delay within limits while allowing to increase system lifetime significantly.

Figure 6.12 shows event delivery ratio wherein DReL outperforms diffusion for all scenarios. Activity ratio for DReL is plotted in Figure 6.13. Activity ratio is defined as ratio of number of time-steps a node is active to total number of time-steps computed over all nodes in the system. DReL manages to reduce activity ratio with increase in number of nodes as only nodes required for data collection at any instance of time are active while other nodes preserve energy. Note here that activity ratio of Diffusion is always 1.

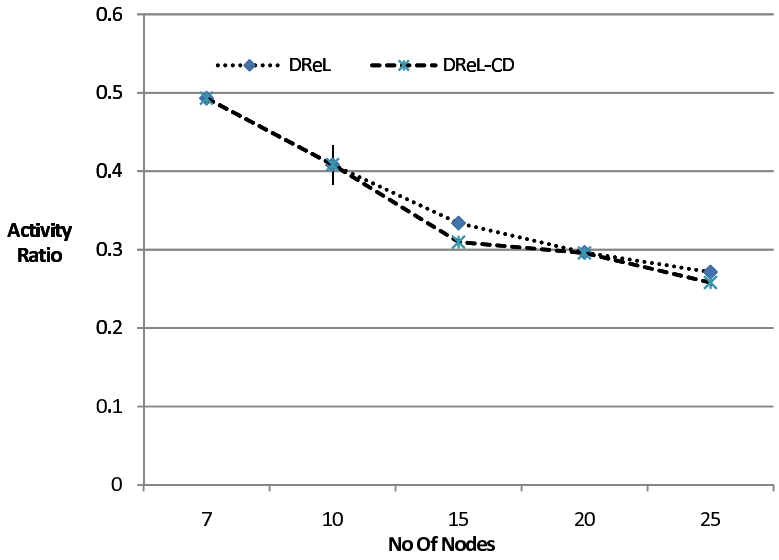


Figure 6.13. Activity ratio for DReL.

6.7.2 Analysis of global optimization

This section presents analysis of effectiveness of cost parameters and cost function in achieving the global optimization goal of the system. To study the impact of cost parameters on global optimization, next set of simulations with object tracking application are carried out with the following cost parameters: a) *Lifetime* with weight 1.0 and b) *NoOfHops* with weight 1.0. Figure 6.14 shows a snapshot of one of the simulation runs for the scenario with *Lifetime* (1.0) as cost parameter. As shown scenario consisted of 10 nodes including a target and a sink. The size of each sensor node corresponds to amount of energy it has available at any given time. Target in this case is stationary and is within sensing range of nodes 5 and 8. To illustrate the effect of cost function, nodes 4 and 8 are given higher energy (represented by larger size) compared to that of other sensor nodes. Node 8 is three hops away from sink, while node 5 is in communication range of node 1 and hence only two hops away from sink.

As shown in Figure 6.14, DReL quickly converges to desired path of 8 $\rightarrow$ 4 $\rightarrow$

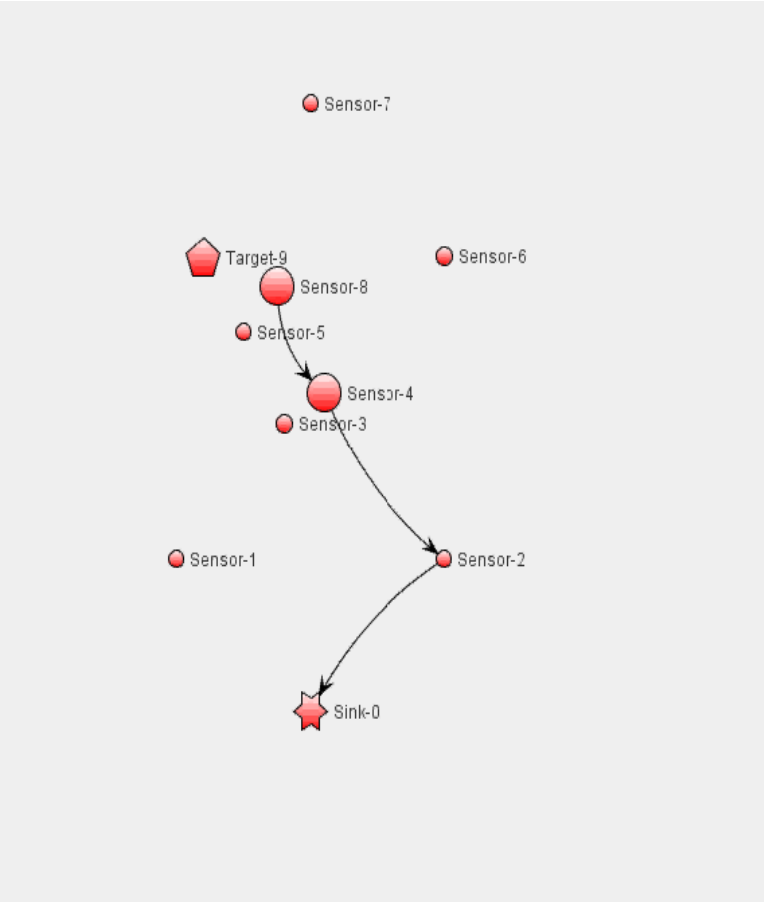


Figure 6.14. Snapshot of simulation run with Lifetime cost parameter.

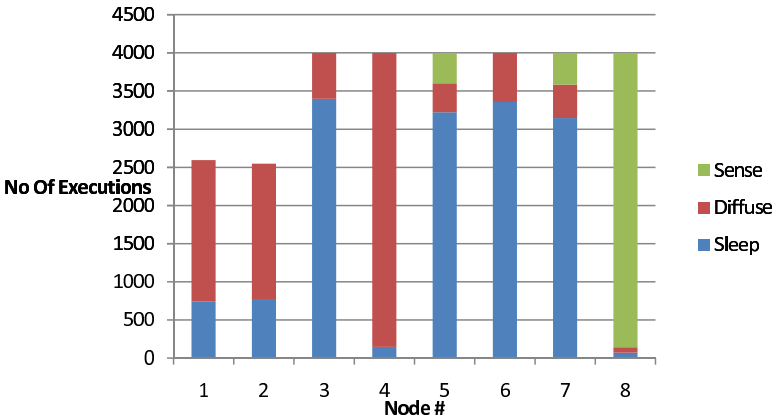


Figure 6.15. Number of executions of various tasks by sensor nodes with Lifetime cost parameter.

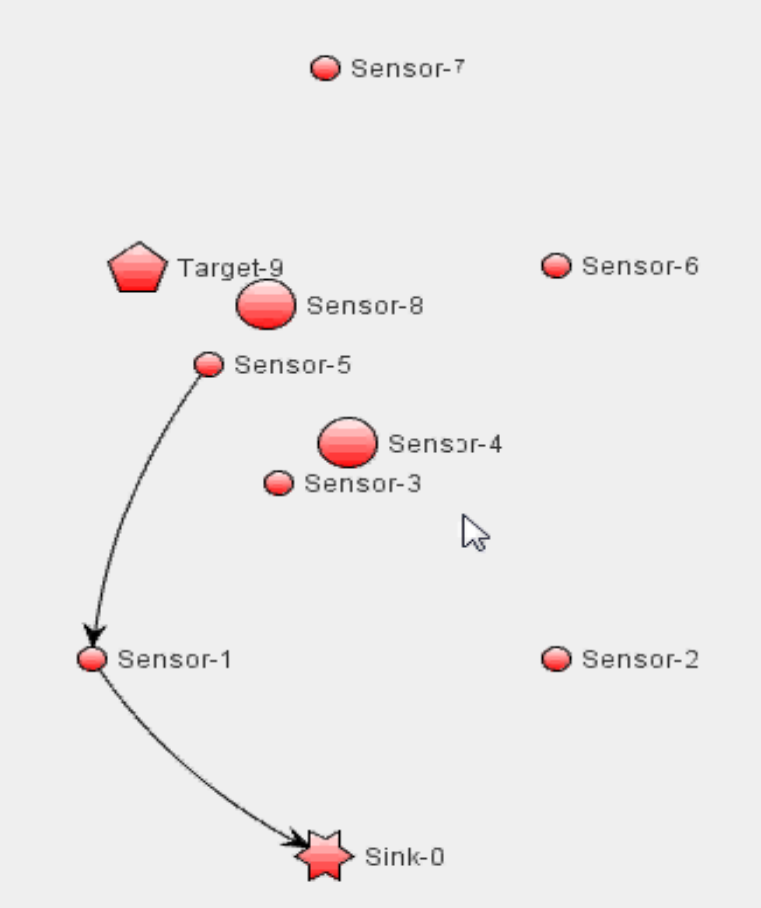


Figure 6.16. Snapshot of simulation run with *NoOfHops* cost parameter.

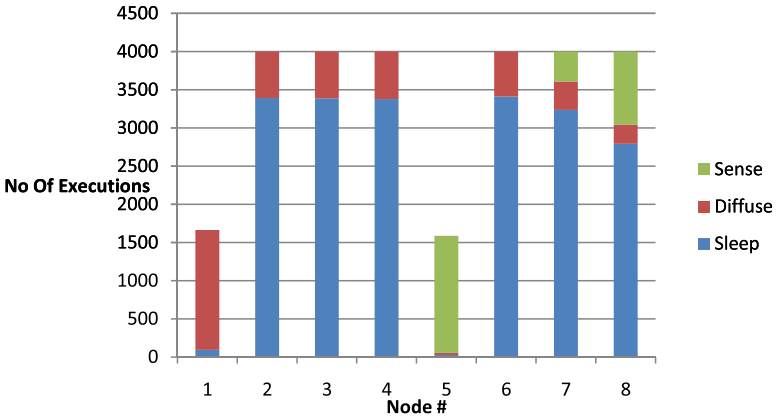


Figure 6.17. Number of executions of various tasks by sensor nodes with *NoOfHops* cost parameter.

$(1, 2) \rightarrow 0$ when using *Lifetime* as cost parameter. As nodes 4 and 8 have much higher energy and hence better lifetime, their cost of participation is negligible compared to other nodes and hence that path is desirable for the global system. This behavior is further demonstrated in Figure 6.15 where number of executions of various tasks involved i.e. Sleep, Diffuse and Sense are plotted for each sensor node in the system. Nodes 4 and 8, with highest lifetime, are the most heavily utilized nodes, with node 8 performing sensing while node 4 participating in the diffusion process for most part of system lifetime. Nodes 1 and 2 have almost equal number of executions of Diffuse task, implying that the DReL is effectively doing energy-balancing between those nodes to extend overall system lifetime. This energy balancing is made possible because of DReL's utility based reinforcement scheme and as lifetime of one node falls compared to adjacent neighbor, system will choose the neighbor instead. The size of reinforcement window as well as time interval for updating gradients/payable determines how often the above switch occurs. Note here that there is no centralized entity responsible for managing DReL and all interactions are confined to one-hop neighborhood.

Figures 6.16 and 6.17 demonstrate behavior of system in the scenario of using *NoOfHops* as cost parameter. This setting dictates system that application only cares about latency of data collection and doesn't care about lifetime. Lowest latency path in this scenario is $5 \rightarrow (1, 2) \rightarrow 0$ and as shown, DReL enables all nodes to quickly learn their role and schedule tasks accordingly. Node 5 is primarily acting as a source sensor and performing *Sense* tasks all the time while node 1 is primarily acting as a router (performing *Diffuse* task). As cost parameters consist of *NoOfHops* only, system lifetime is not considered and hence no energy balancing is possible between nodes 1 and 2 in this scenario. However, one can use both *Lifetime* and *NoOfHops*

Table 6.3. Performance for different cost parameters

Cost Parameter(s)	**Lifetime1**	**Lifetime2**	**Average Delay**	**Event Delivery Ratio**	**Average Dissipated Energy**
Lifetime(1.0)	9826.9 (±170.72)	13971.71 (±189.66)	0.0095 (±0.0008)	0.822 (±0.012)	0.0073 (±0.0002)
NoOfHops(1.0)	7939.3 (±16.20)	7934.20 (±15.95)	0.0071 (±0.0009)	0.498 (±0.0015)	0.0067 (±0.0002)
Lifetime(0.5) NoOfHops(0.5)	11734.54 (±106.56)	11708.81 (±107.28)	0.0089 (±0.0009)	0.633 (±0.001)	0.0086 (±0.0001)

as cost parameters with appropriate weights if effect of both should be considered for system operations (see Table 6.3).

Values for all performance metrics are shown with 95% confidence interval in Table 6.3. As expected, system lifetime (both lifetime1 and lifetime2) is improved considerably when *Lifetime(1.0)* is chosen as cost parameter. Although, cost parameter of *NoOfHops(1.0)* gives better performance in terms of average delay, lifetime2 is just about half compared to first scenario. As event delivery ratio is based on expected lifetime, its value is higher for the first scenario. Note here that average dissipated energy is actually higher in first scenario even though system has higher lifetime. This is because cost of participation is associated with actual lifetime of system and not the amount of energy spent. Thus energy spent by nodes with large energy resource doesn't matter much as far as lifetime is concerned. The last row in Table 6.3 shows results of compromise between lifetime and average delay.

6.7.3 Analysis of utility function

DReL uses Wonderful Life Utility (WLU) as a payoff (utility) function to determine reward for each data-stream terminating at each node. Next, WLU is compared

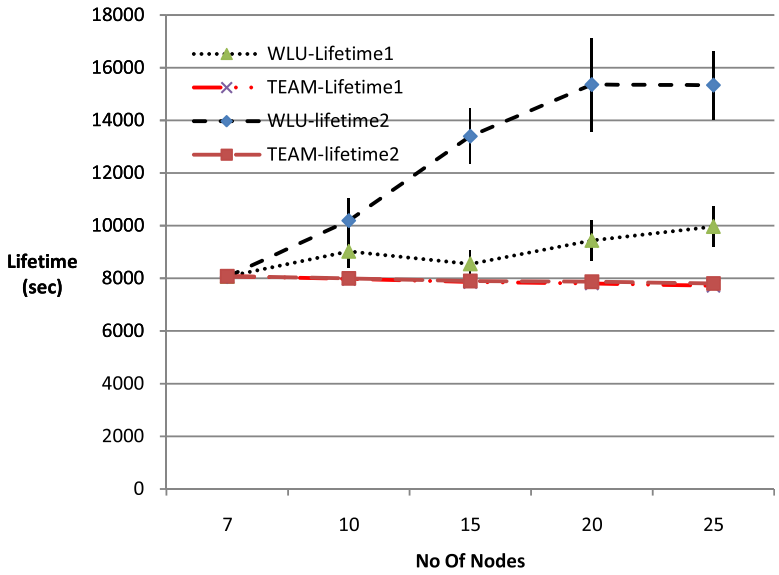


Figure 6.18. Lifetime comparison of WLU and TEAM utility functions.

with a more common approach in cooperative multi-agent systems of using a payoff function based on TEAM game. When using TEAM based payoff function, each data-stream shares the same amount of reward equal to global reward. This ensures that all nodes act cooperatively to increase their collective utility as they all share the same reward. Though, learnability of such utility function is very low as it is difficult for an individual node to determine its contribution towards global reward. In case of using TEAM utility function, reward for each data-stream as given by equation 6.6 is changed to the following:

$$g_{iwT}(\xi_T) = R_{iT}(\xi_{iT}) \tag{6.9}$$

Thus all streams share the same global reward regardless of their contribution towards global utility. Figures 6.18 and 6.19 compare performance of WLU and TEAM utility functions with increasing number of sensor nodes and constant grid size. Figure 6.18 compares two utility functions in terms of lifetime1 and lifetime2. As shown, when

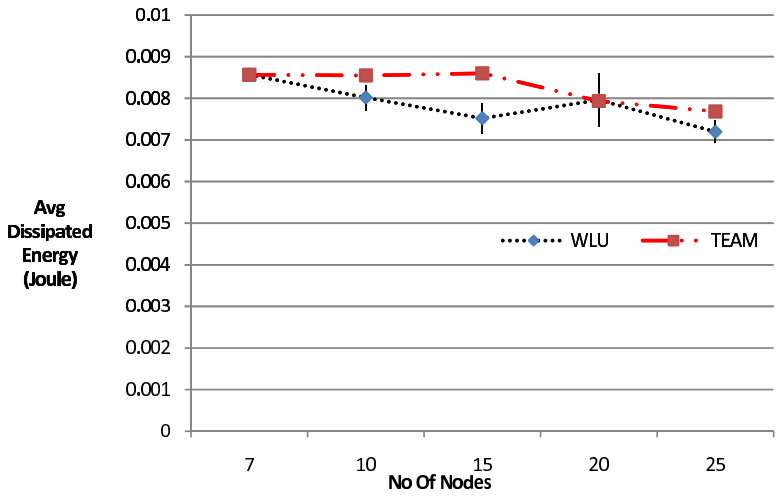


Figure 6.19. Comparison of average energy dissipation with WLU and TEAM utility functions.

using TEAM utility, DReL is not able to achieve any increase in lifetime with increase in number of nodes. Also values of both lifetime1 and lifetime2 are the same with TEAM utility. WLU on other side has much higher lifetime (particularly lifetime2) with increase in number of nodes. WLU enables each node to learn effect of its action towards global goal and hence results in better management of node's resources. As TEAM provides same reward to all streams, nodes involved in all data sreams will be motivated to continue participating and hence not being able to learn effect of their actions clearly. For example, if data provided by a stream is redundant and incurs higher cost compared to any other stream, TEAM provides same reward to both streams and resulting in waste of resources, while WLU provides positive reward to only one stream with minimum cost, enforcing nodes involved in other streams to conserve energy. Figure 6.19 shows average energy spent per node per received event for both schemes. In terms of energy expenditure, performance of both WLU and TEAM is similar with WLU being only slightly better in terms of energy usage. This is because along with total amount of energy consumed, number of tracking events

received in case of TEAM is also higher compared to WLU. Thus when comparing energy consumed per track, difference is not that significant.

6.7.4 Analysis using Intrusion Detection application

Thus far, we considered an object tracking application for performance analysis, where goal is to track a moving object. To prove the generality of our approach, we evaluate DReL for an intrusion detection application. We consider a WSN system consisting of both motion and video sensors employed for detecting an intruder. This application can be either in *monitoring* state or in *alarmed* state upon detecting an intruder. While in the *monitoring* state, requirement of confidence level is low, but application requires high quality data in the alarmed state. Ideally, to conserve energy, while in *monitoring* state, WSN system should employ only motion sensors and reserve video sensors for high quality data in *alarmed* state. DReL is able to do exactly so by managing sensor nodes at different states.

Implementing intrusion detection application on DReL mainly involves defining task description. Initially when application is in *monitoring* state, task is defined as follows.

- *attributes*: latitude, longitude of sensor node or intruder (if present) and confidence level of intrusion detection.
- *qosConstraints*: Confidence level greater than 0.2, coverage over all area required to be monitored
- *costParameters*: *Lifetime* with weight 1.0 as system should conserve energy while just monitoring and no intruder present.

When an intruder is detected, application transitions to *alarmed* state where its requirements are different as given below:

- *attributes*: latitude, longitude of intruder (if present) and confidence level of intrusion detection.
- *qosConstraints*: Confidence level greater than 0.8, coverage nearby intruder location
- *costParameters*: *NoOfHops* with weight 1.0 as system should be able to transmit high quality intruder data with minimum delay.

Assuming confidence level of detection provided by motion sensor is 0.25 and that of video sensor is 0.9, it is clear that only video sensors can satisfy application requirements in *alarmed* state. However, in monitored state, any sensor can satisfy the requirement. But the cost of data acquisition for a video sensor is much higher compared to that of a motion sensor. In DReL, each sensor node is able to decide whether or not it should be active and participate in the application at any state based on its learned utilities as demonstrated below.

Consider the scenario shown in Figure 6.20, for intrusion detection application consisting of 12 sensor nodes and a sink. Here three separate regions are being monitored and each region consists of one video sensor and two motion sensors. Thus sensors 6, 9 and 12 belong to same region and 12 is a video sensor. Optimally, in monitoring state, only one sensor out of each region should be active at any instance of time and others should conserve energy by sleeping. Further this active sensor should be a motion sensor as its cost of data acquisition is much lower than that of video sensor and both satisfies application requirement for confidence level. As shown in Figure 6.20, DReL is able to achieve above optimization using its neighborhood based multi-tier reinforcement learning scheme. Figure 6.21 shows number of executions of each task by all nodes in the system while in the *monitoring* state. All video sensors (10, 11 and 12) are sleeping most of the time after quickly getting reinforcement about better alternatives from neighboring nodes. DReL along with using only single

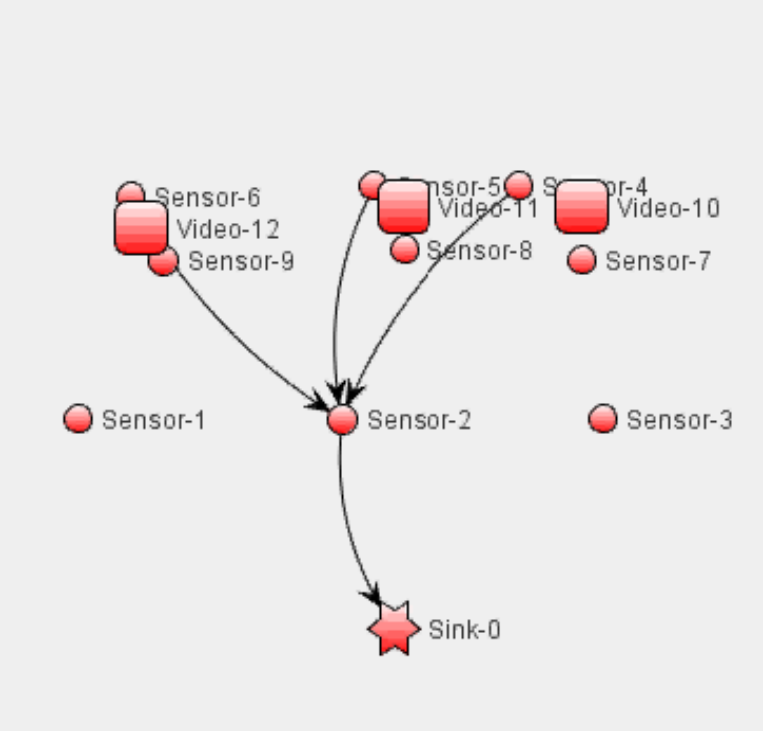


Figure 6.20. Snapshot of Intrusion Detection application in *monitoring* state.

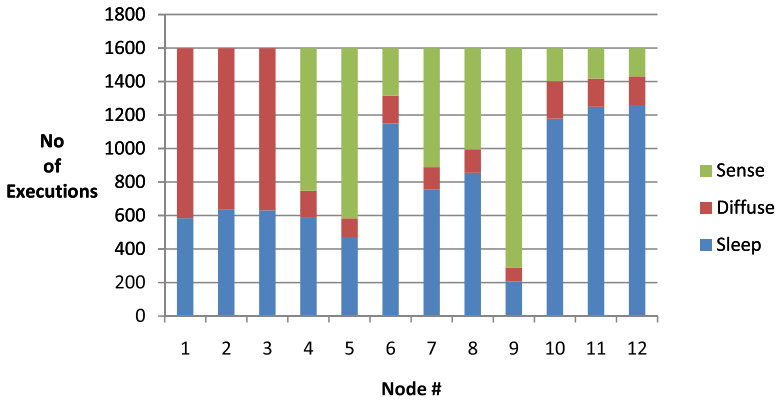


Figure 6.21. Number of executions of various tasks by sensor nodes in *monitoring* state.

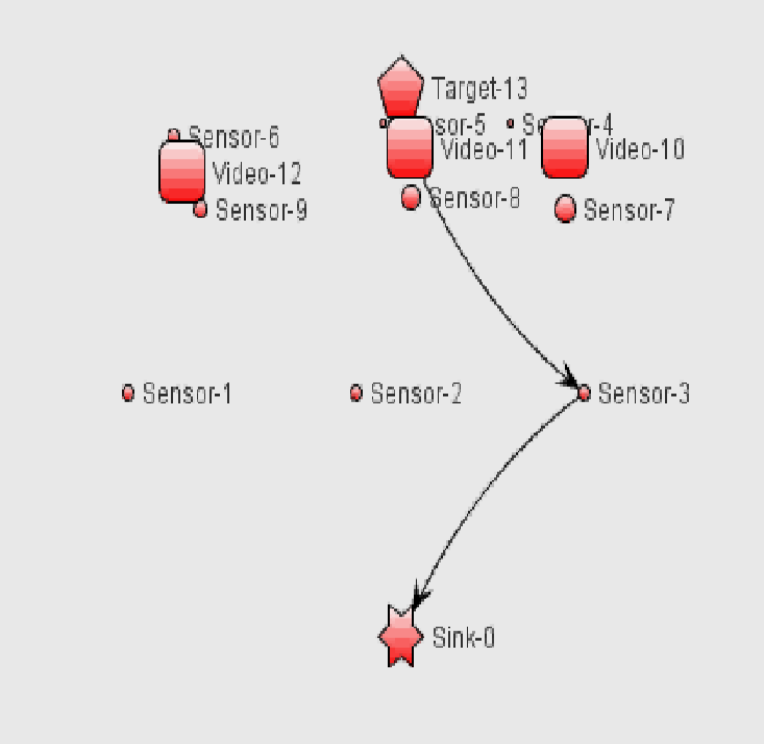


Figure 6.22. Snapshot of Intrusion Detection application in *alarmed* state.

motion sensor per region, is also able to energy-balance between available motion sensors as well as intermediate nodes helping in the diffusion process. Thus nodes 1, 2 and 3 are equally utilized as router effectively doing energy-balance between those nodes.

At around 1600 time-step (8000 sec), an intruder is introduced as shown in Figure 6.22 (marked as Target-13). This intruder is quickly detected by an active node in the region consisting of nodes (5, 8, 11) and event is sent out to the sink. This event triggers an application state change to *alarmed* and a modified interest (task packet) is sent out to reflect new requirements for *alarmed* state. Figure 6.23 plots task executions of all nodes in *alarmed* state. As shown, video sensor 11 is acting as a source in this state and tries to monitor the target, while node 3 is acting as a router helping to diffuse data towards sink. As we are using *NoOfHops* as cost

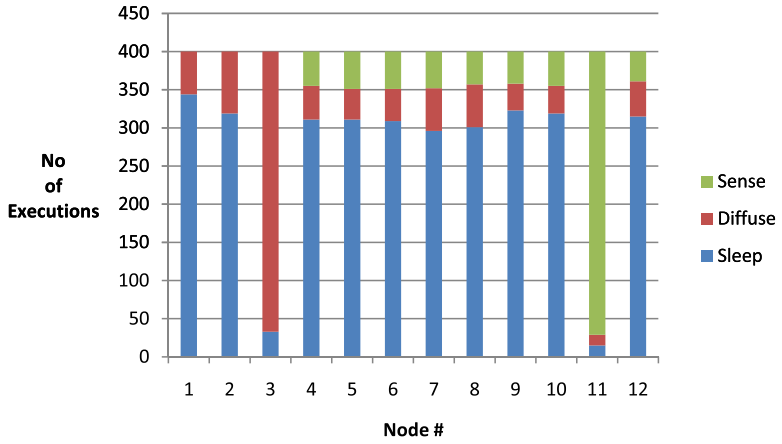


Figure 6.23. Number of executions of various tasks by sensor nodes in *alarmed* state.

parameter in *alarmed* state, there is no energy-balancing involved and system sticks to chosen path with minimal delay.

6.8 Conclusion

In this chapter, we have proposed DReL- a WSN management middleware utilizing techniques from multi-layer reinforcement learning and utility theory. Concepts from directed diffusion is incorporated to achieve data-centric communication paradigm for task, data and reward dissemination. Directed diffusion provides a robust communication paradigm for localized and distributed sensing, but fails to address adaptive sensor management and task scheduling necessary for development of efficient and flexible WSN applications. In DReL, we have addressed this issue with WSN management framework developed by extending directed diffusion and COIN based multi-layer reinforcement learning approach. We have emphasized on generality in designing the framework to allow development of a wide range of applications with different requirements in terms of quality of service, data collection, optimization etc. The framework utilizes an incentive compatible mechanism that provides feedback as well as incentives to individual nodes based on overall system performance and appli-

cation goal. The effectiveness of the proposed approach is illustrated through detailed performance studies for a tracking application. The generality of DReL is shown by implementing object tracking as well as simple intrusion detection application with changing application states and requirements.

CHAPTER 7

CONCLUSIONS

Support for autonomous and adaptive management is essential for many wireless sensor network (WSN) applications expected to be functioning over long periods of time. Additionally, WSN applications are becoming increasingly pervasive, requiring support for multiple applications executing simultaneously on heterogeneous sensor network infrastructure. Thus, rather than building a WSN infrastructure for each application, a single WSN infrastructure may be designed for use by a wide range of applications. In recent years, utility based computing has played a significant role in the proliferation of cloud computing allowing ease of development of applications using shared computing infrastructure. Similarly, one can envision use of utility theory to enable rapid development of pervasive applications on top of shared sensing infrastructure.

There are many proposed approaches for resource management in WSN, but most of them requires a centralized mechanism which is not scalable and robust for many real world WSN applications. None of the approaches try to address uncertainty which is inherent in dynamic networks. Furthermore, most of them require a careful implementation of algorithms on a case-by-case basis which may be quite difficult in sensor networks. Therefore, a generic framework that can enable large set of applications with autonomous adaptation and minimum communication overhead is required.

This dissertation explores techniques from reinforcement learning and utility theory to design a generalized distributed middleware solution that allows develop-

ment of a range of applications on top of heterogenous and resource constrained sensor network infrastructure. To the best of our knowledge, such a middleware framework that can support continuous autonomous adaptation using utility theory doesn't exist today. The work presented is unique in employing a bottom-up approach where each sensor node is responsible for its resource allocation/task selection while ensuring optimization of system-wide parameters like total energy usage, network lifetime etc.

7.1 Summary of Contributions

First contribution is in the form of a simple but effective distributed framework called DIRL. DIRL uses Q-learning/WoLF technique to enable development of adaptive WSN applications. DIRL has intrinsic support to handle dynamics and uncertainty prevailing in WSNs. Dynamism is handled by intelligent exploration throughout system lifetime while uncertainty is tackled by probabilistic actions based on learned utility values. DIRL is exemplified with an object tracking application and performance analysis of DIRL demonstrates its superiority over other approaches.

DIRL is next extended to a two-tier learning scheme in order to ensure optimization of system-wide application specified parameters. The bottom-up approach uses: micro-learning as used by individual nodes to self-schedule their tasks 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. Micro-learner provides autonomous real-time adaptation with minimal or no centralized processing requirement for task allocation and minimal communication overhead. Macro-learner uses COIN theory to make sure that system is actually meeting the global application objectives and steers system towards those objectives. Simulation results show that two-tier learning as used here can significantly improve overall performance compared to micro-learner alone or other traditional schemes. Application of COIN theory for

setting micro-learners utilities, guarantees eventual achievement of Pareto optimal point and avoids system getting trapped in TOC or other related phenomenon.

RADA is a framework for data collection in sparse Wireless Sensor Networks (WSNs) with Mobile Data Collectors (MDCs). RADA exploits DIRL to address the problem of energy-efficient MDC discovery. RADA allows sensor nodes to operate at a very low duty-cycle by learning MDC's arrival pattern and tuning its duty cycle based on the this pattern. Results show that the RADA framework is highly efficient in terms of low duty cycle, high discovery rate and high energy and data transfer efficiency. Compared to existing solutions, RADA not only performs better, but also can adapt to different operating conditions and mobility patterns characterized by high uncertainty. The generality of RADA framework allows applicability to wide range of application scenarios. As a result, it can be effectively used in the development of sparse WSN applications.

Finally, DReL is a complete WSN management middleware that defines an easy to use interface for development of adaptive WSN application. DReL describes data structures and mechanisms for task, data and reward management for micro-learner and macro-learner based two tier learning schemes. DReL allows development of wide range of applications with different requirements in terms of quality of service, data collection, optimization etc. The framework utilizes an incentive compatible mechanism that provides feedback as well as incentive to individual nodes based on overall system performance and application goal.

7.2 Future Research Directions

The research in this dissertation opens up several research directions related to resource management in WSN. Foremost among them is implementation of DReL on real sensor hardware and perform experiments in a WSN testbed. DReL middleware

can be implemented for a variety of sensor platforms viz. SunSpot, TinyOS etc. and make publicly available for further research and experimentation.

DReL only addresses on-demand task deployment where a task is published by application at runtime. Support for tasks available from deployment time should also be addressed in future research. This will require mechanisms for gradient setup without broadcast of task description. Design and analysis involving deployment of multiple applications simultaneously is another challenge that should be undertaken. Issues that need handling in case of multiple applications include sharing of common data and tasks among applications, scheduling of application tasks on single node, handling of conflicting application goals that are competing for common resources etc. Quality factor is not concretely defined in DReL and proper definition of a measure of quality to help evaluate reward/payment for a data-stream is left as a future work. Cost and reward functions can be further refined after trying to apply DReL to another set of applications with completely different requirements, e.g. schedule based rather than trigger based, multiple sinks, rapidly changing requirements etc. Future work should also include study of alternatives to Wonderful-life utility used in this dissertation. Mechanisms for co-ordination among nodes if a task spans across multiple nodes can also be placed in future work category.

Future work for RADA can include tuning of state weight parameters automatically based on learned mobility patterns of application rather than requiring manual configuration. RADA state definition can be further extended based on new application scenarios for sparse WSNs.

REFERENCES

[1] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18, no. 1, pp. 15–21, 2004.

[2] S. Hadim and N. Mohamed, "Middleware challenges and approaches for wireless sensor networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, March 2006.

[3] K. Römer, O. Kasten, and F. Mattern, "Middleware challenges for wireless sensor networks," *Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 59–61, 2002. [Online]. Available: http://doi.acm.org/10.1145/643550.643556

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, Mar. 2002. [Online]. Available: http://www.elsevier.nl/gejng/10/15/22/76/35/28/abstract.html

[5] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, no. 1, pp. 6–14, January 2004.

[6] T. Liu and M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems," in *PPoPP '03: Proceedings of the 9th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2003, pp. 107–118.

[7] P. J. Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel, "TinyCubus: a flexible and adaptive framework sensor networks," in *EWSN '05: Proceedings of the 2nd European Workshop on Wireless Sensor Networks*, 2005, pp. 278–289.

[8] "Citysense - an open, urban-scale sensor network testbed." [Online]. Available: http://www.citysense.net/

[9] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[10] D. Wolpert and K. Tumer, "Collective intelligence, data routing and braess' paradox," *J. Artif. Intell. Res. (JAIR)*, vol. 16, pp. 359–387, 2002. [Online]. Available: http://dx.doi.org/10.1613/jair.995

[11] ——, "An introduction to collective intelligence," *CoRR*, vol. cs.LG/9908014, 1999, informal publication. [Online]. Available: http://arxiv.org/abs/cs.LG/9908014

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *MOBICOM*, 2000, pp. 56–67. [Online]. Available: http://doi.acm.org/10.1145/345910.345920

[13] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," Sept. 28 2003. [Online]. Available: http://citeseer.ist.psu.edu/703281.html; http://www.isi.edu/ johnh/PAPERS/Intanagonwiwat02b.pdf

[14] J. H. D. Estrin, R. Govindan and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, Seattle, Washington, United States, 1999, pp. 263 – 270.

[15] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM SIGMOD*, vol. 31, no. 3, pp. 9–18, September 2002.

[16] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *USENIX Symposium on Op-*

erating Systems Design and Implementation (OSDI), Boston, USA, December 2002.

[17] C. Srisathapornphat, C. Jaikaeo, and C. chung Shen, "Sensor information networking architecture and applications," *IEEE Personal Communications*, vol. 8, pp. 52–59, 2001.

[18] S. L. S. H. Son, "Event detection services using data service middleware in distributed sensor networks," in *Distributed Sensor Networks*, 2003.

[19] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 85–95, Oct. 2002.

[20] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. G. Sirer, "On the need for system-level support for ad hoc and sensor networks," *ACM Operating Systems Review*, vol. 36, no. 2, pp. 1–5, 2002.

[21] TinyOS, "Tinyos: An open-source os for the networked sensor regime," http://www.tinyos.net. [Online]. Available: http://www.tinyos.net

[22] M. Wang, J. Cao, J. Li, and S. K. Das, "Middleware for wireless sensor networks: A survey," *J. Comput. Sci. Technol*, vol. 23, no. 3, pp. 305–326, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11390-008-9135-x

[23] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," *TAAS*, vol. 4, no. 3, 2009.

[24] A. Boulis, C.-C. Han, R. Shea, and M. B. Srivastava, "Sensorware: Programming sensor networks beyond code update and querying," *Pervasive and Mobile Computing*, vol. 3, no. 4, pp. 386–412, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.pmcj.2007.04.007

[25] R. Gummadi, N. Kothari, R. Govindan, and T. Millstein, "Kairos: a macro-programming system for wireless sensor networks," in *SOSP '05: Proceedings*

of the twentieth ACM symposium on Operating systems principles. New York, NY, USA: ACM, 2005, pp. 1–2.

[26] M. Welsh and G. Mainland, "Programming sensor networks using abstract regions," in *NSDI*. USENIX, 2004, pp. 29–42. [Online]. Available: http://www.usenix.org/events/nsdi04/tech/welsh.html

[27] K. Whitehouse, C. Sharp, D. E. Culler, and E. A. Brewer, "Hood: A neighborhood abstraction for sensor networks," in *MobiSys*. USENIX, 2004.

[28] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.8091; http://www.vs.inf.ethz.ch/publ/papers/sensys05.roleassignment.ps

[29] K. Terfloth, G. Wittenburg, and J. H. Schiller, "FACTS - A rule-based middleware architecture for wireless sensor networks," in *COMSWARE*. IEEE, 2006.

[30] S. Kogekar, S. Neema, B. Eames, X. Koutsoukos, A. Ledeczi, and M. Maroti, "Constraint-guided dynamic reconfiguration in sensor networks," in *Proceedings of the third international symposium on Information processing in sensor networks (IPSN-04)*. New York: ACM Press, Apr. 26–27 2004, pp. 379–387.

[31] B. Krishnamachari, S. B. Wicker, R. Béjar, and C. Fernández, "On the complexity of distributed self-configuration in wireless networks," *Telecommunication Systems*, vol. 22, no. 1-4, pp. 33–59, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1023426501170

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

[33] V. Lesser, C. Ortiz, and M. Tambe, *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publisher, 2003.

[34] C. Ortiz, T. Rauenbush, E. Hsu, and R. Vincent, *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publisher, 2003, ch. Dynamic resource-bounded negotiation in non-additive domains, pp. 61–106.

[35] L.-K. Soh, C. Tsatsoulis, and H. Sevay, *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publisher, 2003, ch. A satisfying, negotiated and learning coalition formation architecture, pp. 109–137.

[36] J. W. Byers and G. Nasser, "Utility-based decision-making in wireless sensor networks," in *MobiHoc*. ACM, 2000, pp. 143–144. [Online]. Available: http://doi.acm.org/10.1145/514151.514178

[37] N. Sadagopan, M. Singh, and B. Krishnamachari, "Decentralized utility-based sensor network design," *MONET*, vol. 11, no. 3, pp. 341–350, 2006. [Online]. Available: http://dx.doi.org/10.1007/s11036-006-5187-8

[38] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, adaptive resource allocation for sensor networks," in *NSDI '05: Proceedings of the 2nd Symposium on Networked Systems Design & Implementation*, 2005, pp. 315–328.

[39] W. Zhao and M. Ammar, "Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks," in *FTDCS '03: 9th IEEE International Workshop on Future Trends of Distributed Computing Systems*, May 2003, pp. 308–314.

[40] H. Jun, M. Ammar, and E. Zegura, "Power management in delay tolerant networks: A framework and knowledge-based mechanisms," in *SECON '05: Proceedings of the 2nd IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2005, pp. 418–429.

[41] Y. Gu, D. Bozdag, E. Ekici, F. Ozguner, and C. Lee, "Partitioning based mobile element scheduling in wireless sensor networks," in *SECON '05: Proceedings*

of the 2nd IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2005, pp. 386–395.

[42] Y. Gu, D. Bozdag, and E. Ekici, "Mobile element based differentiated message delivery in wireless sensor networks," in *WoWMoM '06: Proceedings of the 7th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2006, pp. 83–92.

[43] A. Kansal, A. Somasundara, D. Jea, M. Srivastava, and D. Estrin, "Intelligent fluid infrastructure for embedded networks," in *Mobisys '04: Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications and Services*, 2004, pp. 111–124.

[44] A. Somasundara, A. Kansal, D. Jea, D. Estrin, and M. Srivastava, "Controllably mobile infrastructure for low energy embedded networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, pp. 1536–1233, 2006.

[45] A. Chakrabarti, A. Sabharwal, and B. Aazhang, "Using predictable observer mobility for power efficient design of sensor networks," in *IPSN '03: Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, 2003, pp. 129–145.

[46] G. Anastasi, M. Conti, and M. Di Francesco, "Data collection in sensor networks with Data Mules: an integrated simulation analysis," in *ISCC '08: Proceedings of the 13th IEEE Symposium on Computers and Communications*, 6-9 July 2008, pp. 1096–1102.

[47] ——, "An analytical study of reliable and energy-efficient data collection in sparse sensor networks with mobile elements," in *EWSN '09: Proceedings of the 6th European Conference on Wireless Sensor Networks*, 11-13 February 2009, pp. 199–215.

[48] ——, "Reliable and energy-efficient data collection in sparse sensor networks with mobile elements," *Performance Evaluation*, vol. 66, no. 12, pp. 791–810, 2009.

[49] S. Jain, R. Shah, W. Brunette, G. Borriello, and S. Roy, "Exploiting mobility for energy efficient data collection in wireless sensor networks," *ACM/Springer Mobile Networks and Applications*, vol. 11, no. 3, pp. 327–339, June 2006.

[50] G. Anastasi, M. Conti, E. Monaldi, and A. Passarella, "An adaptive data-transfer protocol for sensor networks with Data Mules," in *WoWMoM '07: Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2007, pp. 1–8.

[51] P. Baruah, R. Urgaonkar, and B. Krishnamachari, "Learning-enforced time domain routing to mobile sinks in wireless sensor fields," in *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 525–532.

[52] V. Dyo and C. Mascolo, "Efficient node discovery in mobile wireless sensor networks," in *DCOSS '08: Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems*, 2008, pp. 478–485.

[53] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco, "Mobile data collection in sensor networks: The TinyLime Middleware," *Elsevier Pervasive and Mobile Computing Journal*, vol. 4, no. 1, pp. 446–469, December 2005.

[54] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press, 1998. [Online]. Available: http://www.cs.ualberta.ca/ sutton/book/ebook/the-book.html

[55] K. Tumer and D. H. Wolpert, *Collectives and the design of complex systems*. Springer-Verlag, 2004.

[56] C. yee Chong, Ieee, S. P. Kumar, and S. Member, "Sensor networks: evolution, opportunities, and challenges," in *Proceedings of the IEEE*, 2003, pp. 1247–1256.

[57] J. Heidemann, F. Silva, and D. Estrin, "Matching data dissemination algorithms to application requirements," in *Proceedings of the ACM SenSys Conference*. Los Angeles, California, USA: ACM, Nov. 2003, pp. 218–229. [Online]. Available: http://www.isi.edu/ johnh/PAPERS/Heidemann03b.html

[58] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *AAAI/IAAI*, 1998, pp. 746–752.

[59] R. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA 01003, 1984, published as COINS Technical Report 84-2.

[60] S. Mahadevan and J. Conell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, no. 2, pp. 311–365, 1992.

[61] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, England, 1989.

[62] M. H. Bowling and M. M. Veloso, "Multiagent learning using a variable learning rate," *Artif. Intell*, vol. 136, no. 2, pp. 215–250, 2002. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(02)00121-2

[63] "J-sim: component-based, compositional simulation environment." [Online]. Available: http://sites.google.com/site/jsimofficial/

[64] A. Sobeih, W.-P. Chen, J. C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and H. Zhang, "J-sim: A simulation environment for wireless sensor networks," in *Annual Simulation Symposium*. IEEE Computer Society, 2005, pp. 175–187. [Online]. Available: http://dx.doi.org/10.1109/ANSS.2005.27

[65] H. Alex, M. Kumar, and B. Shirazi, "Midfusion: An adaptive middleware for information fusion in sensor network applications," *Information Fusion*, vol. 9, no. 3, pp. 332–343, 2008. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2005.05.007

[66] J. F. Kurose and K. W. Ross, *Computer Networking – A Top-Down Approach Featuring the Internet*, 5th ed. Addison-Wesley Professional, 2009.

[67] G. Anastasi, M. Conti, E. Gregori, C. Spagoni, and G. Valente, "Motes sensor networks in dynamic scenarios," *International Journal of Ubiquitous Computing and Intelligence*, vol. 1, no. 1, April 2007.

[68] C. Technology, "Mica2 wireless measurement system." [Online]. Available: https://www.eol.ucar.edu/rtf/facilities/isa/internal/CrossBow/DataSheets

[69] J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. N. Chun, "Why markets could (but don't currently) solve resource allocation problems in systems," in *HotOS*. USENIX Association, 2005.

[70] F. Silva, J. Heidemann, R. Govindan, and D. Estrin, "Directed diffusion," Feb 2004.

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

BIOGRAPHICAL STATEMENT

Kunalbhai Shah received his Bachelor of Engineering degree from Sardar Vallabbhai National Insitute of Technology Surat, India in 1999. He spent a year and a half working as a Lecturer at Nirma University. After finishing his Masters in Computer Science in 2003 from Lamar University, Texas, Kunalbhai worked full time as Design Software Engineer for SensorLogic Inc. for 5 years. He began his PhD in 2005 at the University of Texas at Arlington. His research interests include design and development of adaptive middlewares for wireless sensor and pervasive networks, application of reinforcement learning, utility theory and distributed computing. Since 2008, he is working as a Senior Software Engineer at Securus Technologies Inc.