EFFICIENT FRAMEWORKS FOR LIFETIME MAXIMIZATION IN TREE BASED SENSOR NETWORKS

by

SK KAJAL AREFIN IMON

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
May 2014

To my mother and my father

who set the example and who made me who I am.

Acknowledgements

I would like to thank Dr. Matthew Wright for serving as my supervising professor. I would also like to express my deep gratitude to Dr. Sajal K. Das for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral studies. I wish to thank my other committee members Dr. Gautam Das, and Dr. Yonghe Liu for their interest in my research and for taking time to serve in my dissertation committee.

I would like to express my gratitude to Dr. Lynn Peterson for her support. I also thank Dr. Bahram Khalili and Dr. Ramez Elmasri for their help throughout my graduate career. I would like to extend my appreciation to graduate school of UT Arlington for providing financial support for my doctoral studies. I am grateful to all the teachers who taught me during the years I spent in school, first in Bangladesh, and later in the Unites States.

Finally, I would like to express my deep gratitude to my parents, and my brother who have encouraged and inspired me all through my undergraduate and graduate studies. I am fortunate to be so blessed. I am also extremely grateful to my wife for her sacrifice, encouragement and patience. I also thank my friends and fellow members of the CReWMaN lab at UTA who have helped me throughout my graduate studies.

April 14, 2014

Abstract

EFFICIENT FRAMEWORKS FOR LIFETIME MAXIMIZATION
IN TREE BASED SENSOR NETWORKS

Sk Kajal Arefin Imon, Ph.D.
The University of Texas at Arlington, 2014

Supervising Professor: Matthew Wright

In most wireless sensor network (WSN) applications, data are typically gathered by the sensor nodes and reported to a data collection point, called the sink. In order to support such data collection, a tree structure rooted at the sink is usually defined. Based on different aspects, including the actual WSN topology and the available energy budget, the energy consumption of nodes belonging to different paths in the data collection tree may vary significantly. This affects the overall network lifetime, defined in terms of when the first node in the network runs out of energy.

In this thesis, we address the problem of lifetime maximization of WSNs in the context of data collection trees through load balancing and data compression techniques. From load balancing perspective, we propose a novel and efficient algorithm, called Randomized Switching for Maximizing Lifetime (RaSMaLai) that intelligently changes the path (toward the sink) of sensors to distribute traffic load. We analytically show that, under appropriate settings of the operating parameters, RaSMaLai converges with a low time complexity. We further design a distributed version of our algorithm, called D-RaSMaLai. While D-RaSMaLai works on the same principals of

RaSMaLai, the design of the distributed version is novel and energy efficient. Simulation results show that both the proposed algorithms outperform several existing approaches in terms of network lifetime.

We also approach the lifetime maximization problem leveraging compression of sensor data streams. Compression of correlated data is one of the widely used techniques where the amount of transmitted data is minimized along their routes towards the sink. Existing works in this direction do not consider the temporal effect of correlation among data streams generated by periodic sensing. Moreover, the compression can introduce some imperfection that may affect the reliability of the collected data. In this thesis, we address the problem of energy efficient data gathering in WSNs while considering variability of correlation among data streams of neighboring sensors. We perform experiments on real data sets and show that our framework is very energy efficient, and contributes to lifetime maximization.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

Wireless sensor network has emerged as a popular technique for monitoring our physical environment. Typically, a wireless sensor network or WSN consist of a number of sensor nodes and a sink node. The sensor nodes are capable of sensing physical parameters like temperature, air pressure, humidity etc. In addition, these nodes are small battery-powered devices with limited resources and capable of communicating wirelessly. The sink node, traditionally a computer, is connected with the main power line and acts as a repository for data collected by the sensor nodes. Each sensor collect data from the environment with regular time intervals. Since they have limited battery power, they are put into sleep mode as much as possible to save energy. When a sensor has some data, it transmits it to one of its neighbors. Eventually, data from all the sensors that are collected over a time period are forwarded to the sink node. To successfully collect the sensory data at the sink, the sensor nodes must form a communication network. In most scenarios, the sensed data are reported to a data collection point (i.e., the sink), thus organizing the participating sensor nodes into what is called a (logical) *data collection tree* [1] rooted at the sink.

Due to their huge potential, wireless sensor networks are being deployed in a wide variety of applications that range from environmental monitoring and surveillance to event detection and healthcare. They often operate unattended, and may be randomly deployed over the monitoring (sensing) area due to roughness of the terrain or inaccessibility of the physical environment.

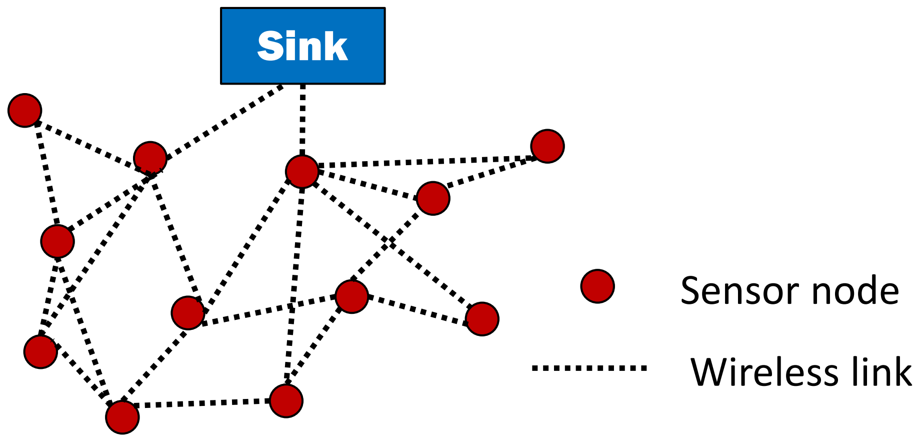

Figure 1.1: A sensor network

We show an example of WSN in the Figure 1.1. The red circles represent sensor nodes and the dotted lines represent the wireless links among neighboring sensor nodes. From the figure we notice that, some sensor node can reach the sink through multiple routes using different neighbors. Since, the collected sensory data must be routed to the sink, a routing strategy must be adopted. On the other hand, sensor nodes have a limited energy budget. Consequently, lifetime maximization [2, 3] of the network is one of the most important challenges in WSNs. Therefore, extensive research has been proposed in the literature to design efficient routing to maximize the lifetime.

1.1 Definition of Lifetime

The term lifetime in the context of wireless sensor network have been defined in different ways in different body of works. A detailed discussion can be found in [4]. Here we mention the well known definitions of lifetime of a sensor network.

- *Time until the first node dies*: This definition is the most popular one for lifetime mximization problems. It is the most useful definition in case of periodic data collection. Periodic data collection is very common in sensor applications.

Monitoring applications requiring continuous data from the environment fall under this category. Sensors collect and tranmit data in regular intervals in periodic monitoring. In such applications, the routing structure is often designed as a tree with the sink node as the root of the tree [5]. When a node runs out of energy, some restructuring is needed to maintain such tree structure. Thus, in tree based structures for periodic monitoring applications, the lifetime is defined as the time until the first node dies. This definition is also useful where high reliability and 100% coverage of the monitoring area are required. In this thesis, we adopt this definition of lifetime for WSNs.

- *Time until the K of N node die*: Other than periodic monitoring, sensor networks can also be used for event based monitoring. In such applications, sensors only transmit data if there are some events. Since, there is no regular intervals for sensors to be active, collect, and transmit data, they follow a random schedule for activity period. This is known as duty cycle [6]. In duty cycled based applications, a sensor randomly wakes up, and communicates with any neighbor that is also awake. If no neighbor is awake, it will go to sleep mode to save energy. There is no fixed route as sensors become active randomly, and it results in multipath routing. In multipath routing, data packets from the same source node may follow different path to the sink. As a result, failure of a single sensor does not make much difference to the routing. However, if there are a significant number of sensors running out of energy, it may either result into loss of desired coverage of the monitoring area or a disconnected network or both. Consequently, in multipath routing strategies, it is more practical to define network lifetime as the time until a set of K nodes among the total of N nodes die, making the network disconnected.

- *Time until N of N node die*: Although not much practical, in some sensor applications, the network may be assumed functional as long as there is at least one sensor that can communicate with the base station. For example, if the monitoring area is very samll and only one sensor is enough to cover the entire area, the application can operate until all of the N sensors die.

1.2 Lifetime Maximization Techniques

Since sensor nodes have a limited energy budget, lifetime maximization is one of the most important challenges in WSNs. Therefore, extensive research has been proposed in the literature on this topic. Popular lifetime maximization approaches exploit techniques like efficient duty-cycling [7], data compression [8, 9], and load balancing [10, 11, 2] among others. We discuss these approaches in the following:

1.2.1 Duty-cycling Approaches

In duty-cycling based approaches, sensors nodes are programmed a predefined duty-cycle or sleep-wake up schedule. The main goal is to put the sensors in the sleep state as much as possible to save energy. Since, sensors usually have a low rate of data collection, it is not necessary to keep them in awake mode all the time. In duty cycle based approaches, a sensor can transmit its data to its neighbor only if the node itself and the neighbor is in awake mode. Such a scenario is shown in the Figure 1.2. The sleep-wake schedule can either be fixed or be randomized. When the schedule is fixed, each node has a specific time to wake up and collect or transmit data. In a randomized scheme, nodes wake up probabilistically after an interval. Though randomized schemes are easy to design, it can not guarantee an estimate on certain routing parameters, such as the amount of delay or the number of hops a packet travels to reach the sink.

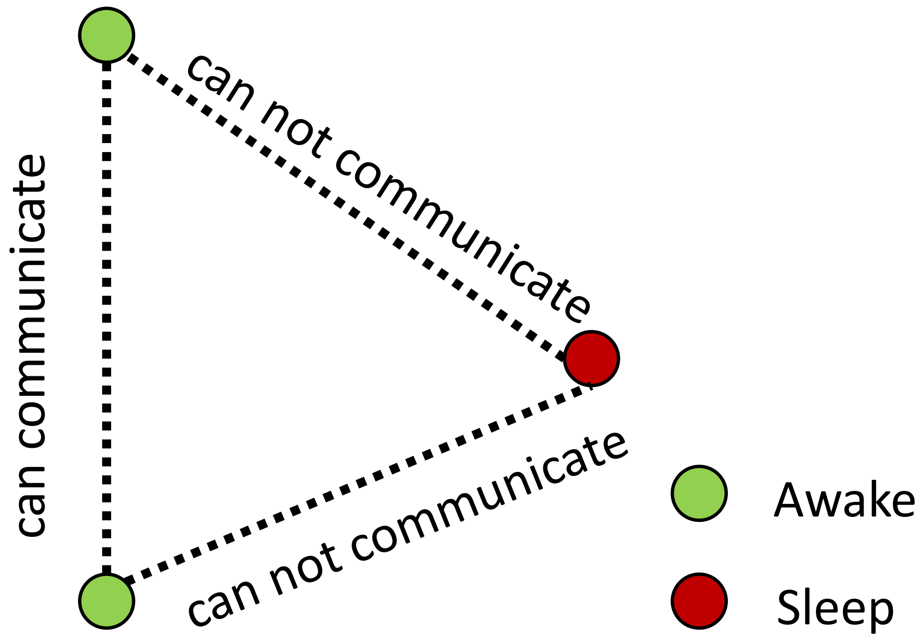

Figure 1.2: Example of duty cycle

1.2.2 Compression Approaches

In compression based lifetime maximization approaches, some of sensors in the network act as compression nodes. The data compression reduces the amount of data that needs to be transmitted. Since, sensors spend most of its energy during transmission and reception, rather than the sensing activity, compression helps to save energy by reducing traffic volume [12]. In Figure 1.4, we show a branch of a WSN with four sensors. The sensor node marked in green takes inputs from two neighbors We assume the amount of data taken from the two neighbors is $x1$ and $x2$, and the green node has z amount of its own data. Since the green node acts as a compression node, it applies some compression algorithm and produces the outgoing data $y < x1 + x2 + z$.

Compression-based schemes explicitly attempt to reduce the amount of data transmission in such a way that sensor nodes have less energy expenditure in terms

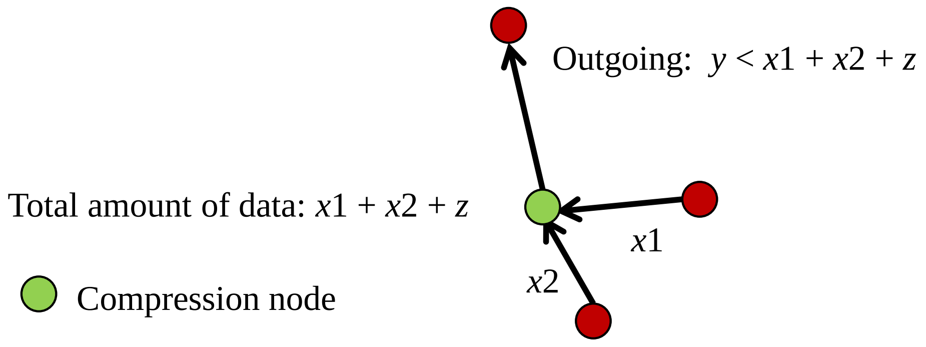
Outgoing: $y < x1 + x2 + z$
Total amount of data: $x1 + x2 + z$
$x1$
$x2$
Compression node

Figure 1.3: In network data compression

of data forwarding. Nodes in close proximity often have high correlation in their sensed data. Thus, data compression approaches are suitable to address the lifetime maximization problem [13] by creating a *compression* tree. In the context of this dissertation, a compression tree is a data collection tree with a set of nodes performing data compression. Also, we use the terms data aggregation and data compression interchangeably.

1.2.3 Load Balancing Approaches

Unlike the first two approaches, load balancing-based schemes explicitly attempt to organize the network topology in such a way that sensor nodes have uniform loads in terms of data forwarding. Since nodes closer to the sink have higher traffic to forward, they run out of their energy earlier. Thus, load balancing approaches are suitable to address the lifetime maximization problem [2, 3] by creating a *balanced* data collection tree.

In the Figure 1.4, we show a WSN with an embedded data collection tree. The solid lines represent connection between two neighboring sensors with the arrow head signifying the outgoing link. If a sensor knows which of its neighbors it will forward its data, we call the forwarding node as the *child node* and the node beign forwarded

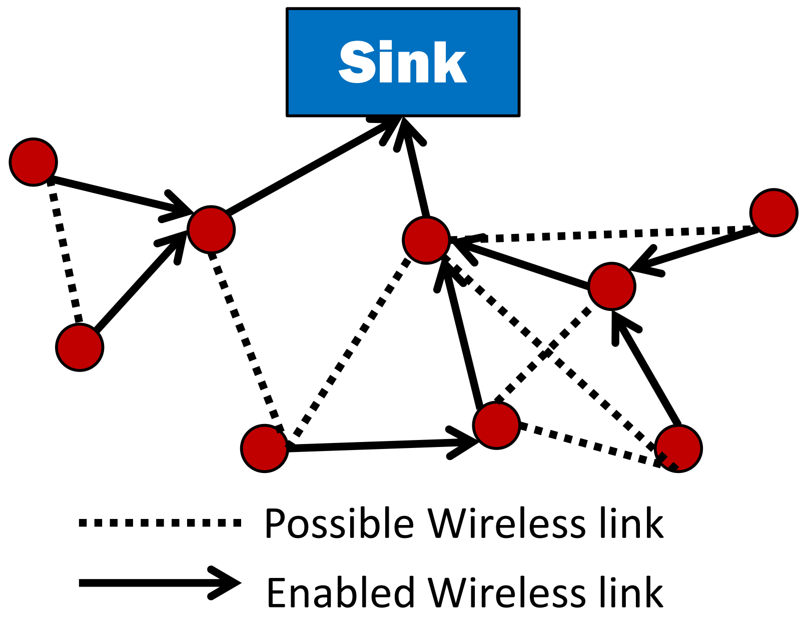

Figure 1.4: A data collection tree

to as the *parent node*. By defining a parent node as the upstream node to transmit the data, a tree can be formed with the sink node as the root, also known as the data collection tree.

Adopting data collection tree as the routing topology for any given WSN has several striking advantages. First of all, since a tree structure is well defined, it makes routing simpler. Secondly, a tree structure can help to define a deterministic duty cycle schedule, mitigating the channel intererence. Such deterministic schedule reduces the uncertainty of delay bound [14]. Also, tree structures are suitable for traffic load balancing by proper selection of the parent nodes. We discuss the benefits of data collection trees for modeling of the lifetime maximization problem of WSNs in more details in Section 1.4.

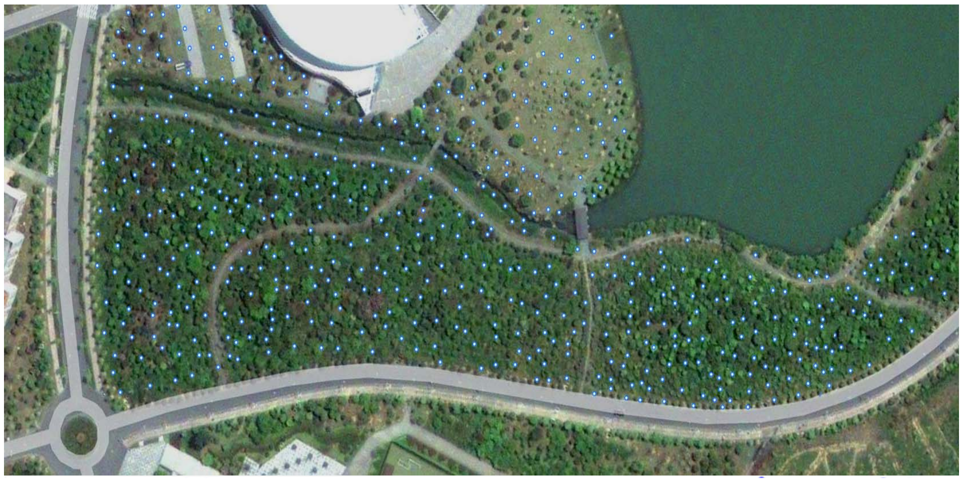

Figure 1.5: Sensor network deployed in a forest

1.3 A Real World WSN Example

In this Section, we discuss a real world example of WSN deployment for monitoring environmental data. In Figure 1.5, the wireless sensor network deployment of the GreenOrbs [15] project is shown. This project aims for a measurement study of a large scale deployment with 330 sensor nodes deployed in a forest of eastern china. The network topology of this deployment is shown in Figure 1.6.

From the study of network dynamics of this large deployment, several properties have been discovered. Firstly, the load of data traffic is distributed in a non-uniform way. Specifically, only a few nodes carried about 95% of the total traffic. Consequently, though there are 330 sensors, only a handful of them deplete energy at a much higher rate than most of the others. As a result, these bottleneck nodes die early, while a lot of sensors still have sufficient energy. When these bottleneck nodes die, the routing topology disrupts, as the network start to get disconnected. Eventually, a fragmented network emerges with a lot of sensors still alive, but they cannot route data to the sink. In such situation, a network is no more operable, and the time

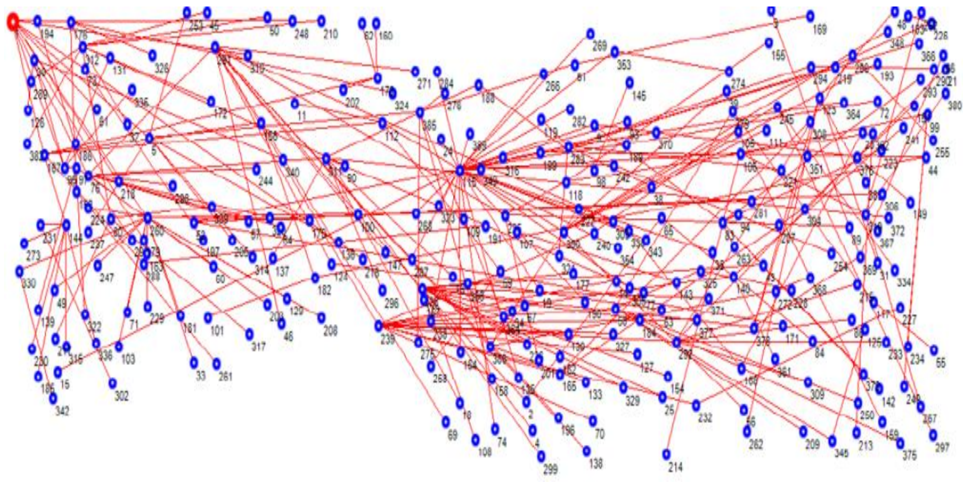

Figure 1.6: Network topology of GreenOrbs deployment

length from the deployment of the network till this phenomenon is called the lifetime of the network.

From the example of GreenOrbs, we can distinguish some key points that affect the lifetime of WSNs. First of all, we realize that the lifetime of the network can be low even if most of the sensors have sufficient battery energy left. Secondly, the underlying routing structure can significantly limit the lifetime of the network. Thirdly, the size of the network can be a vital issue for any lifetime maximization approach. In the next section, we discuss how to design effective lifetime maximization framework with the help of the data collection trees.

1.4 Motivation: Lifetime Maximization of Data Collection Trees

In this dissertation, we explore the lifetime maximization problem in the context of data collection trees [10, 16, 17]. The reasons we are interested in data collection trees are the following:

- Trees are natural choice as the routing structure for periodic monitoring applications. Once a data collection tree is defined with the sink node as the root, every node knows its parent node to forward the data. This makes the routing a lot easier than random duty cycling.

- Since it eliminates the need for random duty cycles, and it is possible to have a well defined scheduling algorithm for data collection [14]. This contributes to have predictable deadline for the sink to receive all data.
- In tree based structures, a predefined schedule for each node for data transmission also helps to mitigate the effect of interference [18].
- Tree structures are also suitable for data compression [19]. In a data collection tree, the parent nodes may act as aggregator for the children nodes. Aggregation or data compression help reduce amount of transmitted data and thus saves energy of the nodes. To have effective compression, it is important to collect data from sources that are highly correlated. In trees, the data compression can be optimized by defining the parent children relation appropriately, i.e., child nodes may select its parent based on the degree of correlation among their sensory data stream.

In this thesis, we approach the lifetime maximization problem for data collection trees from two different perspectives. The first one is load balancing, where a tree structure is derived considering traffic loads of the nodes. Secondly, we explore a data aggregation or compression framework, where we reduce the amount of data forwarded to the sink by considering compression of data streams from neighboring sensor nodes.

Consider the Figure 1.7. Here a sensor network G with 9 sensors is shown. The sink node is marked as node 0. Edges between the nodes represent the communication links. As we can see, several different data collection trees can be created from the graph representing the sensor network. Assume, each node creates one packet in each data collection round. A node forwards its own packet as well as packets from the

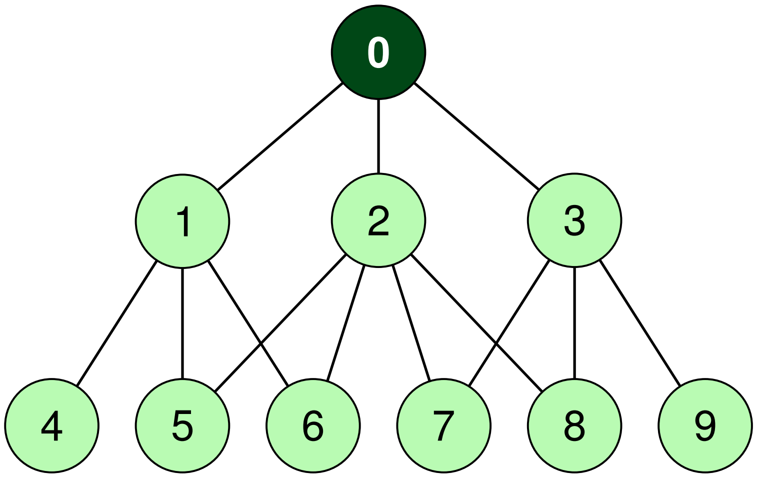

Figure 1.7: A sensor network, G

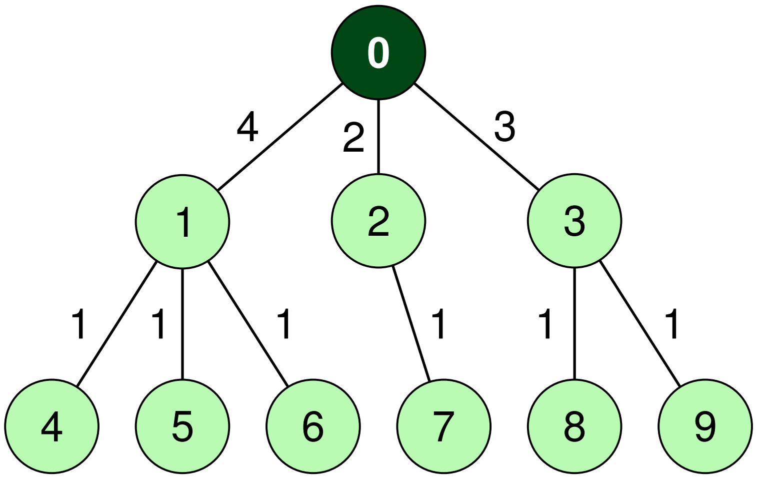

Figure 1.8: First data collection tree of G

children. For example, we can build data collection trees as shown in Figures 1.8, and 1.9 respectively. Among these two possible trees, the one in Figure 1.9 is the most balanced in terms of number of packets forwarding. Thus, in this tree, the time until the first node die would be higher than that in Figure 1.8. With this example, we see that load balancing deals with finding better topologies of a given sensor network.

On the other hand, reducing the amount of traffic through aggregation or compression of correlated data from nearby sensors can also lead to significant energy

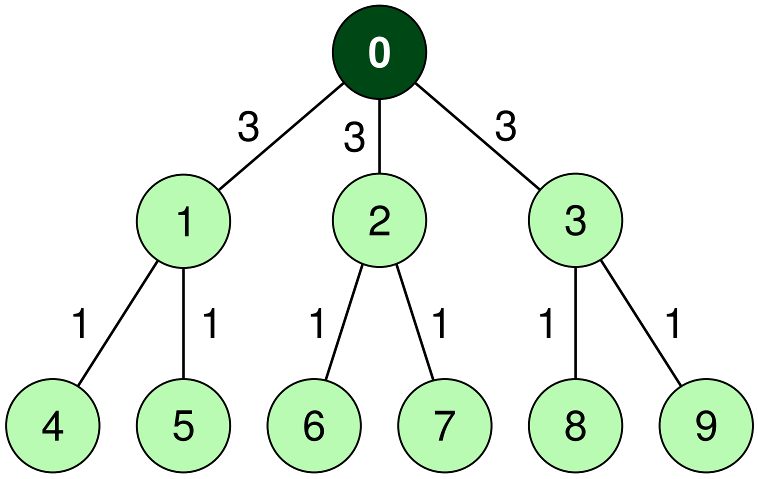

Figure 1.9: Second data collection tree of G

savings. However, compression may also cause inaccuracy when original signals are reproduced from the compressed data. Moreover, the degree of correlation among a set of neighboring sensors may change over time. These two factors make the in network compression a challenging problem. While most works on data compression focus on static correlation structure among the sensors, we developed a novel data compression famework, combined with a reinforcement learning based route selection approach to address these challenges.

1.5 Contributions

Even though there have been a lot of research on the lifetime maximization in wireless sensor networks, the context of data collection trees provide some unique challenges. In this dissertation we try to answer some of these challenges by providing two novel frameworks, namely load balancing, and data compression. The contributions of the research work presented in this dissertation are the followings:

- We propose a simple but effective framework for load balanced trees for periodic data collection in wireless sensor network. Based on our framework, we then propose a novel randomized algorithm, namely RaSMaLai, to achieve balanced trees and maximize the lifetime of the network. We provide a unique route exploration strategy that makes intelligent balance between exploration of new topologies and conforming to better topologies.

- Through extensive simulation, we show that RaSMaLai achieves higher lifetime than existing approaches with significantly lower time complexity. We also make a comprehensive study on the scalability of our approach by simulating large networks. We take the average number of forwarded packets per node, and the average number of switching per node as measures of scalability and study the effect of various network parameters like density, number of nodes, area, and the load balancing parameter.
- We then provide a distributed implementation of the proposed algorithm, called D-RaSMaLai, that incurs low energy overhead. D-RaSMaLai works in a three phase message passing work flow. We design this distributed protocol in a novel way, such that control messages are piggybacked on data messages, and optimized with message suppression when no new load information is needed to be exchanged. This saves significant amount of energy on control messages.
- We also explore the domain of data compression to tackle the lifetime maximization problem. In this regard, we propose a novel framework to build energy efficient fusion tree for a given sensor network that is suitable for periodic and continuous monitoring applications. In such applications, the sensed data can be considered as time series, and we can apply compression methods that are well known for time series. However, the novelty of our framework is that, we consider compressing multiple time series from neighboring nodes, and perform in network compression.

- Our framework is intelligent enough to discover the temporal variation in the corrleation among sensors, and adjust the compression tree accordingly. In addition, our framework also ensures that the compression process does not introduce imperfection more than a given bound while the original signals are reconstructed.
- One greatest advantage of compression framework is that, unlike other approaches, our approach does not require any a priori knowledge of input signal statistics and have the ability to trace the internal variation of the signal statistics. This is due to the fact that, the framework incorporates reinforcement learning approach to discover the correlation among signals of neighboring sensors, and it selects the aggregator nodes accordingly. The learning strategy can discover any deviation in the correlation, and can adjust the tree structure accordingly. We also provide a low cost implementation of maintaining and updating the fusion tree.

1.6 Organization of the Dissertation

The dissertation aims to present novel frameworks for lifetime maximization of tree based sensor networks. Chapter 2 discusses related works for lifetime maximization in WSNs. In Chapter 3, we introduce the RaSMaLai algorithm that maximizes lifetime through load balancing. In Chapter 4, we present the distributed implementation of RaSMaLai framework. In Chapter 5, we discuss effect of data compression on lifetime, and propose a novel framework for aggegation that takes the dynamic correlation structure among sensors. Finally, Chapter 6 summarizes the fndings and discuss the opprotunities for further research in future.

Chapter 2

Related Work

The problem of lifetime maximization of wireless sensor networks has been a prominent area of research in recent years. There has been a plethora of research works in this domain. A detailed survey on different approaches of lifetime maximization of sensor networks has been presented in [7]. Here we only mention the works that are relevant to our modelling of the problem of lifetime maximization of sensor network. We categorize them in several broad aspects.

The rest of this chapter is organized as follows. Section 2.1 discusses the aspects of lifetime from the perspective of WSN coverage. Section 2.2 presents research works on duty cycle based approaches to maximize lifetime. A discussion on load balancing is presented in Section 2.3. In Section 2.4, distributed algorithms are presented and discussed. Section 2.5 presents data compression algorithms for lifetime maximization. Finally, Section 2.6 summarizes this chapter.

2.1 Lifetime and Coverage

In [20], authors presented an approximation algorithm that maximizes sensor lifetime with coverage guarantees. In their work, a subset of sensors are activated in each time slot. The subset is selected in a way that the entire sensing region is covered. Authors of [21] also proposed a similar idea. However, instead of covering the entire region, in their work, the concept of desired sensing coverage (DSC) was used to partially cover the region in one round, and the entire region would eventually be covered in multiple rounds. In [22], authors proposed a theoretical framework for deployment of the sensors to satisfy a predefined lifetime and coverage requirements. In [23], a minimum set cover based approach was presented for optimal deployment of sensors. In [24], authors presented a maximum cover tree algorithm to solve the connected target coverage problem with lifetime maximization objective. In their work, the algorithm produces a tree based network that can approximate the theoretical maximum lifetime. In [25], a genetic algorithm was proposed to maximize lifetime with k-coverage constraint. In this work, authors utilized mobile nodes to move them to appropriate positions. In [26], authors designed a Tabu search based approach to find near optimal topology for lifetime maximization with low time complexity.

2.2 Lifetime and Duty-cycling

Duty cycle is a popular energy conservation approach in WSNs. In [6, 27], authors presented a randomized duty-cycling approach by leveraging trade-off with connection delay. In [28], authors proposed a cross-layer approach to show the effects of frequency reuse in lifetime maximization by restricting the link schedules to the class of interference-free time division multiple access. In [29], a tunable mechanism for reducing the variance of the node's duty cycle was proposed for energy harvesting in WSNs. In [30], authors established tight analytical bounds on the sleeping probabilities of nodes and on the achievable lifetime of WSNs. In [31], authors argued that for broadcasting under low duty-cycles, sensors could easily fail to cover the whole network in an acceptable timeframe. In [32], an efficient sleep schedule policy was introduced that guarantees a bounded-delay sensing coverage while maximizing network lifetime. In [33], authors presented an anycast based packet forwarding scheme, where each node opportunistically forwards a packet to the first neighboring node that wakes up among multiple candidate nodes. Authors in [34] also presented a tier-based anycast protocol and developed a new technique of improving network lifetime. A protocol named MeeCast was presented in [35]. MeeCast was shown to work well in terms of energy efficiency for sensor applications with ultra low traffic rate. The work in [36] also studied energy efficient protocol in ultra low duty cycle, and presented a novel forwarding scheme based on distributed wakeup scheduling which can guarantee bounded delay on the messages that are delivered, and can have higher delivery ratios. In [37], authors presented a novel sleep-scheduling technique called Virtual Backbone Scheduling (VBS). In VBS, traffic is only forwarded by backbone sensor nodes, and the rest of the sensor nodes turn off their radios to save energy.

2.3 Lifetime and Load Balancing

In [14], algorithms were proposed to construct degree-constrained trees and capacitated spanning trees in order to reduce the number of bottleneck nodes for scheduling purposes. In [38], an adjustable convergecast tree algorithm was presented to regulate the children of aging nodes by using localized information.

In [39], game theoretic approaches were proposed, wherein each node acts selfishly to maximize its local utility. To achieve an energy balanced tree, the energy dissipation of a node was assumed to be proportional to the size of the subtree rooted at that node. However, these approaches does not formally relate load balancing with lifetime maximization, and also lacks a consistent definition of load balanced tree.

In [40], two routing algorithms were presented based on the max-min path energy and weighted path energy. Both of these algorithms work on a state-based routing strategy, wherein a state is defined by the current energy level of the nodes.

In [41], an energy-efficient node-disjoint multipath routing algorithm was proposed, wherein multiple collision-free paths were established between a source and a sink. One of the inherent assumptions here is that overhearing nodes should not be on any route, which may not be realistic for dense deployments.

In [42], a traffic adaptive routing algorithm was proposed, wherein a node decides its next hop to forward data based on the current traffic loads of its neighbors. Selecting neighbors with lower traffic load may lead to oscillation, such that a set of nodes switch between their common neighbors at every data collection round. In our proposed work, we address this scenario and provide a strategy to achieve convergence even in the presence of such oscillating situation. Another key difference is that, these approaches fall into the category of multipath routing where nodes forward their packets through different routes and do not have a structured tree.

In [43, 44], it was shown that, in a tree-based data collection network, the lifetime maximization problem is NP-complete. The underlying energy cost model of both of their works assumed that each node performed perfect aggregation and thus transmitted only one message. In contrast, we consider a more generic scenario wherein a node does not perform aggregation and can forward a varying number of messages. In [45], the lifetime maximization was addressed by mapping it to a semi-matching problem. However, such formulation is also restricted to perfect compression and shortest path trees.

The algorithm in [46] obtains an initial loosely-balanced tree, that is then adjusted by moving subtrees. The LOCAL-OPT algorithm [47] optimizes the data collection tree by means of local information, while the maximum lifetime tree (MITT) algorithm [2] is based on a min-max-weight spanning tree. Both LOCAL-OPT and MITT share the same basic principle, namely the lifetime of a given tree can be improved by switching the parent of the node under consideration. Our proposed schemes, RaSMaLai and D-RaSMaLai, are also built on the concept of switching, but the characteristic choice of switching strategy achieves the converged state much faster and enhances the lifetime of data collection trees significantly.

2.4 Distributed Approaches for Lifetime Maximization

In [48], a distributed probabilistic load balancing approach, called Local-Wiser was presented for lifetime maximization. In that work, each node assigns a certain packet forwarding probability to each of its links to the upper level nodes. Even though Local-Wiser provably converges, it is still limited to shortest path routing. In [49], authors presented a binary search based approach to assign minimal power level to each node while maintaining the network connectivity. In [50], authors formulated the lifetime maximization problem using flow contention graph model. However, their model has a high message complexity. In [51], authors presented a regularization method that can jointly maximize the network lifetime and minimize a secondary objective like packet delay. In [52], a framework was presented that maximizes a vector of lifetime. In this model incorporates failure of nodes in series as opposed to failure of a single node.

2.5 Lifetime and Data Compression

The problem of energy efficient data compression in wireless sensor networks has been a prominent area of research in recent years. In [53], an attribute aware data aggregation scheme was presented to regulate the packets from different applications to converge spatially. In their approach, packets generated by different applications were considered completely uncorrelated, and aggregation was only performed on the packets from the same application. This ignores the spatio-temporal correlation effect of sensor data streams.

In [8], algorithms were proposed to construct compression trees with constant approximation factor. However, authors condsidered a simplified compression model where data is always compressed by a constant factor. Their model overlooks the loss of reliability of data due to compression and does not consider the varibility of correlation.

In [54], authors considered the data reliablity issue while performing compression. In their approach, they assigned different information weight on data packets from different sensors and used multiple transmission to achieve desired transmission reliability. However, their compression model is naive as it only considers full compression, where compressed data packets have the same size as the individual data packets.

In [55], authors proposed a semistructured approach where multiple shortest path trees were constructed to support large scale compression tree. Their framework achieves energy efficiency by early aggregation without incurring overhead of constructing a fixed structure. Authors in [56] also proposed semistructured and unstructured topologies to address the scheduling problem in data compression.

In [57, 58], authors proposed algorithms for routing correlated data while considering both the transmission cost and the compression cost in the energy optimization space.

The effect of compression on delay parameter has been investigated in some recent works. Authors in [59, 19] proposed a framework was provided to maximize the aggregated information that reaches the sink under deadline constraint. A major drawback of their work is that they assumed a tree structure for routing is already given. In [60], authors formulated a semi-Markov decision process to tradeoff between delay and energy efficiency.

Authors in [61] proposed the concept of compression tree for energy efficient compression. In contrast to traditional approaches, in their model the parent node broadcast its value to its children, and the compression takes place in the individual child's node. Their framework utilized the second order conditional entropy as a measure for spatial correlation.

Unlike the existing research works on compression, our model incorporates temporal correlation as well as spatial correlation in sensory data. We propose a novel approach by first considering the periodic data collected by sensors as time series. Then the energy efficient en-route compression problem reduces to finding a compression tree where parent nodes optimally compress multiple time series representing data streams of the child nodes.

2.6 Summary

In this chapter, we summarize the existing works on lifetime maximization of WSNs. We categorize the works in several broad domains, namely coverage, duty-cycling, tree based WSNs, distributed load balancing, and data compression. Our work in this dissertation encompasses the last three domains.

Chapter 3

A Randomized Algorithm for Lifetime Maximization with Load Balancing

In this chapter, we propose a load balancing approach for lifetime maximization of tree based WSN. Load balancing-based schemes explicitly attempt to organize the network topology in such a way that sensor nodes have uniform loads in terms of data forwarding. Since nodes closer to the sink have higher traffic to forward, they run out of their energy earlier. Thus, load balancing approaches are suitable to address the lifetime maximization problem [2, 3] by creating a *balanced* data collection tree. However, existing approaches in this domain often encounter three major challenges: a) oscillation, a situation where topology changes repeatedly, while trying to balance the loads; b) high time complexity; c) unsuitability for distributed implementation. We propose a novel randomized approach that efficiently addresses these challenges.

Specifically, the focus of this chapter is to maximize the lifetime of data collection trees that route raw (i.e., not aggregated) data to the sink. Here, the *lifetime* of a data collection tree is defined as the time elapsed until the first node in the network depletes all its energy [2]. Since the initial data collection trees formed in a deployed sensor network may not be balanced (in terms of the associated load), some nodes may run out of their energy long before other nodes in the tree. The major contributions of this chapter are as follows.

- We propose a novel randomized switching algorithm (called RaSMaLai) to maximize the lifetime of data collection trees based on the concept of *bounded balanced* trees. RaSMaLai exploits oscillation in a controlled fashion to explore such trees.

- Through extensive simulation, we show that RaSMaLai achieves higher lifetime than existing approaches [2, 47], with significantly lower time complexity. Specifically, in a network of N sensor nodes, each having at most Q neighbors, the time complexity of RaSMaLai is given by $O\left(N^2Q\sqrt{\log \frac{N}{\sqrt[(K+1)]{\delta}}}\right)$, where K is a constant and δ is a load balancing parameter discussed later.
- We make a comprehensive study on the scalability of our approach by simulating large networks. We take the average number of forwarded packets per node, and the average number of switching per node as measures of scalability and study the effect of various network parameters like density, number of nodes, area, and the load balancing parameter δ.

The rest of the chapter is organized as follows. Section 3.1 formally defines the problem and relates lifetime maximization to load balancing of data collection trees. The proposed algorithms are detailed and analyzed in Section 3.2, and analyzed in Section 3.3. Simulation results are presented in Section 3.4 and a summary is drawn in Section 3.5.

3.1 Problem Formulation

In this section, we formally define the lifetime maximization problem. We also introduce the concept of bounded balanced trees, and show how an optimal bounded balanced tree solves the lifetime maximization problem.

3.1.1 System Model and Assumptions

Let $G = (V, E)$ be a graph representing randomly placed sensor nodes over a monitoring area A, where: $V = \{v_0, v_1, ..., v_N\}$ denotes the set of vertices corresponding to the N sensor nodes and the sink node v_0; and E is the set of edges representing the (radio) communication links between sensors. We assume that sensors are placed densely enough so that there is no disconnected component in G. We call G as the *connectivity graph* of N sensors.

Definition 3.1 (Data Collection Tree). A *data collection tree* $T = (V_T, E_T)$ is an acyclic spanning subgraph of $G = (V, E)$ with $V_T = V$ and $E_T \subseteq E$, where v_0 is the root of T at level 0.

Let L denote the maximum level of nodes in T. When the level information l is needed, a node will be denoted as v_i^l; otherwise we will simply drop the superscript for brevity. In a rooted spanning tree T of G, nodes v_i and v_j are siblings if they have a common parent. The set of children of v_i is denoted as C_i. Let M denote the set of leaf nodes in T. There can be different paths from v_i to the sink v_0 in different data collection trees of G. We denote T^k as the k-th data collection tree of G and P_i^k as the path from v_i to v_0 in T^k. The subtree rooted at a node v_i is denoted as $T(v_i)$, while the current energy budget of v_i is referred to as e_i. We define the data reception rate R_i^c of v_i as the amount of data received from its children in a data collection round. Here a data collection round denotes the process where the sink collects data from all the sensor nodes [62, 2].

The data generation rate R_i^g of v_i is defined as the amount of data generated by v_i in a data collection round. Similarly, the transmission rate R_i^t of v_i is the amount of data transmitted by v_i in a data collection round. We define the *energy loss rate* r_i of node v_i as the amount of energy v_i spends in a data collection round. For all nodes, we denote E^t and E^c as the units of energy spent for data transmission and reception, respectively. The energy spent for data generation is assumed to be negligible [63]. Thus, the energy loss rate of v_i is given by $r_i = R_i^t E^t + R_i^c E^c$. We finally define the *load*, γ_i, of a node v_i as the ratio of r_i to e_i. Note that, in our model, the *lifetime* of a node v_i is defined as $t_i = \frac{e_i}{r_i} = \frac{1}{\gamma_i}$. In our approach, we assume that individual sensor nodes may have different initial energy budgets, and that data are forwarded without

Table 3.1: Summary of the used notation

Symbol	**Description**
$G = (V, E)$	Graph of vertices V and edges E
$T = (V_T, E_T)$	Tree of vertices $V_T = V$ and edges $E_T \subseteq E$
$T(v_i)$	Subtree of T rooted at node v_i
T^k	Tree at the k-th data collection round
C_i	Set of children of node v_i
M	Set of leaf nodes in T
P_i^k	Path from v_i to the sink in T^k
γ_i	Load of node v_i
σ_i	Load of path P_i
t_i	Lifetime of node v_i
$t(P_i^k), t(P_i^k)$	Lifetime of path P_i^k and tree T_i^k, respectively
η	Load balancing parameter of a tree
ω	Load bound of a tree
δ	Tradeoff parameter, equal to $\eta - \omega$

any aggregation. A node can generate data from its own sensing activity, and also receive data from other nodes. For simplicity, in the following we will assume that $R_i^t = R_i^c + R_i^g$ for each v_i. We list the notations in the Table 3.1.

3.1.2 Maximum Lifetime Revisited

Before proceeding further, let us provide a few definitions by recalling the concept of lifetime t_i of a node v_i from the last subsection.

Definition 3.2 (Lifetime of a path). The lifetime $t(P_i^k)$ of a path P_i^k is the minimum lifetime of all nodes on P_i^k. Formally, $t(P_i^k) = \min \left\{ t_j | v_j \in P_i^k \right\}$.

Definition 3.3 (Lifetime of a tree). The lifetime of a data collection tree T^k of G is the minimum lifetime of all paths in T^k. Formally, $t(T^k) = \min \left\{ t(P_i^k) | v_i \in V \right\}$.

In the literature, there exist several notions of lifetime in wireless sensor networks, depending on the underlying applications. In our context, *lifetime* is defined as the time until the first node runs out of energy, which is consistent with Definition 3.3.

Definition 3.4 (Path Load). In a given data collection tree T^k, the path load σ_i of a node v_i is the maximum load of all nodes along the path from v_i to v_0, where the load of v_i is $\gamma_i = \frac{r_i}{e_i}$. Formally, $\sigma_i = \max\left\{\gamma_j | v_j \in P_i^k\right\}$. If v_j is the parent of v_i on P_i^k, then the path load of v_i can be recursively defined as $\sigma_i = \max\left(\sigma_j, \gamma_i\right)$. If node v_i is at level 1, then its path load is $\sigma_i = \gamma_i$.

Let $S = \left\{T^k\right\}$ denote the set of data collection trees in G, where the size of S can be of exponential order of N, the number of sensors [64]. We now define the lifetime maximization problem (LMP) as follows:

Definition 3.5 (LMP). Given a connectivity graph $G = (V, E)$ of a sensor network, the lifetime maximization problem is to find $T^k \in S$ such that $t(T^k) \geq t(T^j)$, $\forall T^j \in S$.

In the following, we introduce the concept of load balanced data collection tree. Let $\left\{\sigma_i^M\right\}$ denote the set of path loads of all leaf nodes, i.e., $\{\sigma_i^M\} = \{\sigma_i | v_i \in M\}$.

Definition 3.6 (Load balanced tree). A tree T^k is η-load balanced, if there exists an $\eta \in R$ (the set of real numbers) such that $\max\left\{\sigma_i^M\right\} = \eta$, $\forall\, v_i \in M$. An η-load balanced tree T^k is optimally balanced if, for any η'-load balanced T^m, it holds $\eta \leq \eta'$.

We derive the first theorem as follows.

Theorem 3.1. If T^i is an optimally balanced tree, then $t(T^i) \geq t(T^j)$ for any $T^j \in S$. In other words, T^i provides the maximum lifetime for the entire network.

Proof 1 (By contradiction). Suppose T^i is an optimal η-balanced tree and there exists some non-optimal tree T^j that provides the maximum lifetime, i.e., $t(T^j) > t(T^i)$. Let T^j be η'-balanced. Let v_x lie on the path P_x^j in T^j such that P_x^j has the minimum lifetime. On this path in T^j, let any node v_u have the least lifetime. So $t\left(T^j\right) = t(P_x^j) = t_u$. Since v_u has the least lifetime in P_x^j (hence in T^j), the load γ_u of v_u is the maximum along the path P_x^j (hence in T^j). Thus, the path load of v_x is $\sigma_x = \gamma_u$. Also, let v_w have the least lifetime on the path P_y^i with the minimum lifetime in T^i. Similarly, we have $t\left(T^i\right) = t\left(P_y^i\right) = t_w$ and $\sigma_y = \gamma_w$. Now, t_u/t_w is inversely proportional to γ_u/γ_w, and hence to σ_x/σ_y, where σ_x and σ_y are the maximum path loads in T^j and T^i respectively. Now by definition, $\sigma_y = \eta$, $\sigma_x = \eta'$ and $\eta \leq \eta'$. We have, $t_u > t_w \Rightarrow \sigma_x < \sigma_y \Rightarrow \eta' < \eta$. This leads to a contradiction since T^j is not optimal. Hence, T^i provides the maximum lifetime for the network.

Definition 3.7 (δ-bounded balanced tree). A tree is δ-bounded balanced, with $\delta = \eta - \omega$, if there is a pair $(\eta, \omega) \in R^2$ such that $\max\left\{\sigma_i^M\right\} = \eta$, $\min\left\{\sigma_i^M\right\} = \omega$, and $\omega \leq \eta$.

The δ-bounded balanced tree is the key concept behind our formulation of the load balancing problem. Note that an η-load balanced tree can be expressed as a special case of δ-bounded balanced tree where $\delta = \eta - \omega$ for $0 \leq \omega \leq \eta$. Hence, if we can find an optimal δ-bounded balanced tree, we can also find an optimal η-balanced tree. Note that an η-load balanced tree can be expressed as a special case of (η, ω)-bounded balanced tree with $0 \leq \omega \leq \eta$.

Theorem 3.2. A data collection tree T^j is optimally (η, ω)-bounded balanced if and only if $\delta = \eta - \omega$ is minimum.

Proof 2. (Necessity). Let T^j be optimally (η, ω)-bounded balanced. For any (η', ω')-bounded balanced tree T^k, we have $\delta' = \eta' - \omega' \geq \eta - \omega' \geq \eta - \omega \geq \delta$. Thus δ is minimum in T^j if T^j is an optimally (η, ω)-bounded balanced tree.

(Sufficiency). Now we prove that if T^j has minimum δ then T^j is optimally (η, ω)-bounded balanced. We reason by contradiction. Let T^j have minimum δ, but there is some tree T^k with $\delta' > \delta$, that is optimally balanced. If T^k is optimally (η', ω')-bounded balanced, then $\eta' < \eta$ and $\omega' > \omega$. Thus, $\delta' > \delta \Rightarrow \eta' - \omega' > \eta - \omega \Rightarrow \eta' - \eta > \omega' - \omega$. However, $\eta' - \eta < 0$ and $\omega' - \omega > 0$, and we reach to a contradiction.

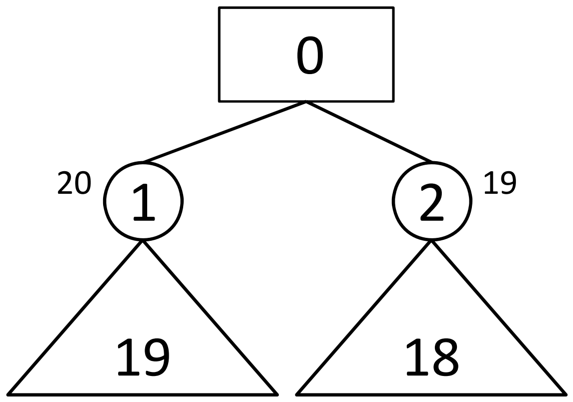

Figure 3.1: Comparison based on η and δ for T^4

Thus, if δ is minimum then there can not be any such (η', ω')-bounded balanced tree with $\eta' < \eta$ and $\omega' > \omega$. Hence, T^j is optimally (η, ω)-bounded balanced.

We have proved that a tree T^j is optimally δ-bounded balanced if and only if there is no other δ'-bounded balanced tree in the connectivity graph G such that $\delta' < \delta$.

3.1.3 Rationale for δ-Bounded Balanced Trees

According to our definition of lifetime, one may argue that it is sufficient to find an η-load balanced tree with minimum η. However, for a given tree the value of δ provides more insight about how much the tree is actually balanced. The mere value of η only tells about the maximum path load of a tree, but provides no information about how the other path loads are distributed with respect to η.

In Figures 3.1 and 3.2, we compare the η and δ values of data collection trees T^4 and T^5 of two different connectivity graphs. The triangles denote the subtrees rooted under respective nodes. The numbers inside the triangles denote the size of the subtrees (excluding the root of the subtree). Let $e_i = 1$, $R_i^g = 1$, $E^t = 1$ and $E^c = 1$ for each node of T^4 and T^5. In this settings, the load of each node,

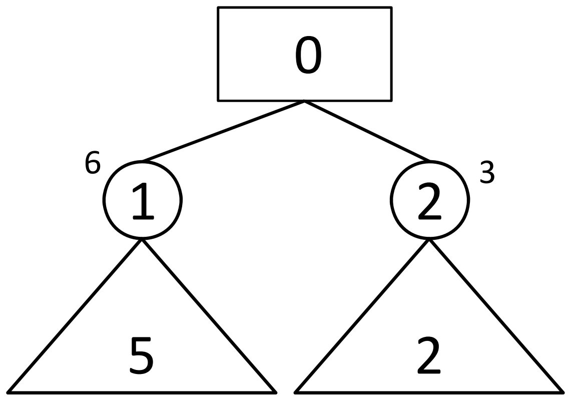

Figure 3.2: Comparison based on η and δ for T^5

$\gamma_i = \frac{E^t R_i^t + E^c R_i^c}{e_i} = R_i^t + R_i^c$. We also assume that the nodes have a uniform energy budget in this example. In this example, the nodes at level 1 are the bottleneck nodes, i.e., the path loads of the leaf nodes are given by the loads of the nodes at level 1. We mark the R_i^t values of the nodes (at level 1) beside the vertices representing them. We see that for T^4 it is $\eta = 20 + 19 = 39$, while for T^5, it is $\eta = 6 + 5 = 11$. By looking at the η values, it may seem that T^4 is more unbalanced than T^5. However, the δ value of T^5 is 6, which is greater than the δ value of T^4 (i.e., 2). Thus, according to the δ values, T^5 is more unbalanced than T^4, which is indeed the case.

3.1.4 Bounded Load Balanced Tree Problem

Let us now define the bounded load balanced tree problem on the basis of the δ-bounded tree.

Definition 3.8 (B-LBTP). For a given connectivity graph G, the bounded load balanced tree problem is to find a data collection tree T^j of G such that T^j is δ-bounded balanced and $\delta \leq \delta'$ for any δ'-bounded balanced tree of G.

Theorem 3.3. The B-LBTP is NP-complete.

Proof 3. As detailed in Theorem 3.2, the B-LBTP solves the lifetime maximization problem. Since the lifetime maximization of data collection trees is known to be NP-complete [47], the B-LBTP is also NP-complete.

In the next section, we describe in details our algorithm that achieves balanced trees with very low running time.

3.2 RaSMaLai Algorithm

The path loads of a data collection tree can be balanced by exploiting the concept of tree transformation and switching. For instance, let us refer to Figures 3.3, 3.4, and 3.5 as an example. We can convert T^1 to T^2 by dropping the edge (v_2, v_7) and adding the edge (v_7, v_3). By recalling that a subtree rooted at v_i is denoted as $T(v_i)$, we can say that v_7 is switched from $T(v_2)$ to $T(v_3)$ by this operation. Similarly, we can switch v_6 from $T(v_1)$ to $T(v_2)$ to obtain T^3 from T^1. For simplicity, we again assume that $e_i = 1$, $R_i^g = 1$ for all nodes in the figure. In T^3, each node at level 1 has $R_i^c = 2$ and $R_i^t = 3$. If $E^t = E^c = 1$, the loads of v_1, v_2 and v_3 are five units each. By Definition 3.4, the path load of all the leaf nodes is also five units. Thus in T^3, the path loads are perfectly balanced.

In the following, we propose our randomized algorithm, RaSMaLai, which balances the path loads of nodes in the network through switching.

3.2.1 Switching Concepts

Let G be a given connectivity graph and T^k be an arbitrary data collection tree of G. Let v_a and v_b be two leaf nodes in T^k and assume their path loads are σ_a and σ_b, respectively. Since δ denotes the maximum allowable difference in the path loads among the leaf nodes, we say the node v_a is in an *unbalanced* condition with v_b if $\sigma_a - \sigma_b > \delta$. Conversely, when $|\sigma_a - \sigma_b| \leq \delta$, we say v_a and v_b are in a *balanced*

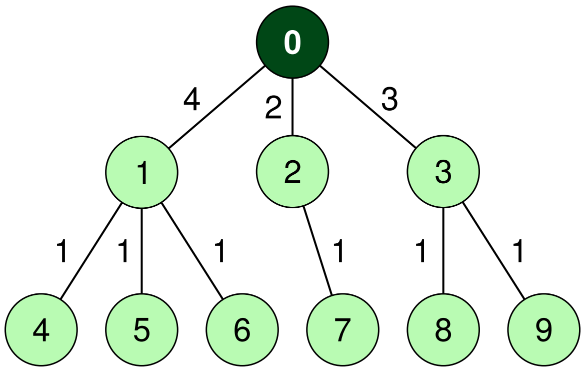

Figure 3.3: Data collection tree, T^1

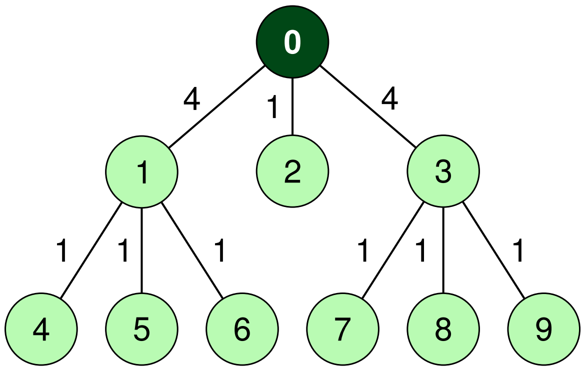

Figure 3.4: Data collection tree, T^2

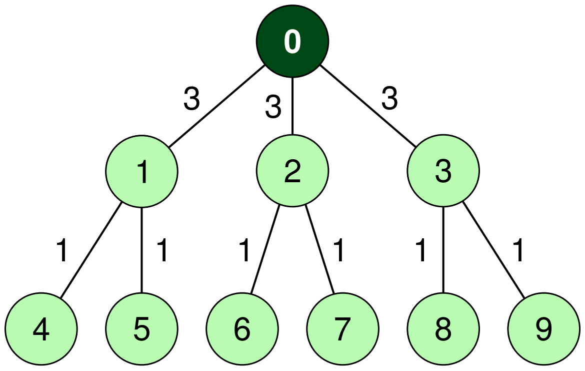

Figure 3.5: Data collection tree, T^3

condition with each other. Let v_c and v_d be the nodes with the highest load on the paths of v_a and v_b, respectively (i.e., v_c and v_d are the bottleneck nodes along these paths). According to Definition 3.4, $\sigma_a = \gamma_c$ and $\sigma_b = \gamma_d$. Let v_x be a descendant node of v_c. Then v_x is a switchable node between v_c and v_d if it has an alternate path to v_0 in G through node v_d. If $\sigma_b < \sigma_a$, then v_x is switched with a certain probability to another parent, so that the related data can be forwarded along the alternate path. We say that v_x is switched from $T(v_c)$ to $T(v_d)$. The probability that v_x is switched is called the *switching probability*.

There may be multiple leaf nodes who are in unbalanced condition. Precisely, when $\max\left\{\sigma_a^M\right\} - \min\left\{\sigma_a^M\right\} \leq \delta$, all leaf nodes are in a balanced condition. Conversely, when $\max\left\{\sigma_a^M\right\} - \min\left\{\sigma_a^M\right\} > \delta$, at least one leaf is unbalanced.

In RaSMaLai, if there is at least one unbalanced leaf node, the node with the highest load (contributing to the maximum path load for some leaf) is selected to initiate the switching of its descendants. Let v_c be the node with the highest load in a given tree. If a descendant v_x of v_c has an alternate path (not including v_c), RaSMaLai randomly decides whether to switch it or not. If v_x does not have an alternate path or is not chosen for switching, the children of v_x are considered in turn.

Observe that switching helps v_c to lower its load γ_c, since v_c does not have to forward messages for the descendants which have switched to alternate paths. However, after switching some descendants from $T(v_c)$ to some $T(v_d)$, the load γ_d of v_d increases. After a few rounds of switching from v_c, the load γ_d of v_d may become greater than γ_c by more than the allowable threshold, δ. Consequently, the path load of leaf nodes of $T(v_d)$ may exceed the path load of leaf nodes of $T(v_c)$ by more than δ. In such a case, we say that an *oscillation* has occurred and v_d has oscillated.

We define an iteration of RaSMaLai as a *switching step*. Let $\gamma_i(h)$ and $\sigma_i(h)$ denote the load and the path load of a node v_i at the end of the h-th switching step.

Definition 3.9 (Oscillation). A node v_d oscillates at the $(h+1)$-th step if, after switching some descendants from $T(v_c)$, the load γ_d of v_d becomes the highest among all nodes, and a leaf node v_a of $T(v_d)$ is in an unbalanced condition with a leaf node v_b of $T(v_c)$. In other words, $\sigma_a(h+1) - \sigma_b(h+1) > \delta$.

Since $\gamma_d(h+1)$ is the highest load at $(h+1)$-th step, $\sigma_a(h+1) = \max\{\gamma_i | v_i$ is on the path of $v_a\} = \gamma_d(h+1)$. If $\gamma_d(h)$ is the highest load at the h-th step, then v_d itself is selected for switching of its descendants. Hence, at the end of $(h+1)$-th round, we have $\gamma_d(h+1) \leq \gamma_d(h)$. In this case, if no other node oscillates, $\gamma_d(h+1)$ may still remain the highest load in $(h+1)$-th step. However, in such a scenario, we do not say that v_d oscillates, even though $\gamma_d(h+1)$ is the highest load in the current round. We make such distinction because the switching of $(h+1)$-th step did not actually increase the load of v_d.

3.2.2 Details of RaSMaLai Algorithm

RaSMaLai consists of three major functions: Switch, FindPotentialParents, and UpdateTree. Switch is the core of RaSMaLai, while FindPotentialParents is used to select suitable parents for a node when it is selected for switching.

Algorithm 1: Switch(T)

```
1  Initialize $(\gamma_i, \sigma_i)$ for each $v_i \in V$;
2  Set $\beta_i \leftarrow 0$ and $p_i \leftarrow \frac{1}{2}$ for each $v_i \in V$;
3  Let $v_a$ be the node with highest load;
4  while $\beta_a \leq \beta_{max}$ do
5      if $\max\{\sigma_i^M\} - \min\{\sigma_i^M\} \leq \delta$ then Return $T$;
6      else
7          Set $\alpha \leftarrow C_a$; Increase $\beta_a$ by 1;
8          while $\alpha \neq \emptyset$ do
9              Remove node $v_j$ from $\alpha$ in FIFO order;
10             $W \leftarrow$FindPotentialParents($G, v_j$);
11             if $W = \emptyset$ then $\alpha \leftarrow \alpha \cup C_j$;
12             else
13                 if SwitchingDecision($p_j$) then
14                     Uniformly select a node from $W$ to be the new parent of
                       $v_j$;
15                     Set $p_j \leftarrow \frac{p_j}{2}$;
16                 else $\alpha \leftarrow \alpha \cup C_j$;
17         UpdateTree($T$);
18         Select $v_a$ as the node with the highest load;
19 Return $T$;
```

Finally, UpdateTree updates the loads and the path loads of the nodes. Algorithm 1 describes Switch, in which β_i counts how many times v_i is selected as the node with the highest load. Furthermore, β_{max} denotes the maximum number of times an individual node can be selected for switching. We will later discuss how to adjust β_{max} so that convergence is reached. Finally, p_i denotes the switching probability. The loads and the path loads for all nodes in a given tree T are initialized first. The initial count on the number of switching and the initial switching probability of all nodes are set to 0 and $\frac{1}{2}$ respectively (lines 1–2). Then v_a, the node with the highest load, is selected (line 3). The switching procedure is continued until some node has been selected for β_{max} times (line 4). In such a case, the while loop terminates and the updated tree is returned (line 19). Within the loop, the tree is returned if the δ-bounded condition is reached. Otherwise, the children of v_a are inserted in the queue α (line 7) and the count for v_a is updated. The second loop runs until the queue becomes empty (line 8). At each step, a node v_j from the queue is removed and a list W is populated with its potential parents (lines 9–10). If v_j has no potential parent, then the descendants of v_j are added to the queue and considered for switching in subsequent rounds (line 11). The selection of potential parents for v_j will be explained later in this subsection. If v_j has some potential parents it can switch to, then a random decision is made based on its current switching probability p_j through the SwitchingDecision function. If the outcome of the decision is to switch, a node is chosen with uniform probability from the list W to be the new parent for v_j. Otherwise, v_j remains with its current parent and the descendants of v_j are added to the queue for subsequent consideration (lines 13–16). Note that, when v_j is switched, its descendants are not considered for switching, since the entire subtree $T(v_j)$ forwards data through the new parent after v_j has switched. When the queue becomes empty, the switching of the descendants of v_a is completed. The tree is updated with new values for loads and path loads of the nodes and in the next round, the node with the highest load is selected (lines 17–18).

Algorithm 2: FindPotentialParents(G, v_j)

```
1 for $\forall v_i$ that are neighbors of $v_j$ in $G$ do
2     if $\sigma_j - \sigma_i > \delta$ then $W \leftarrow W \cup \{v_i\}$;
3 Return $W$;
```

Algorithm 3: UpdateTree(T)

```
1 Compute $\gamma_i$ for all $v_i \in V$ in a bottom-up traversal on $T$;
2 Update $\sigma_i$ for all $v_i \in V$ in a top-down traversal on $T$;
```

FindPotentialParents, illustrated in Algorithm 2, returns the list of potential parents of a given node v_j. Specifically, a neighbor v_i of v_j is added to the related potential parent list if it has a path load that is lower than that of v_j by more than δ. After each switching round, UpdateTree, shown in Algorithm 3, obtains the new values of (γ_i, σ_i) for each node v_i in the tree.

3.2.3 Illustrative Example

We illustrate the flow of the RaSMaLai algorithm with the help of Figures 3.6 through 3.10. Let us assume that, for a given connectivity graph G, the h-th iteration of the algorithm produces the tree in Figure 3.6. Let us also assume that the energy budget $e_5 = 0.5$ for node v_5 and for all other nodes it is 1. Each node produces 1 unit of data in each data collection round, with transmission and reception costs of 1 unit. In the figure, solid lines indicate the edges used in the current tree, while the dotted lines represent potential edges that can be used to transform the current tree. We mark node v_i with (γ_i, σ_i) to denote its load and path load in the current tree. Thus the first value in the parenthesis denotes the load of the node, and the second value denotes the path load of respective nodes. In this example, we set $\delta = 2$. At each

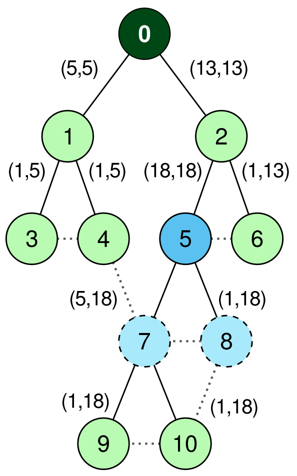

Figure 3.6: A tree at the h-th iteration

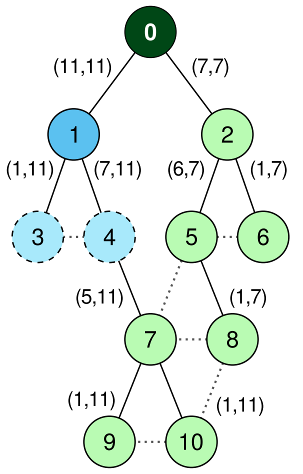

Figure 3.7: Node 7 is switched

iteration, the node with the highest load is drawn with a dark blue background, while its descendants added to α are drawn with a light blue background and a dashed border. The initial switching probability for each node is $\frac{1}{2}$. Thus v_5 is marked as the one with the highest load and consequently, its children v_7 and v_8 are added to α. Although v_7 has both v_4 and v_8 as its neighbors, the path load of v_8 is the same as that of v_7. But v_4 has a path load that is lower than the path load of v_7 by more than δ. Thus, v_4 is added to the potential parent list of v_7. On the other hand, no neighbor of v_8 offers a lower path load. Hence, the potential parent list for v_8 is empty.

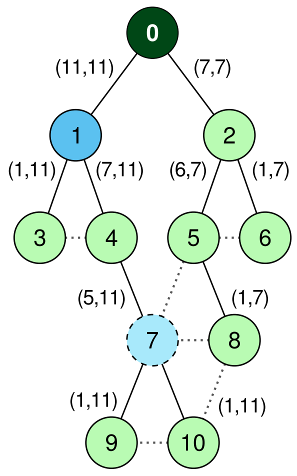

Figure 3.8: Node 7 is selected

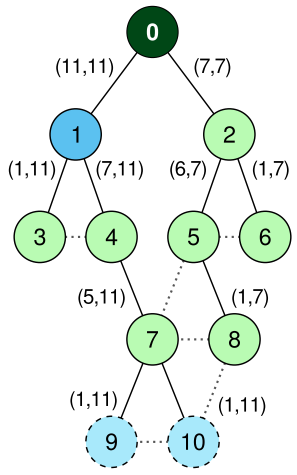

Figure 3.9: Nodes 9 and 10 are selected

Let us assume that v_7 was never switched in previous iterations and, thus, has a switching probability of $\frac{1}{2}$. Furthermore, assume that the decision in the current iteration is to switch. In this example, the potential parent list of v_7 contains only v_4. In case of more than one choice, a node would be uniformly selected from such a list. After v_7 is switched to v_4, its children are not added to α which becomes empty. Since v_7 is switched once, its subsequent switching probability becomes $\frac{1}{4}$. Finally, the function UpdateTree is called upon. Figure 3.7 shows the new tree with the updated values of (γ_i, σ_i). After switching of v_7, node v_1 oscillates and is marked

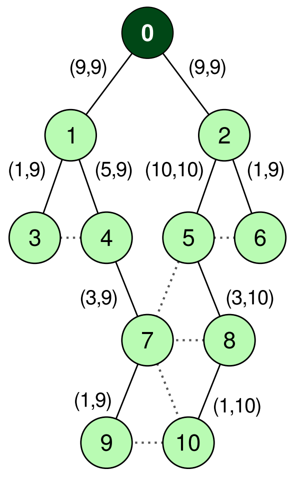

Figure 3.10: Node 10 is switched

as the node with the highest load in the current iteration. As in the previous case, the children of v_1 (i.e., v_3 and v_4) are added to α. Note that both children have empty potential parent list and that v_3 is a leaf node. Thus, in Figure 3.8, v_7 is added to α as the child of v_4. Now, v_7 has a lower switching probability, so let us assume that it is not selected in this round. Consequently, v_9 and v_{10} are added to α in Figure 3.9. The potential parent list for v_9 is empty. v_{10} has v_8 as its potential parent and it is switched to v_8 according to its switching probability p_{10}. Figure 3.10 shows the tree when v_{10} is indeed switched to v_8. Note that in this tree $\max\{\sigma_i^M\} - \min\{\sigma_i^M\} \leq \delta$.

In the above example, v_1 oscillates and eventually the algorithm finds the final tree that is more balanced. Thus, oscillation helps to explore balanced trees that may never be found without it. However, oscillation needs to be controlled to ensure convergence. We discuss further how controlled oscillation leads to a non-myopic strategy for our algorithm.

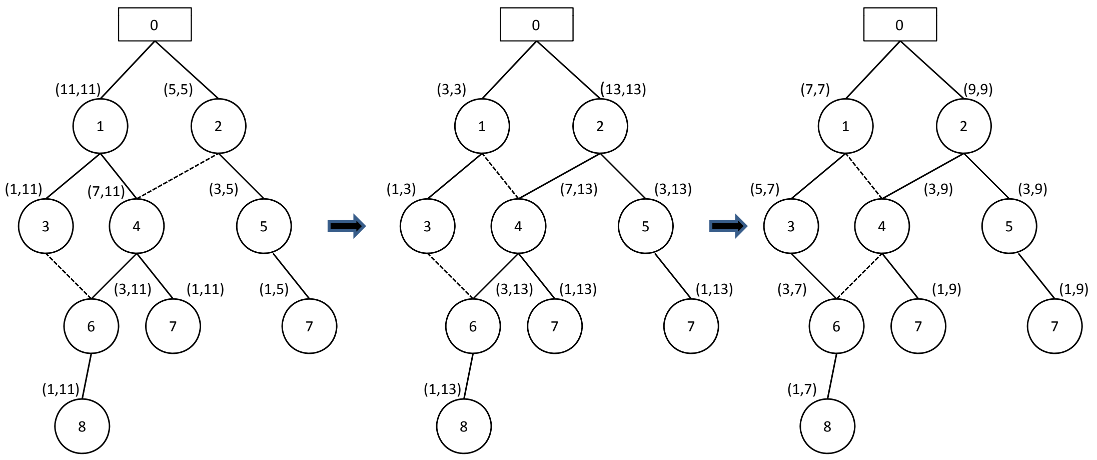

Figure 3.11: Local and global optima

3.2.4 Non-myopic Strategy of RaSMaLai

In RaSMaLai, when we consider neighbors of a node v_i to be its potential parent, we compare the path loads rather than the load values. The neighbor with the minimum load may have a node with much higher load along its path to the sink. Therefore, choosing the neighbor with the minimum load as parent may not have the desired impact on balancing the path loads. So, the loads of the neighbors cannot be treated as the only metric to take switching decision. Instead the path load of the neighbors can be used as a suitable metric to choose the potential parent. For instance, in Figure 3.7, it may seem beneficial to switch node 4 to node 3. However, this switching does not help node 1 to lower its load.

Since oscillation is allowed in RaSMaLai, it is possible that sometimes the difference between the maximum and the minimum path loads increases from the initial topology. However, such intermediate *bad* topologies may lead to better results in subsequent iterations. This may appear counter intuitive, so let us consider the example in Figure 3.11. Here, three different trees are shown for a given connectivity graph. When node 4 is switched to node 2, the tree in the middle is obtained. As a result of this switching, node 2 oscillates and the difference between the maximum and the minimum path load increases. However, in RaSMaLai, there is a non-zero probability of switching node 6 to node 3 to obtain the rightmost tree. In this tree, the difference is decreased and a tree is obtained which is better balanced than the initial one. Note that, the δ values of the left, middle and right tree are 6, 10, and 2, respectively. If oscillations were not allowed, RaSMaLai would be too conservative to leave a local optima and we would never be able to explore better topologies in subsequent iterations.

3.3 Analysis of RaSMaLai

In this section, we present the time complexity of RaSMaLai. Let $d(h)$ be the difference between the maximum and the minimum path load after h iterations of the algorithm. We express the time complexity as the number of iterations required for $d(h)$ to converge, i.e., $d(h)$ approaches towards the given threshold δ. Note that, if $d(h)$ is convergent, then there must be some value of h such that $d(h+1) \leq d(h)$ always holds. On the other hand, if $d(h+1) > d(h)$ holds for arbitrary h, it follows that, after running the algorithm for an infinite number of iterations, $d(h)$ is not guaranteed to converge. Thus, we should set the loop counter β_{max} to such a value that we achieve $d(\beta_{max}+1) \leq d(\beta_{max})$ with high probability. We present the following theorem which is the core of the time complexity of RaSMaLai.

Theorem 3.4. When $p_{h+1} = \frac{p_h}{2}$, with high probability, a node participates in oscillation $\Theta(\sqrt{\log \frac{N}{\sqrt[(K+1)]{\delta}}})$ times, where K is an arbitrary large constant.

Proof. Let n be the number of switchable nodes. Assume after h iterations of RaSMaLai, the maximum number of siwtching by any node is i. That is no node has switched more than i times. Note that, if a node has to switch again and again, it must participate in oscillation repeatedly. In other words, a node can participate in oscillation i times only if it has switched i times. Let v_a be the current bottleneck node and S_i be the set of nodes in $T(v_a)$ that have switched i times. We are interested to bound i so that a node in S_i does not switch again with high probability.

Note that, the probability of a node to be in S_i is atmost $1/2^{\frac{i(i+1)}{2}}$, since it has to switch from 1 to i times and some of them may not be in $T(v_a)$ after h-th iteration. Thus, $|S_i| \leq n/2^{\frac{i(i+1)}{2}}$. Let, N_{S_i} be the random variable denoting the number of nodes in S_i switch for $i+1$-th times when v_a is selected for switching. Assume, ϵ is an arbitrarily small number and close to 0. If $E[N_{S_i}] < \epsilon$, then the avaerage number of nodes that switch $i+1$-th times is arbitraryly close to 0. So in such case, we can say that, with high probability no node switches more than i times. We now bound i for which $E[N_{S_i}] \geq \epsilon$ holds. It is $E[N_{S_i}] = |S_i|/2^{(i+1)} \leq n/2^{(i+1)+\frac{i(i+1)}{2}} \leq n/2^{\frac{(i+1)^2}{2}}$. Thus, $E[N_{S_i}] \geq \epsilon$, hence $n/2^{\frac{(i+1)^2}{2}} \geq \epsilon$ and also $i+1 \leq \sqrt{2\log\frac{n}{\epsilon}}$. Let K be an arbitrarily large constant, such that by setting $\epsilon = \delta/N^K$, we can make ϵ arbitrarily close to 0. Since, $n \leq N$, we have that $i+1 \leq \sqrt{2(K+1)\log\frac{N}{\sqrt[(K+1)]{\delta}}}$. This implies that, when $i = \Omega\left(\sqrt{\log\frac{N}{\sqrt[(K+1)]{\delta}}}\right)$, we have $E[N_{S_i}] < \epsilon$. But now, with high probability, no node switches again. Hence the maximum number of switches by any node i becomes bounded by $O\left(\sqrt{\log\frac{N}{\sqrt[(K+1)]{\delta}}}\right)$. Combining both bounds, we get, $i = \Theta\left(\sqrt{\log\frac{N}{\sqrt[(K+1)]{\delta}}}\right)$. □

Using the above theorem we can deduce the β_{max} used in our algorithm.

Theorem 3.5. When $\beta_{max} = O(N\sqrt{\log\frac{N}{\sqrt[(K+1)]{\delta}}})$, with high probability, we have $d(\beta_{max}+1) \leq d(\beta_{max})$.

Proof. From 3.4, we get the bound on i, the maximum number of oscillation a node can participate. Since there can be atmost $O(N)$ switchable nodes, there can be atmost $O(Ni)$ oscillations. Thus, when $\beta_{max} = O(N\sqrt{\log \frac{N}{(K+1)\sqrt{\delta}}})$, with high probability, there is no more oscillation and RaSMaLai converges.

□

Let Q be the maximum number of neighbors of a node in G. Using 3.5, we can determine the time complexity of RaSMaLai as follows.

Theorem 3.6. The running time of RaSMaLai is $O(N^2Q\sqrt{\log \frac{N}{(K+1)\sqrt{\delta}}})$.

Proof. If all nodes periodically participate in oscillation, the loop on line 4 of the Switch function runs at most $N\beta_{max}$ times. From the previous discussion, we know that $O(N\sqrt{\log \frac{N}{(K+1)\sqrt{\delta}}})$ iterations are needed for $d(h)$ to converge. Hence, the loop runs $O(N^2\sqrt{\log \frac{N}{(K+1)\sqrt{\delta}}})$ times. Lines 1–3 of the Switch function can be performed in $O(N)$ time. The calculation of $\max\{\sigma_i^M\}$ and $\min\{\sigma_i^M\}$ is done during the initialization and afterwards by the UpdateTree function. As a consequence, lines 5–5 require $O(1)$ time. The initialization of α takes $O(C_{max})$ time, where C_{max} is the maximum number of children of any node. The second loop on line 8 can run for $O(N)$ times, since $T(v_a)$ can have at most N descendants. On line 10, the FindPotentialParents function takes $O(Q)$ times, where Q is the maximum number of neighbors of a node in the connectivity graph. Updating α on lines 16 and 11 takes $O(C_{max})$ time. UpdateTree on line 8 takes $O(N)$ time. Since $C_{max} \leq Q$ and all other operations inside the inner loop take $O(1)$ time, the running time for operations from line 5 to line 18 is dominated by $O(NQ)$. Hence, the total time complexity of RaSMaLai is $O(N^2Q\sqrt{\log \frac{N}{(K+1)\sqrt{\delta}}})$.

□

Note that, the running time of MITT [2] is $O(N^3E)$ and the running time of LOCAL-OPT [47] is atleast $O(NEd)$ where d is the diameter of the network. Some of the constants in the above expressions are ignored since they do not have any impact on the running time asymptotically. Since, $E = \Omega(N)$ and $Q = O(N)$ for any connected network, running time of RaSMaLai is asymptotically lower than that of MITT. For LOCAL-OPT, we can analyze it by expressing E in terms of Q, and assume the upper bound for d, i.e, $E = O(NQ)$ and $d = O(N)$. Putting these bounds yield $O(N^3Q)$ for LOCAL-OPT, which is asymptotically greater than the time complexity of RaSMaLai.

3.4 Performance Evaluation

In this section, we evaluate the performance of our algorithm, RaSMaLai through simulation experiments.

3.4.1 Simulation Setup

We simulated RaSMaLai in MATLAB, under the same conditions as for the MITT scheme [2]. Specifically, we considered a deployment wherein sensor nodes are randomly placed within a square area of 100 m $\times$ 100 m. The number of N sensors was varied from 100 to 400. Each sensor was randomly assigned an initial energy between 0.5 to 1 Joule (J), and the radio transmission range was set to 25 m. We assumed the energy required to receive a message was 50 nJ/bit and the energy required to transmit a message was 100 nJ/bit. The message size was 16 bytes. For the distributed implementation, the size of control messages was assumed to be 10 bytes. Each data point of simulation was averaged over 200 runs, and scaled between 0 and 1.

We considered two scenarios associated with different locations of the sink in the network. In Scenario 1, the sink was placed in the middle of the deployment area, namely, at the (50 m, 50 m) coordinate. In Scenario 2, the sink was placed at one side of the deployment area, specifically, at the (100 m, 50 m) coordinate. We then evaluated the following metrics:

- *lifetime*, measured as the number of data collection rounds until the first node runs out of energy;
- *runtime*, measured as the number of iterations required for convergence;
- *energy expenditure*, measured as the amount of energy spent to transmit and recieve the control packets in the distributed implementation.

We compared our results with the following data collection schemes: a) MITT [2]; b) LOCAL-OPT [47]; c) a random shortest path tree (RST); d) a degree-constrained tree (DCT). For DCT, the degree was set to 5 after some preliminary simulations performed to ensure that a connected tree is obtained without overloading the nodes. Finally, we also study the scalability of our approach in terms of average number of switching and number of packets forwarded by the nodes and compare it with our analysis.

3.4.2 Impact of δ on Convergence

The choice of δ, the maximum allowable difference among the path loads of leaf nodes, governs the quality of the convergence, i.e., the lifetime of the tree produced at the converged point. Figure 3.12 shows the lifetime achieved by the tree at the converged point from a network of $N = 400$ nodes in Scenario 1, when δ is varied from 1 to 10 with step size of 0.5. We observe that, for $\delta = 10$, the lifetime is lower. As δ is decreased, trees with higher lifetime are obtained, since they tend to get more balanced. However, for $\delta < 4.5$, the value of lifetime sharply decreases. This suggests that there may be some threshold for δ beyond which the tree can not be balanced.

It may appear that choosing δ as low as possible results in a more balanced tree. However, this is not necessarily true, since such a low value may not be realistic.

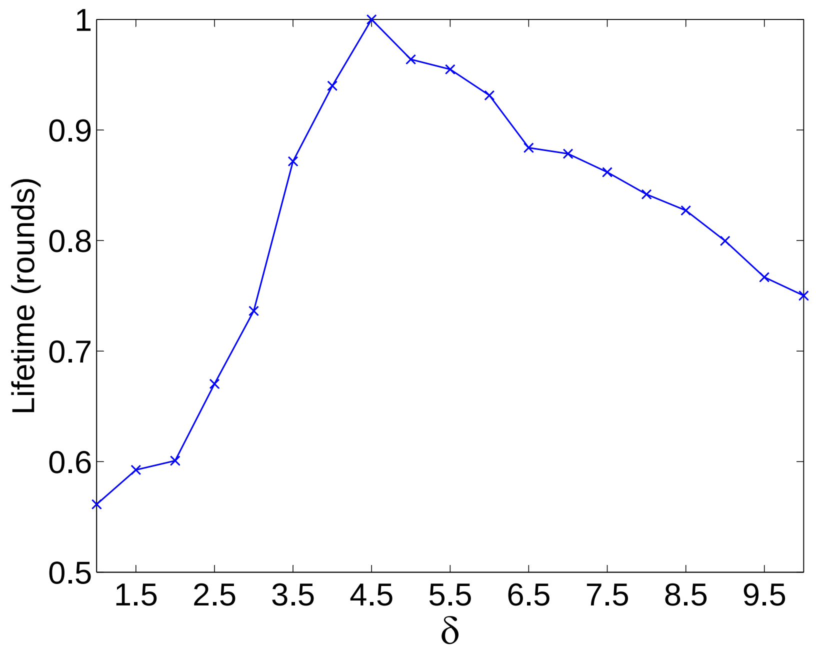

Figure 3.12: Impact of δ on lifetime in scenario 1

In fact, it may be topologically impossible to achieve an arbitrarily low δ-bounded balanced tree for a given connectivity graph representing the sensor network. When δ is too low, the RaSMaLai algorithm will cause more switching, trying to find a more balanced tree which may not actually exist. Thus more oscillations will occur and diminish the quality of the convergence state (i.e., the lifetime of the final tree). This is not desirable, especially for the distributed version.

Recall also that it is not possible to know the optimum value of δ for a random connectivity graph, which would otherwise solve the decision version of the B-LBTP problem in polynomial time. As a consequence, we need to learn the appropriate ranges of δ for different network configurations. To tackle this issue, we employed a training phase where we generated 300 connectivity graphs for networks of different sizes and different sink locations. We used 1/3 of the generated graphs as training inputs to RaSMaLai with varying δ values and measured their lifetimes. From our

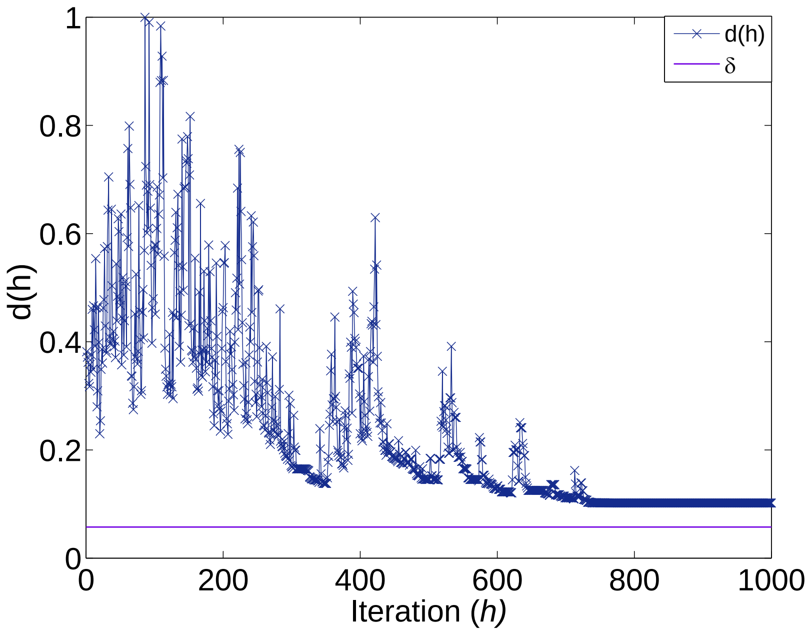

Figure 3.13: Transient behavior

experiments, we observed that RaSMaLai produces the best results when δ was varied between 3-7 and 9-14 for Scenarios 1 and 2, respectively. For test inputs, we set $\delta = 5$ for Scenario 1, and set $\delta = 12$ for Scenario 2, for both RaSMaLai.

We have already mentioned that oscillation may occur in RaSMaLai. However, proper choices of δ and β_{max} will allow RaSMaLai to converge eventually. This property of RaSMaLai is apparent from Figure 3.13, that shows $d(h)$ (i.e., the difference between the maximum and the minimum path load after h iterations) for Scenario 2, when the network size is 400. The straight line parallel to the x-axis represents δ. From the figure, the difference $d(h)$ between the maximum and the minimum path loads eventually reaches very close to the given threshold δ. This property is unique to RaSMaLai as it benefits from controlled oscillation. Although not shown due to lack of space, for Scenario 1, the transient behavior lasts shorter, in other words fewer iterations are needed to converge. This happens since the sink positioned at the middle naturally gives better balancing.

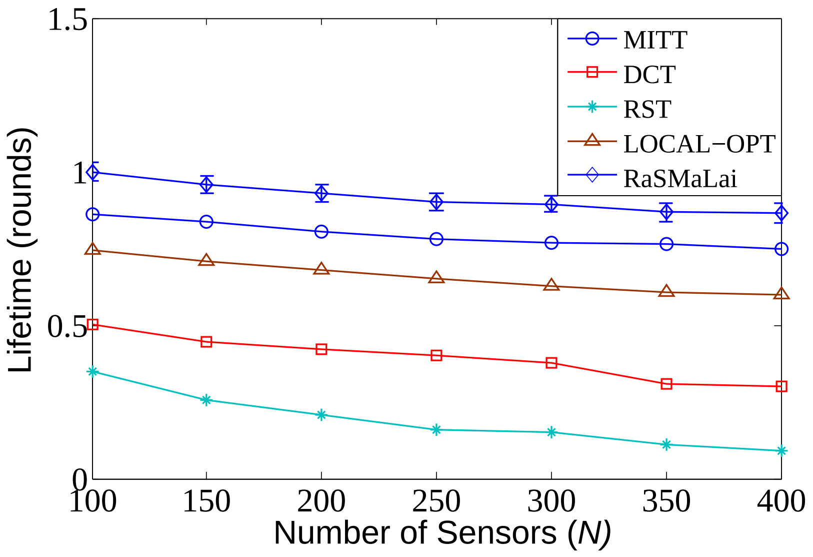

Figure 3.14: Lifetime in scenario 1

3.4.3 Experimental Results

3.4.3.1 Lifetime and Energy Efficiency

The performances of RaSMaLai is compared with other schemes (for the considered scenarios) as a function of the number of nodes in Figure 3.14 and Figure 3.15. The results on the lifetime are shown in Figures 3.14 and 3.15 for Scenarios 1 and 2, respectively. We also show the corresponding standard deviations as error bars for our approaches. In particular, Figure 3.14 clearly shows that the baseline approach represented by the RST tree has the smallest lifetime. In contrast, the tree produced by RaSMaLai has the highest lifetime. Similarly, Figure 3.15 shows how RaSMaLai outperforms other schemes. Note that RaSMaLai achieves better results in terms of lifetime than Local-Wiser and the three centralized approaches: LOCAL-OPT, DCT, and RST. In this setting, the performance of RST and DCT is very poor. This illustrates how adaptive approaches such as RaSMaLai and MITT are more resilient to varying network topology than pure graph-theoretic approaches. Unlike MITT and LOCAL-OPT, RaSMaLai allows oscillation to some extent that enables it to explore

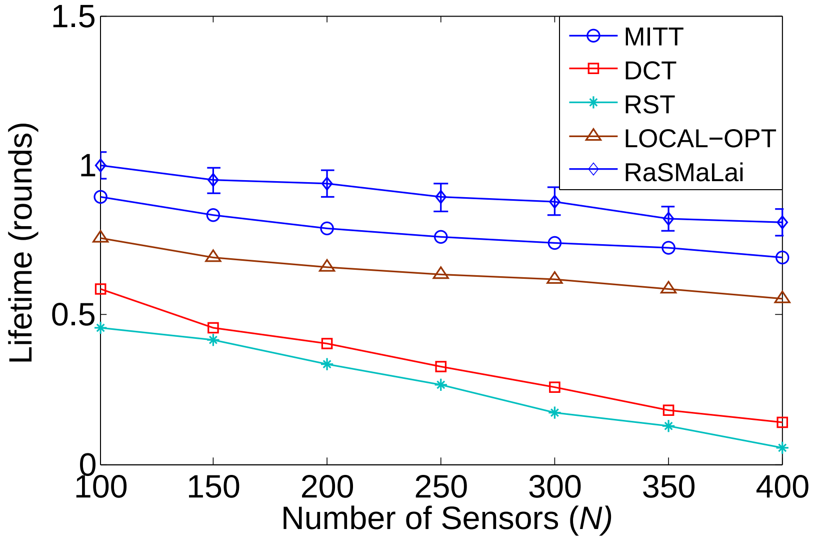

Figure 3.15: Lifetime in scenario 2

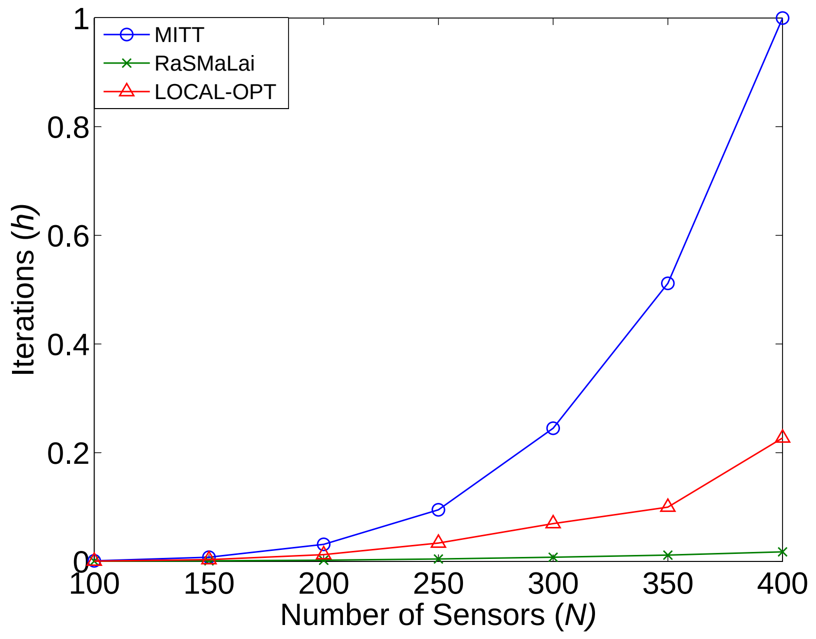

Figure 3.16: Runtime in scenario 1

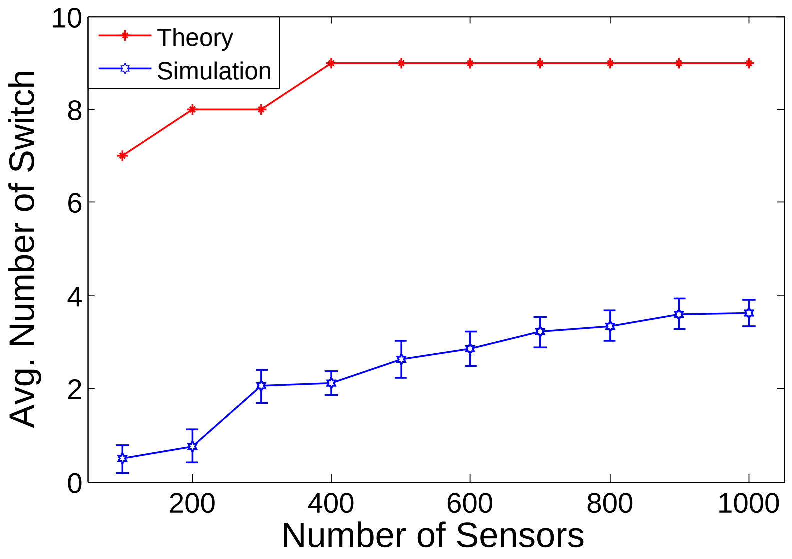

Figure 3.17: Average number of switching in scenario 1

better topologies, which may never be found by the other two algorithms. Figure 3.16 shows the average runtime of MITT, LOCAL-OPT, and RaSMaLai for Scenario 1. We observe that, as the network size increases, the runtime of RaSMaLai does not increase as much as that of MITT and LOCAL-OPT. Figure 3.16 shows that RaSMaLai is much faster than MITT and LOCAL-OPT. Similar results were obtained also for Scenario 2, but we do not report them here for brevity.

3.4.3.2 Mean Number of Switching

In Figure 3.17 and 3.18, we present our results for average number of switching that sensors participate into according to the RaSMaLai algorithm. Since, we are interested to investigate the scalability of our approach, we simulate networks from 100 sensor nodes to 1000 sensor nodes spread over 100m by 100m area. This gives us deployment with pretty low density of 0.01 to high density of 0.1. We experiment for both Scenario 1 and Scenario 2 as described before. We also study the effect of δ on average number of switching. In Figure 3.17 and 3.18, we compare the average number

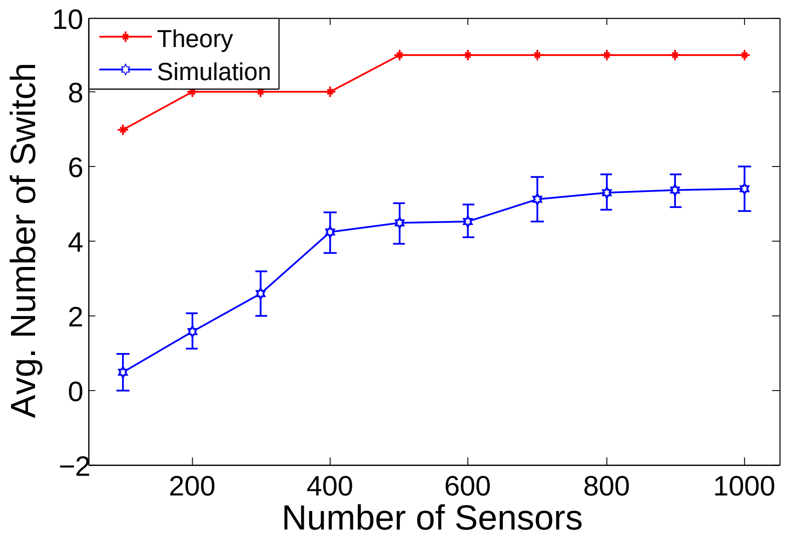

Figure 3.18: Average number of switching in scenario 2

of switching from simulation with the theoritical bound given by Theorem 3.4. We see that, with higher network size, nodes participate into greater number of switching on average. This is expected, since in such cases, each node has more neighbors and thus more potential parents for a given δ. However, the growth rate of this is very slow compare to the increase of network size, and clearly it is in logerithmic order of network size. The growth rate of average number of switching becomes slow for networks with size larger than 700 in both scenarios. We observe that, in Scenario 2, nodes switch slightly more than they do in scenario 1. This is because, in scenario 1, the sink is placed at the middle and that gives some natural symmetry in all directions. As a result, less nodes are in unbalanced contition in scenario 1 than in scenario 2. Finally, in both scenarios, no node violates the analytical bound on number of swithces.

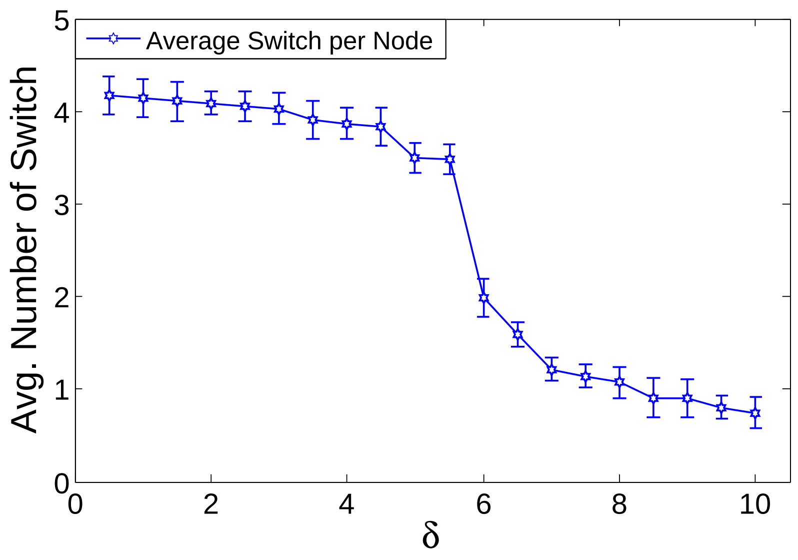

Figure 3.19: Effect of δ

3.4.3.3 Effect of δ on Number of Switching

In Figure 3.19, we study the effect of the load balancing parameter δ on the average number of switching of the sensors. We simulate a network of 1000 nodes for Scenario 1 with different δ values ranging from 0.5 to 10. We observe that as δ increases, the average switching of the nodes decreases. Larger δ limits the number of potential parents of the nodes, and hence nodes perform fewer switching. However, as we have seen that larger δ also results lower lifetime for the tree, there is a tradeoff between the global load balancing objective and the individual workload of the sensors. We note that, while lowering δ, there is a sharp increase at $\delta = 5.5$. This suggests that there is a threshold for δ for the given scenario, beyond which, load balancing may be topologically impossible. This increases switching load on individual sensors, and at the same time results into trees with lower lifetime (Figure 3.12). The analytical bound in this case is 9, and we see that the experimental results are withing the bounds of analytical result.

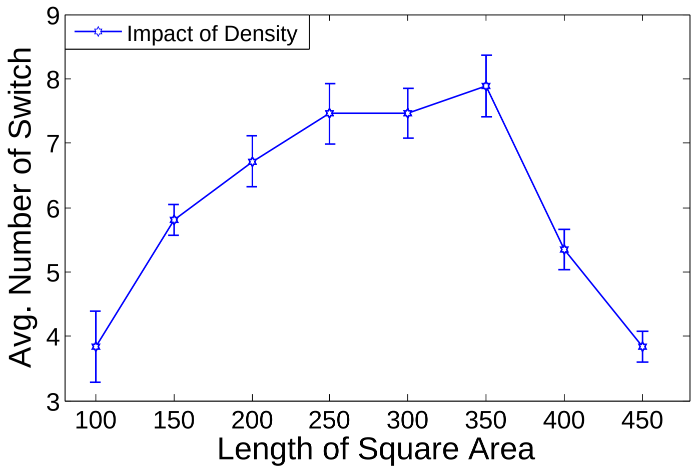

Figure 3.20: Average number of switching for δ =5

3.4.3.4 Effect of Density on Number of Switching

We also study the impact of density on node switching. We simulate a network of 1000 nodes with different deployment areas. The dimension of the area is varied from 100m by 100m to 450m by 450m. This gives a deployment with density ranging from 0.1 to 0.05. We perform this experiemnt with 3 different values of δ, namely 5, 15, and 25. From the Figures 3.20, 3.21, and 3.22, we see a common pattern that the nodes initially perform more switching with the decrease of density, and if the density is decreased even further, the switching decreases. For a given δ, when the deployment area is made larger, i.e., the density of the deployment is decreased, nodes have fewer neighbors. If two subtrees are in unbalanced condition, in dense deployment, there are more nodes that are able to switch to the subtree with lower pathload and eventually may come into balanced condition quickly. With sparse deployment, when two subtrees are in unbalanced condition, there are fewer nodes who can actually switch to the other subtree even if the individual subtrees have lots of nodes. Thus, it is less likely to have fine grained change in the pathloads in contrast to the dense deployments. Since, switching is restricted to a handful of nodes, and

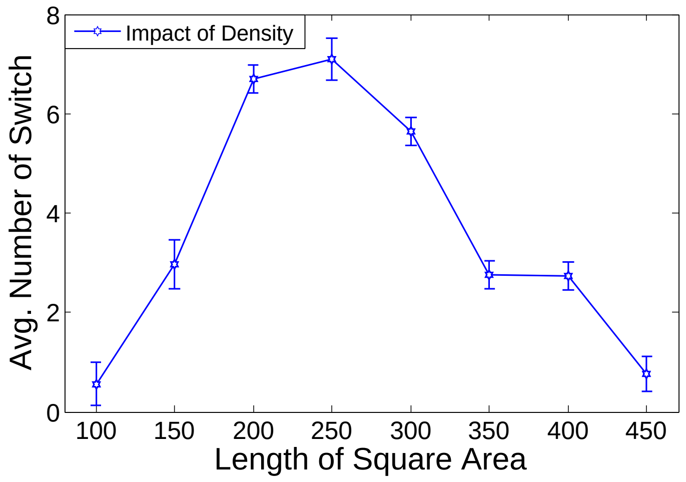

Figure 3.21: Average number of switching for δ =15

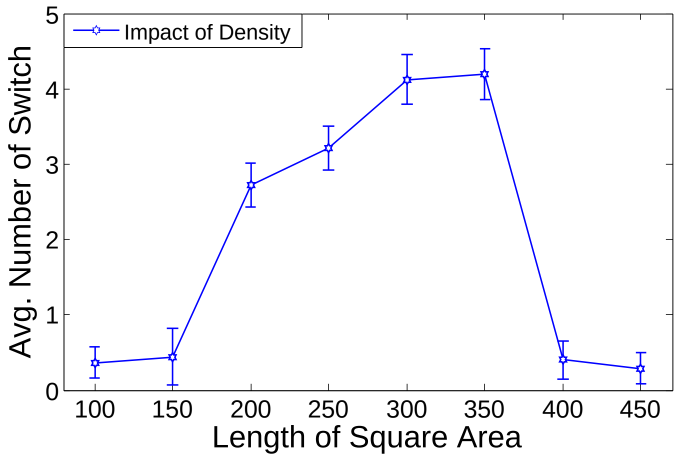

Figure 3.22: Average number of switching for δ =25

changes in the pathloads due to their switching is rather drastic. This leads to more oscillation than in dense deployments. Hence, when the density is lowered, initially, the average number of switching goes up. However, when the network is made even more sparse, nodes are barely coonected and they have very few neighbors. Fewer neighbors gives even fewer potential parents, and prevents node from from switching. Thus, in very sparse deployments, the average number of switching performed by the nodes decreses. From these figures we also see that, for a given deployment with

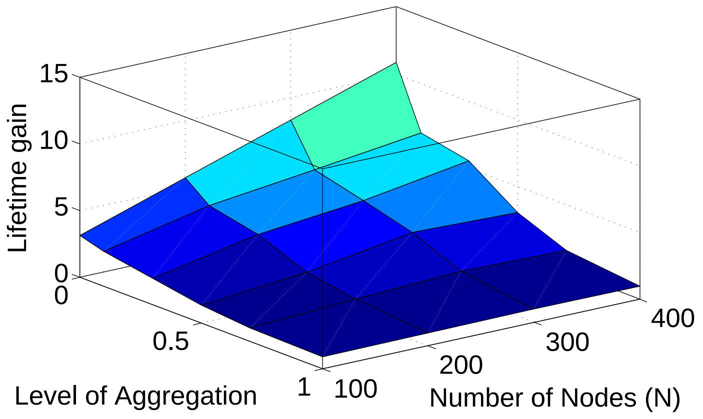

Figure 3.23: Lifetime (with aggregation) ratio with RST

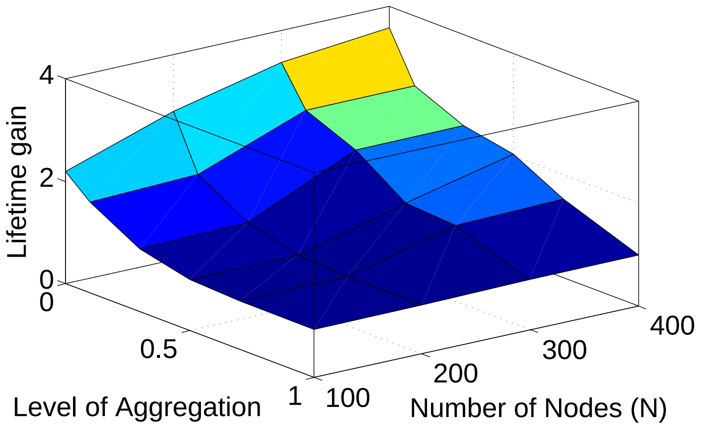

Figure 3.24: Lifetime (with aggregation) ratio with DCT

higher δ nodes participate into less switching. This is consistent with our previous results.

3.4.4 Impact of Aggregation

Figures 3.23 and 3.24 illustrate the impact of data aggregation on the performance of RaSMaLai. Specifically, we considered the lifetime of RaSMaLai relative to that of RST and DCT. We varied the level of aggregation from no aggregation (i.e., raw data), represented as 0, and full aggregation (i.e., each intermediate node in the tree combines all received messages into a single one), represented as 1 in the plots. We also varied the number of nodes for each considered value of aggregation level.

Figure 3.23 shows that, when no aggregation is used, RaSMaLai obtains a lifetime that is four times higher than that of RST for a network with 100 nodes. The relative lifetimes increases to 11 when the network size is equal to 400 nodes. On the other hand, as the aggregatio ratio increases, the gain of RaSMaLai decreases to the point that it performs only slightly better than the RST when full aggregation is used. However, for moderate aggregation (i.e., values ranging from 0 to 0.5 of the aggregation level), RaSMaLai outperforms RST, especially when the number of nodes is high.

Figure 3.24 shows the impact of aggregation with respect to DCT. Similar to the previous case, RaSMaLai achieves a higher lifetime than DCT when no aggregation is used and the size of the network is large. RaSMaLai is still 20% better than DCT when the aggregation level is 0.5 and the network size is equal to 100 nodes. At the same aggregation level, RaSMaLai is more than twice as much better than the DCT with network size of 400 nodes. For higher values of the aggregation level, RaSMaLai actually obtains a similar lifetime like that of DCT.

From the discussion above it is clear that RaSMaLai can not keep up its performance as the aggregation ratio is increased with a fixed network size. This happens because, as the aggregation ratio is increased, the load of the nodes tend to become proportional to their degrees rather the their size of the subtrees rooted at them: with no aggregation, the load directly depends on the subtree size; on the other hand size of subtree has no effect when full aggregation is used. In contrast, the node degree has the fundamental contribution to the load. Now, RaSMaLai is originally designed to work on raw data; consequenctly the switching strategy involves nodes among subtrees. In RaSMaLai, if a node is not selected for switching, its descendants are recursively considered. When the load is proportional to subtree, such switches of the descendants makes a difference. However, when the load tends to depend less and less on the subtree size, switching of descendants does not really make much difference. In addition, since it does not give any priority based on the node degree, RaSMaLai cannot cope with high levels of aggregation. However, RaSMaLai is still effective for large networks and scenarios with a low to moderate levels of aggregation.

3.5 Summary

In this chapter, we presented an efficient randomized switching algorithm, called RaSMaLai, that maximizes the lifetime of data collection trees in wireless sensor networks by means of load balancing. Based on the concept of the bounded balanced trees, our algorithm randomly switches the data forwarding paths of nodes. We provided a simple yet effective switching strategy for the sensor nodes, resulting into faster convergence. The simulation results confirm that our approaches can significantly increase the lifetime of data collection trees with a lower time complexity than other existing schemes. We also perform a thorough study on the scalability of our approach and show that the average number of switching does not violate our theoritical results. As a future work, we seek to study the effect of δ on the lifetime that can be achieved in different scenarios. Furthermore, we intend to characterize the switching probability as a function of different network parameters (e.g., density, degree) in order to reduce the convergence time.

Chapter 4

Distributed Rasmalai Algorithm

In this chapter, we propose a distributed algorithm for load balancing in WSNs. Specifically, we focus on tree based networks. Distributed algorithms are suitable to accommodate dynamic changes that occur in a sensor network. Events like node failure, link failure, interference etc. affect the routing structure of a WSN. In a tree based network, the next hop node for routing must be well defined. This next hop node is called the parent node. From the discussion in the previous chapter, we know that an efficient tree structure can be built to maximize the lifetime of a given WSN by intelligent parent selection. In case of any failure affecting the routing, the parent node of a particular sensor may need to be reselected. Thus, a distributed lifetime maximization algorithm is required that can cope up with network dynamics.

In this chapter, we extend our work on RaSMaLai (as presented in the previous chapter) to implement a distributed algorithm, named D-RaSMaLai, for lifetime maximization of tree based WSN. The rest of this chapter is organized as follows. In Section 4.1, we discuss what are the core challenges of a distributed lifetime maximization algorithm. Section 4.2 discusses the problem formulation. The D-RaSMaLai algorithm is detailed in Section 4.3. The performance evaluation is discussed in Section 4.4. Finally, Section 4.5 summarizes this chapter.

4.1 Challenges of Distributed Lifetime Maximization Algorithm

From the discussion of the previous chapter, we know that the lifetime maximization problem of tree based sensor network is computationally very hard. Thus, most existing algorithms that can produce trees with higher lifetime require high time complexity. Algorithms like MITT and LOCAL-OPT are very slow to converge. This becomes a serious concern when the network is large. If a distributed tree construction algorithm converges slowly, it introduces several disadvantages. First of all, slow convergence results into high message complexity. This creates energy overhead as a large number of control messages are generated until a stable data collection tree is produced. Thus, the distributed algorithm should be designed in such a way that the energy overhead for the control messages is minimum and does not affect the lifetime of the produced tree. Secondly, higher convergence time also impacts the update of the data collection tree in case of node failure or link failure. If the tree is not updated quickly, it may result into a large number of packet drops. Moreover, the imbalance created by node and link failures may also affect the lifetime. Thus, it is important for any distributed tree construction algorithm to converge fast with low message complexity and low energy overhead. We present our D-RaSMaLai algorithm that can meet these challenges.

4.2 Problem Formulation

In this section, we formally define the distributed lifetime maximization problem. We utilize the concept of bounded balanced trees introduced in the previous chapter.

4.2.1 System Model and Assumptions

As before, we denote $G = (V, E)$ to be a graph representing randomly placed sensor nodes over a monitoring area. We assume that sensors are placed densely enough so that there is no disconnected component in G.

In our approach, we assume that individual sensor nodes may have different initial energy budgets, and that data are forwarded without any aggregation. A node can generate data from its own sensing activity, and also receive data from other nodes. For simplicity, in the following we will assume that $R_i^t = R_i^c + R_i^g$ for each v_i. We also use the same definition of *path load* as in the previous chapter.

4.2.2 Distributed Lifetime Maximization Problem

The lifetime maximization problem for data collection trees in wireless sensor networks is presented in the previous chapter. We take advantage of the concept of the δ-bounded balanced tree, where δ signifies the difference between the maximum and the minimum path load in the tree. To find the tree with the maximum lifetime, we have to minimize δ. In the RaSMaLai algorithm, which worked in a centralized fashion, the parameter δ is taken as an input parameter. The algorithm then employ intelligent switching strategy for the sensor nodes that converge to a data collection tree with the difference of the maximum and the minimum path load very close to δ.

In the distributed lifetime maximization problem (DLMP), we consider an additional constraint to ensure the energy efficiency. Since, any distributed algorithm generates control packets, we must devise a way to minimize the number of such packets while finding the optimal δ-bounded balanced tree. This will help to make the distributed lifetime maximization algorithm energy efficient.

Let χ_i denote the number of control packets received by a node v_i in a distributed lifetime maximization algorithm. We define the distributed lifetime mximization problem as follows:

Definition 4.1 (DLMP). For a given connectivity graph G and a load balancing parameter δ, find a data collection tree T^j of G such that T^j is δ-bounded balanced and the number of control messages received by the nodes i.e., $\sum_{1\leq i\leq N} \chi_i$, is minimum.

Since the basic lifetime maximization problem is NP-complete, the distributed lifetime maximization problem is also an NP-complete problem. In the next section we present our distributed algorithm D-RaSMaLai, that can maximize the lifetime of data collection tree with few number of comtrol packets and very low energy overhead.

4.3 D-RaSMaLai Algorithm

4.3.1 Basic Principles

We build the D-RaSMaLai algorithm focusing on two key aspects of data collection trees in WSNs. Firstly, our algorithm should increase the lifetime of the computed tree. Secondly, we must limit the number of control packets generated by D-RaSMaLai to a minimum. From the analytical and simulation results discussed in the previous chapter, we know RaSMaLai uses an intelligent switching or parent selection strategy that results into faster convergence than other state of the art lifetime maximization algorithms. This encourages us to apply similar strategy for parent selection for the D-RaSMaLai algorithm also. In the case of D-RaSMaLai such intelligent parent selection results into exchange of fewer number of control packets for the nodes.

Our D-RaSMaLai algorithms works in three phases. In the first phase, each node computes its own load and informs its current parent node. In the second phase, nodes update their current path load based on the maximum load of its path towards the sink. Finally, in the third phase, the sink detects the subtree with the bottleneck node, i.e., the node with the maximum path load, and sends *Switch* messages. Upon receiving the *Switch* message, a node can select a new parent from one of its neighbors. All nodes have an initial switching probability of $\frac{1}{2}$, and if a node switches to a new parent it decreases its next switching probability by $\frac{1}{2}$.

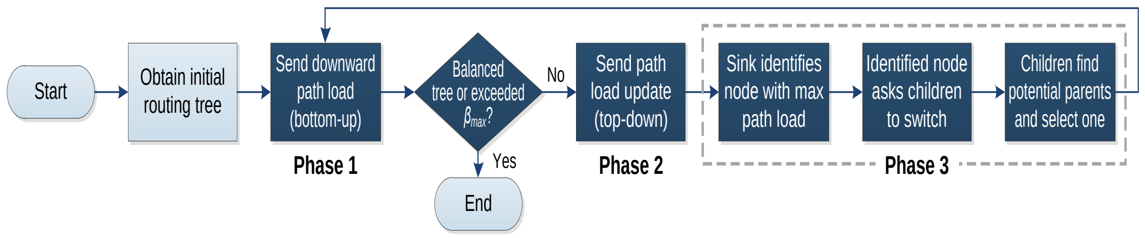
Start
Obtain initial routing tree
Send downward path load (bottom-up)
Phase 1
Balanced tree or exceeded β_{max}?
No
Yes
End
Send path load update (top-down)
Phase 2
Sink identifies node with max path load
Identified node asks children to switch
Children find potential parents and select one
Phase 3

Figure 4.1: Flow diagram of D-RaSMaLai

4.3.2 Three Phase Workflow

Now we discuss the three phases of D-RaSMaLai in details. Similar to its centralized counterpart, i.e., RaSMaLai, the distributed implementation starts with a random tree, which repeatedly goes through the different phases shown in Figure 4.1. We detail these phases below.

4.3.2.1 First Phase

In phase 1, a node transmits its current data transmission rate (R_j^t for node v_j) and a downward path load value $\hat{\sigma}_j$ in a packet to its parent v_i in the current tree. The value $\hat{\sigma}_j$ signifies the maximum load among all the paths that start from node v_j and ends in a leaf. When v_j is a leaf node, we have $\hat{\sigma}_j = \gamma_j$. Otherwise, $\hat{\sigma}_j = \max\{\gamma_j, \max\{\hat{\sigma}_k\}\}$, where $v_k \in C_j$, the set of children of v_j. After v_i receives these values from all of its children, it calculates the R_i^t and R_i^c values. Node v_i also maintains a table where the transmission rates and the downward path loads of its children are stored. Now, v_i computes its own load γ_i and the maximum downward path load $\hat{\sigma}_i$, and then transmits R_i^t and $\hat{\sigma}_i$ to its current parent. Starting from the leaves, this process is executed bottom-up at each level of the tree as shown in Figure 4.2. When v_i is a node at level 1, it only transmits the $\hat{\sigma}_i$ value to the sink. Once the sink receives the downward path load values from all level 1 nodes, it stores

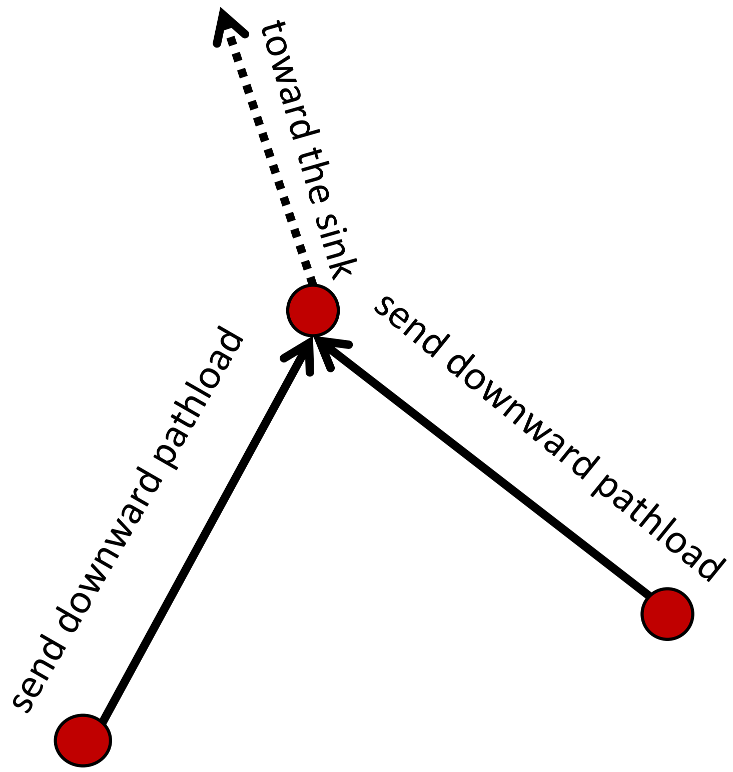

Figure 4.2: Bottom-up packet transmission in phase 1

the ID of the node with the maximum downward load value. This completes phase 1.

4.3.2.2 Second Phase

At the end of phase 1, the maximum downward path load at the sink represents the maximum load value of the current tree. Moreover, the ID of the corresponding level 1 node represents the root of the subtree that contains the node with the maximum load. This maximum load also signifies the maximum path load of all leaf nodes, $\max\{\sigma_i^M\}$. Similarly, the minimum downward path load signifies the minimum path load of all leaf nodes, $\min\{\sigma_i^M\}$. If $\max\{\sigma_i^M\} - \min\{\sigma_i^M\} \leq \delta$, the tree is balanced. Otherwise, the sink starts phase 2 by sending a *PathLoadUpdate* message

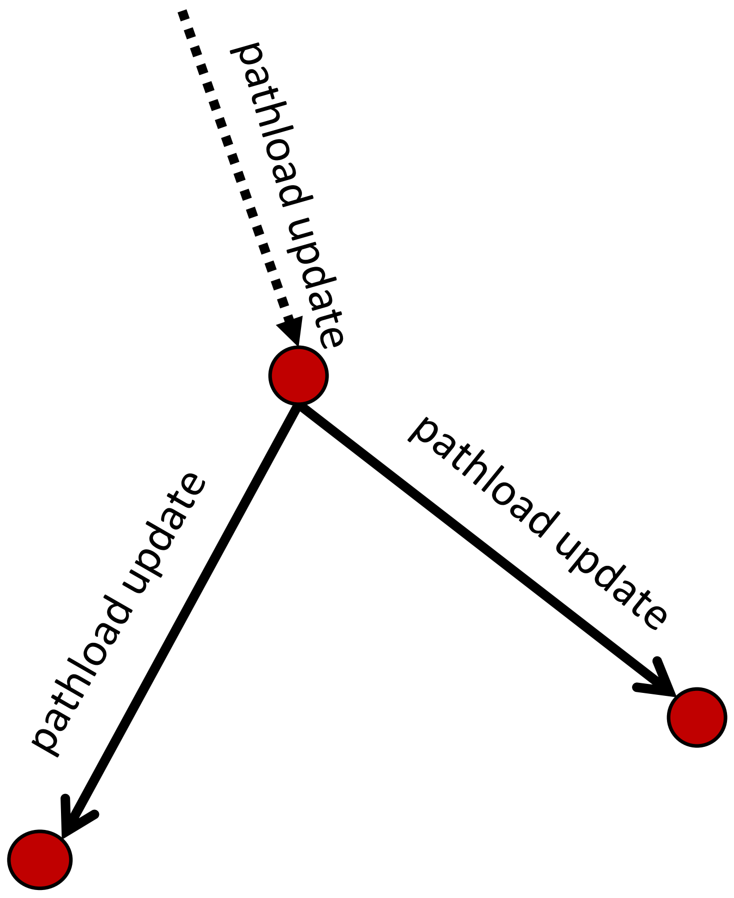

Figure 4.3: Top-down packet transmission in phase 2

to all the level 1 nodes. A node v_i at level 1 first sets $\sigma_i = \gamma_i$ and then forwards the message to their children with σ_i embedded in it. Each node then updates its path load by setting it to the maximum of their own load and the path load of its parent. This process is executed at each level in a top-down approach as shown in Figure 4.3. When a leaf node updates its path load, it sends an *Ack* message to its parent v_i, thus confirming its completion of the update. Node v_i waits to receive an *Ack* from all its children and then sends it to its own parent. This way, an *Ack* from a node v_i indicates that all nodes in $T(v_i)$ have updated their path loads. When the sink receives *Ack* from all level 1 nodes, the path loads of the entire tree have been updated and phase 2 is completed.

4.3.2.3 Third Phase

In phase 3, the sink identifies the node with the maximum load for switching. To accomplish this, the sink routes the *Find* command message down the tree. The sink first sends the *Find* message to its child v_a with the maximum downward path load $\hat{\sigma}_{\max}$. Then v_a looks up its own load γ_a and if $\gamma_a = \hat{\sigma}_{\max}$, it identifies itself as the bottleneck node. Otherwise v_a looks up its downward path load table and finds the node v_b with $\hat{\sigma}_b = \hat{\sigma}_{\max}$. Thus, v_a knows the node with the maximum load must be in $T(v_b)$, the subtree rooted at v_b. Now, v_a forwards the *Find* command to v_b. This way, the *Find* message is routed until some node v_c finds that $\gamma_c = \hat{\sigma}_{\max}$. Immediately, v_c knows that it is the current bottleneck node. As a result, it broadcasts the *Switch* message to its children. After a node v_i receives the *Switch* message, it decides whether to switch according to the strategy of RaSMaLai. If v_i decides not to switch (with probability $1 - p_i$), it broadcasts its unwillingness by sending a *SwitchDeny* message to all of its neighbors. Thus, neighboring nodes (including its parent) know that v_i will not change its current path. On the other hand, if v_i decides to switch (with probability p_i), it first collects information about its potential parents by broadcasting a *SwitchReq* message to its neighbors. This message embeds the current path load value of v_i. If a neighbor v_j does not satisfy the constraint $\sigma_i - \sigma_j > \delta$, it replies with a *RequestDeny* message. Otherwise it replies with a *RequestAccept* message and waits for a *SwitchTimeOut* period. After receiving replies from all of its neighbors, v_i knows which nodes are its potential parents and chooses one of them with uniform probability as the new parent. Then v_i immediately sends a *Join* message to its new parent. Figure 4.4 illustrates this scenario.

When other potential parents do not receive the *Join* message within the *SwitchTimeOut* period, they know that v_i has chosen some other node for switch-

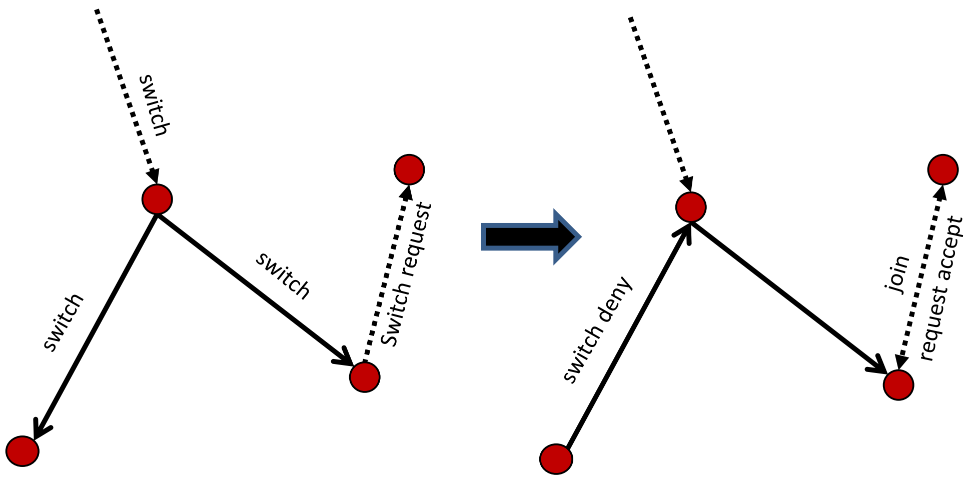

Figure 4.4: Packet transmission in phase 3

ing. The *Join* message contains the values for R_i^t and $\hat{\sigma}_i$. The new parent of v_i updates its own load and downward path load accordingly. Alongside, v_i also sends a *Leave* message to its old parent. In turn, the old parent deletes R_i^t and $\hat{\sigma}_i$ from its table and updates its load and downward path load, knowing that nodes in $T(v_i)$ are no longer its descendants. When v_i has no pending *SwitchReq* message, it is done with switching for the current round. Finally v_i updates its load and sends its updated R_i^t and $\hat{\sigma}_i$ to its parent. As in phase 1, the updates of load and downward path load go along the path from v_i to the sink. This way, the loads and downward path loads are updated while a switching takes place. Thus, after the initialization, phase 1 is actually executed jointly with phase 3. Also, in the subsequent rounds of phase 2, a node forwards the *PathLoadUpdate* request to its children only if its own path load has changed from the previous round. This optimizes phase 2 by forwarding the update only along those paths that have participated in switching.

According to our switching strategy, nodes progressively have a lower switching probability as the iterations proceed. Also, some subtrees may become balanced locally, and thus nodes may have fewer potential parents. Consequently, as the steady state approaches, nodes in a subtree may become less likely to perform switching. If a bottleneck node realizes its descendants are not likely to switch further, either because they have very low switching probabilities or have only a few potential parents or both, it suppresses its *Switch* message in phase 3. A node can identify this scenario by looking at its own path load. If its path load is not changed during a series of consecutive rounds, it may conclude that its descendants are not likely to switch further, and at this point it ceases to send further *Switch* messages.

The process described above continues until either a δ-bounded balanced tree is found or the sink has sent the *Find* message $\beta_{\max}$ times. Thus, the distributed version, D-RaSMaLai terminates after a maximum of $\beta_{\max}$ rounds. Like the RaSMaLai algorithm, we set $\beta_{\max}$ to $N\sqrt{\log \frac{N}{\sqrt[2]{\delta}}}$ to ensure the convergence.

4.4 Performance Evaluation

In this section, we evaluate the performance of our algorithm, D-RaSMaLai through simulation experiments.

4.4.1 Simulation Setup

We simulated D-RaSMaLai in MATLAB, under the same conditions as for the RaSMaLai scheme [10]. Specifically, we considered a deployment wherein sensor nodes are randomly placed within a square area of 100 m $\times$ 100 m. The number of N sensors was varied from 100 to 400. Each sensor was randomly assigned an initial energy between 0.5 to 1 Joule (J), and the radio transmission range was set to 25 m. We assumed the energy required to receive a message was 50 nJ/bit and the energy required to transmit a message was 100 nJ/bit. The data packet size was 16 bytes. The size of control messages was assumed to be 10 bytes. Each data point of simulation was averaged over 100 runs, and scaled between 0 and 1.

We considered two scenarios associated with different locations of the sink in the network. In Scenario 1, the sink was placed in the middle of the deployment area, namely, at the (50 m, 50 m) coordinate. In Scenario 2, the sink was placed at one side of the deployment area, specifically, at the (100 m, 50 m) coordinate. We then evaluated the following metrics:

- *lifetime*, measured as the number of data collection rounds until the first node runs out of energy;
- *energy expenditure*, measured as the amount of energy spent to transmit and recieve the control packets in the distributed implementation;
- *scalability*, characterized in terms of the average number of message forwarding overhead per node with the size of the network.

We compared our results with the following data collection schemes: a) LocalWiser [48]; b) a random shortest path tree (RST); c) a degree-constrained tree (DCT); d) LOCAL-OPT [47]; e) MITT [2] for Scenario 1. Among these algorithms, LocalWiser is implemented as a distributed algorithm and we compare the energy overhead results of D-RaSMaLai with that of LocalWiser. For DCT, the degree was set to 5 after some preliminary simulations performed to ensure that a connected tree is obtained without overloading the nodes. Finally, we also study the scalability of our approach in terms of average number of packets forwarded by the nodes.

4.4.2 Experimental Results

The performance of D-RaSMaLai is compared with that of other schemes (for the considered scenarios) as a function of the number of nodes. The results for the lifetime are shown in Figures 4.5 and 4.6 for Scenarios 1 and 2, respectively. We also show the corresponding standard deviations as error bars for our algorithms.

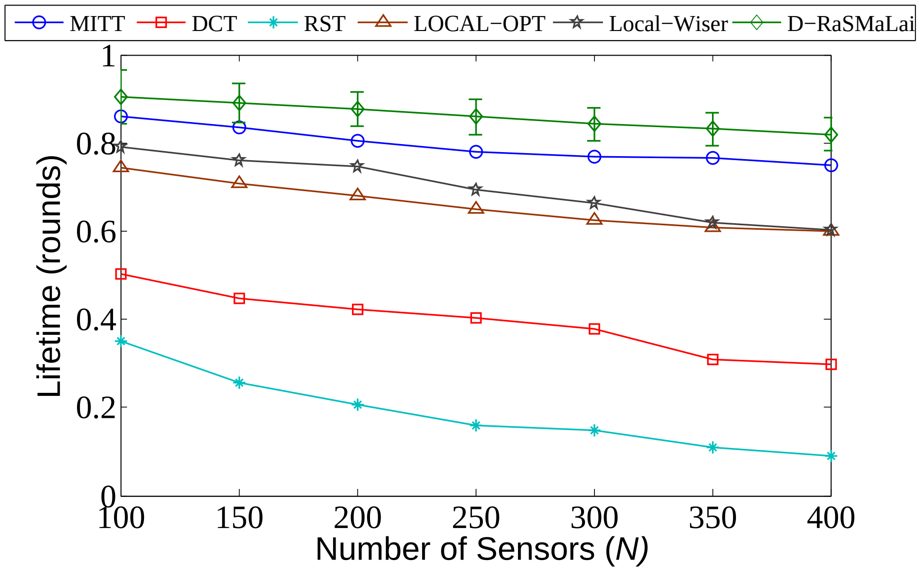

Figure 4.5: Comparing lifetime in scenario 1

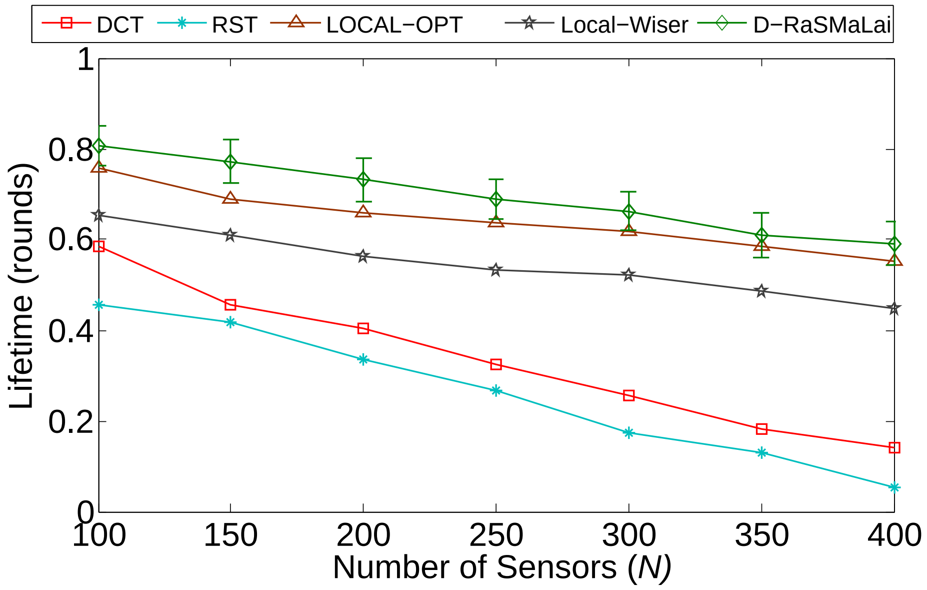

Figure 4.6: Comparing lifetime in scenario 2

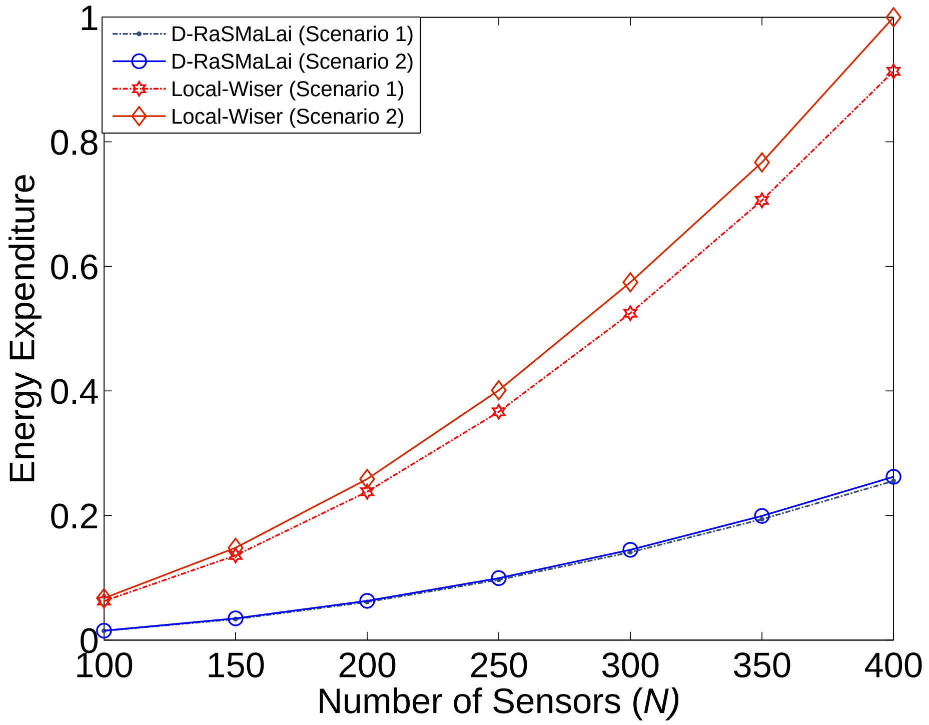

Figure 4.7: Energy expenditure in the two scenarios

In particular, Figure 4.5 clearly shows that the baseline approach represented by the RST tree has the lowest lifetime. In contrast, the tree produced by D-RaSMaLai has the highest lifetime. Similarly, Figure 4.6 shows how D-RaSMaLai outperforms the other schemes. Note that D-RaSMaLai achieves better results in terms of lifetime than Local-Wiser and the three centralized approaches: LOCAL-OPT, DCT, and RST. In this setting, the performance of RST and DCT is very poor. This illustrates how adaptive approaches such as D-RaSMaLai and MITT are more resilient to varying network topology than pure graph-theoretic approaches. Unlike MITT, LOCAL-OPT, and Local-Wiser D-RaSMaLai allows oscillation to some extent that enables it to explore better topologies, which may never be found by the other two algorithms.

The energy expenditure of the two distributed algorithms D-RaSMaLai and Local-Wiser is shown in Figure 4.7. We observe that, for large networks, Local-Wiser

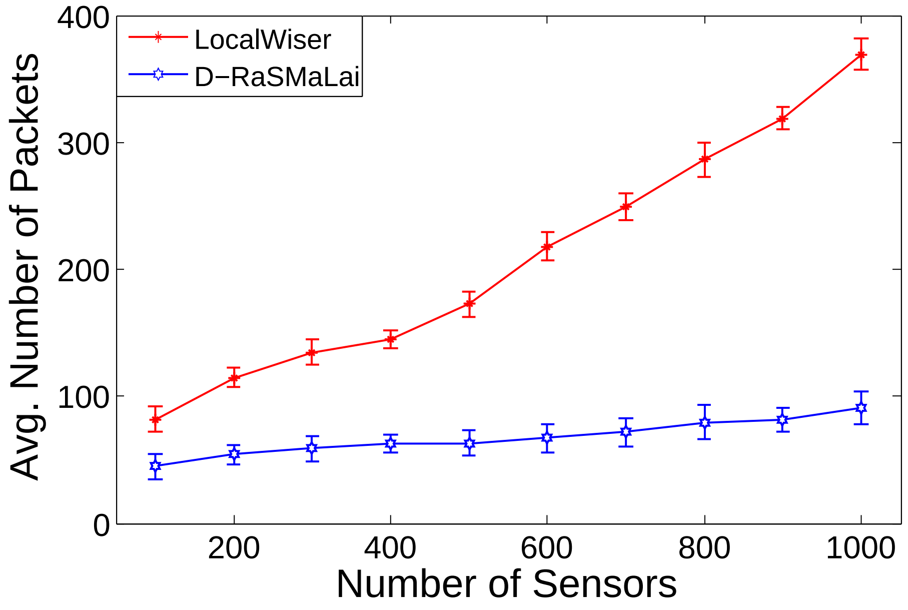

Figure 4.8: Average number of forwarded packets in scenario 1

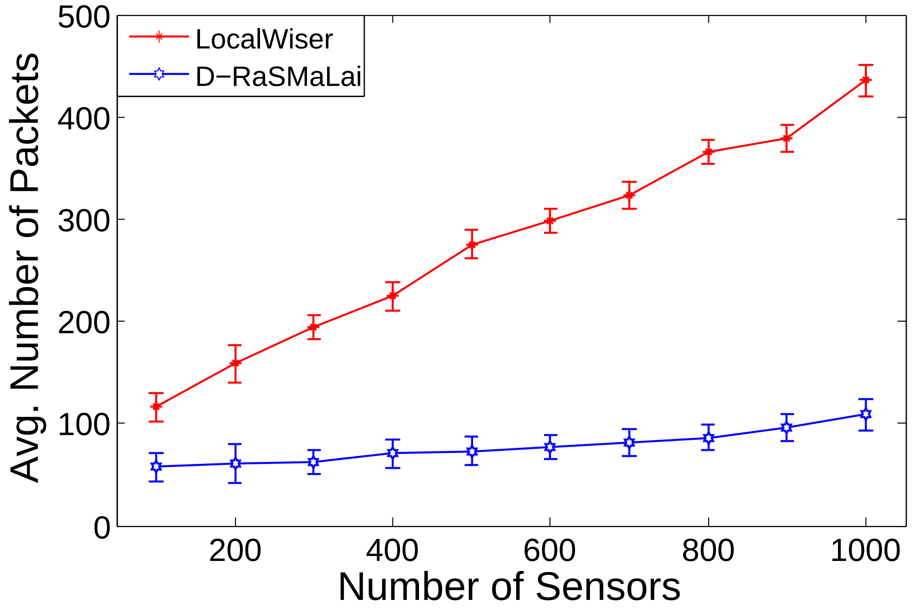

Figure 4.9: Average number of forwarded packets in scenario 2

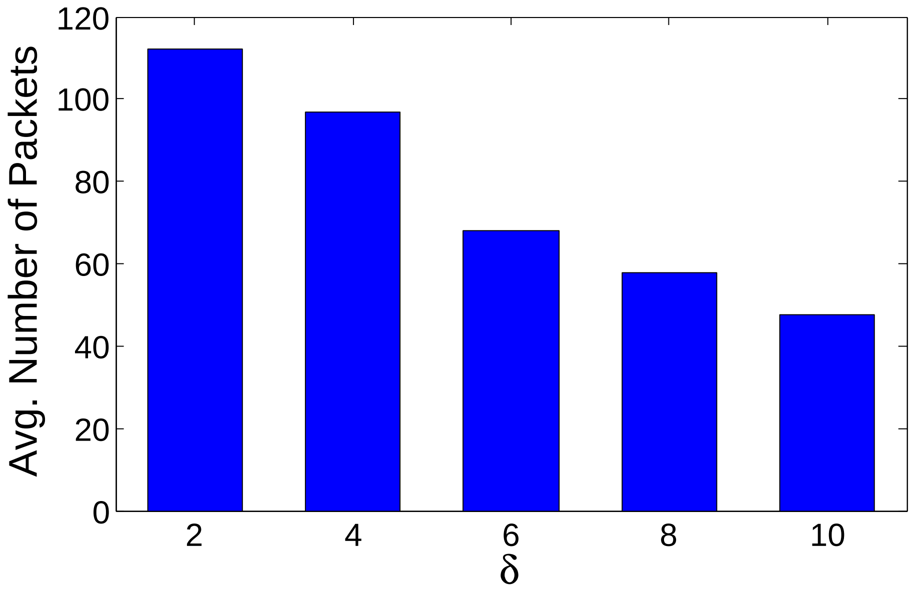

Figure 4.10: Average number of forwarded packets as a function of δ

consumes nearly five times more energy than D-RaSMaLai in both scenarios. The energy consumption of D-RaSMaLai does not vary much with the location of the sink. For Local-Wiser, the energy consumption is slightly higher when the sink is on the border. D-RaSMaLai, instead, is more stable and not sensitive to the location of the sink.

4.4.2.1 Average Number of Packets

We also analyze the overhead of the D-RaSMaLai. To this end We compare the average number of packets forwarded by both D-RaSMaLai and Local-Wiser. Specifically, Figures 4.8 and 4.9 show the average number of packets forwarded by each node for Scenarios 1 and 2, respectively. We observe that, with increasing size of the network, nodes forward a higher number of packets in both protocols. This happens as, in a larger network, nodes have a higher number of neighbors to forward packets to. Furthermore, the forwarding load on the nodes increases much faster

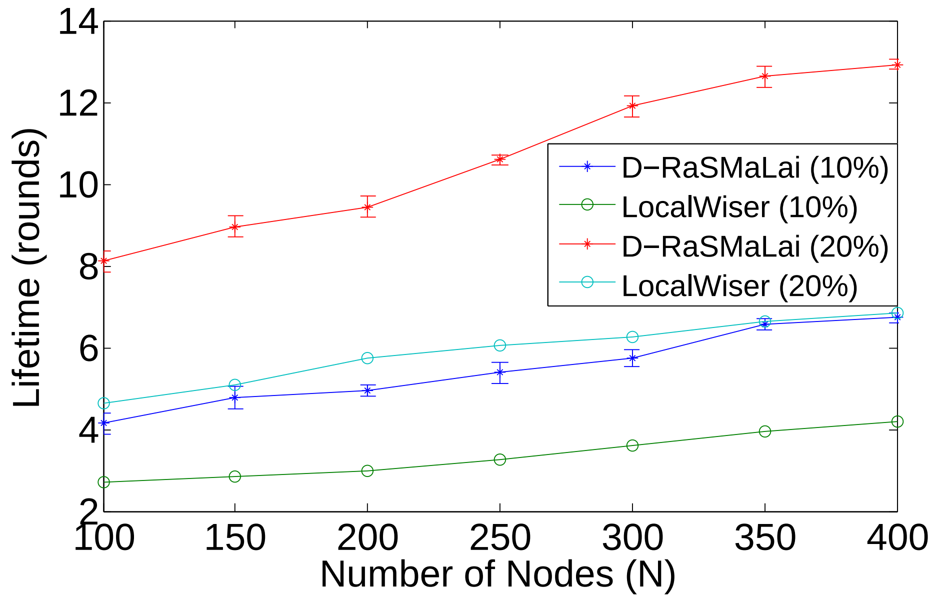

Figure 4.11: Lifetime until a percentage of nodes die

for Local-Wiser rather than for D-RaSMaLai. Specifically, with Local-Wiser nodes forward at least twice as many packets as with D-RaSMaLai in both scenarios, with higher values in Scenario 2. From the figures, we clearly see that D-RaSMaLai is more energy-efficient than Local-Wiser and also more scalable as its forwarding load increases slowly as the network size increases. Figure 4.10 shows that nodes forward less packets with D-RaSMaLai when the load balancing parameter δ increases. This is expected, since higher δ results in a fewer number of unbalanced subtrees, and eventually results in less packet forwarding load.

4.4.3 Impact of Different Definitions of Lifetime

Finally, we measure the performance of our algorithm according to a different definition of lifetime. Specifically, we measure the lifetime of a network as the time until a certain percentage of nodes runs out of battery power. Figure 4.11 compares the lifetime of D-RaSMaLai and Local-Wiser in two different scenarios, corresponding

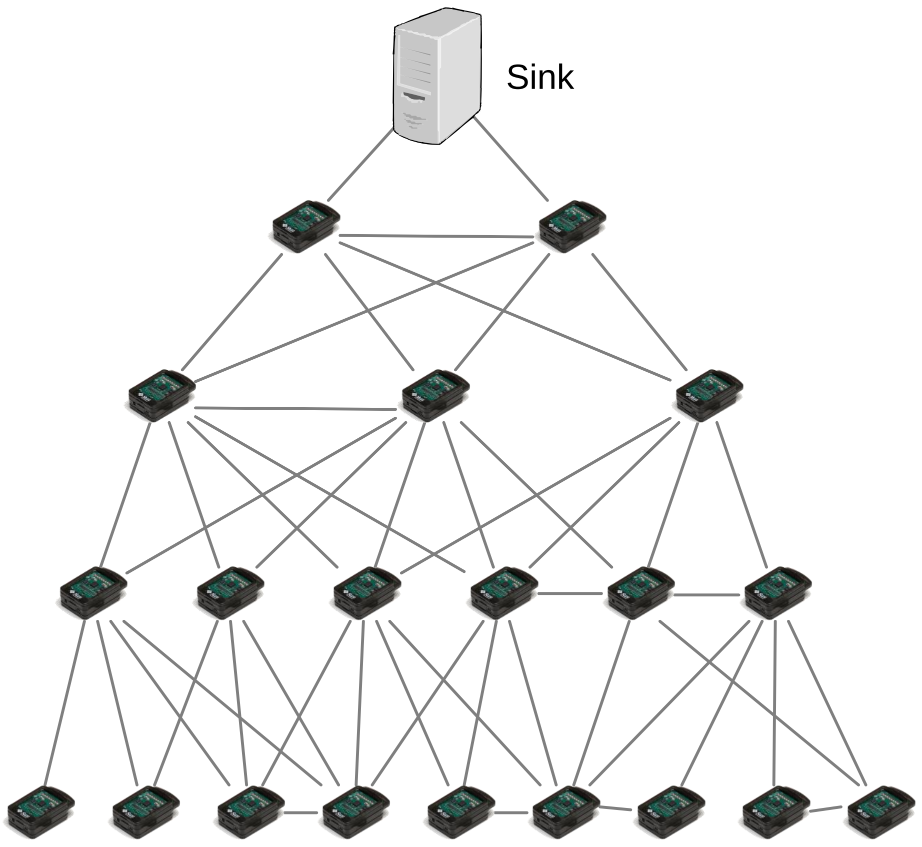

Figure 4.12: Connectivity graph of the network in the testbed experiments

to different thresholds for the lifetime (i.e., 10% and 20%, respectively). The figure shows that, considering a threshold of 10% and a network of 100 nodes, D-RaSMaLai results in a 50% higher lifetime than Local-Wiser. The gain is even higher when the number of nodes is higher as well. Considering a threshold of 20% and a network of 100 nodes, D-RaSMaLai increases the network lifetime by 71% over that of Local-Wiser. With the same threshold, for a network of 400 nodes, this gain is further increased to 88%. We again observe that the proposed algorithm performs far superior for larger networks. Furthermore, as we relax the definition of lifetime by increasing the percentage of nodes allowed to die, the relative gain of D-RaSMaLai becomes much higher.

Table 4.1: Summary of the testbed experiments

Metric	**Local-Wiser**	**D-RaSMaLai**
Lifetime (10^3 rounds)	418.65	561.76
Number of switches	n/a	0.87
Number of packets	56.38	19.44
Energy consumption (mJ)	165.78	65.39

4.4.4 Testbed Experiments

We implemented D-RaSMaLai and Local-Wiser for the SunSPOT wireless sensing platform [65]. SunSPOTs operate at 3.7 V and have a built-in 750 mAh lithium-ion battery. They are equipped with the CC2420 radio, whose current draw is 17.4 mA during transmission, 18.8 mA during reception, and 30 μA in the deep sleep state. Data packets have a maximum length of 128 bytes, and are transmitted at a nominal rate of 250 kbps rate. We deployed 20 devices in a indoor sensing area of 20 m $\times$ 20 m. The physical topology of the sensor nodes, along with the corresponding connectivity graph, are illustrated in Figure 4.12. The sink was represented by a workstation using a wireless sensor node as a gateway. The sink collected the data from the whole network and processed them to obtain the metrics of interest. We configured the SunSPOT MAC to use a duty cycle of 5% and a time slot length of 200 ms. Nodes generated one messages during each active slot; for D-RaSMaLai δ was set to 0.005.

The experimental results are summarized in Table 4.1. We measured lifetime as the number rounds until the battery level is reported as low by the operating system of the sensors, once convergence has been reached.

We see from the table that Local-Wiser obtains a lower lifetime than D-RaSMaLai. In terms of energy efficiency, D-RaSMaLai generates fewer control packets per node than Local-Wiser. Consequently, the average energy spent to run the D-RaSMaLai is also lower, due to limited overhead. The table also shows how the average number of packets and switching are comparable with the corresponding values obtained in the simulations, thus validating our earlier findings.

4.5 Summary

In this chapter, we presented an energy efficient distributed algorithm, namely D-RaSMaLai, for lifetime maximization of tree based WSNs. We devise a novel three phase packet exchange method for D-RaSMaLai to keep the number of control packets lower. It employs efficient path load based parent selection approach that produces data collection tree with significantly higher lifetime. In addition, the novel switching strategy of D-RaSMaLai results into very low energy overhead. From the extensive simulation as well as the experimental results, we show that D-RaSMaLai indeed is more suitable for distributed implementation, specially for large scale sensor deployment.

Chapter 5

Lifetime Maximization by Compression Tree

Compression of correlated data is one of the widely used techniques where sensory data are compressed along their routes towards the sink. Consequently, a data compression tree is formed where a sensor selects its parent based on the degree of correlation among their sensed data, and data from the child node is compressed at the parent node. In periodic data collection, data collected by individual sensors can be considered as a time series. The amount of correlation between two time series of sensory data may not be constant over time. Most works in this direction do not consider the temporal effect of correlation among data streams generated by periodic sensing. Also, compression indroduces some imperfection and the compressed data may introduce loss of relability. We address the problem of energy efficient data gathering in WSN while considering variability of correlation among data streams of neighboring sensors. We propose a bucket approximation based framework named EFFECT (energy efficient framework for compression tree) that produces a compression tree based on the compression ratio of data streams from the neighboring sensors in a given sensor network. We perform experiments on real data sets and show that our framework can produce trees with significantly higher lifetime while reducing the average energy consumption of the sensors by at least 20%.

Compression-based schemes explicitly attempt to reduce the amount of data transmission in such a way that sensor nodes have less energy expenditure in terms of data forwarding. Nodes in close proximity often have high correlation in their sensed data. Thus, data compression approaches are suitable to address the energy efficiency

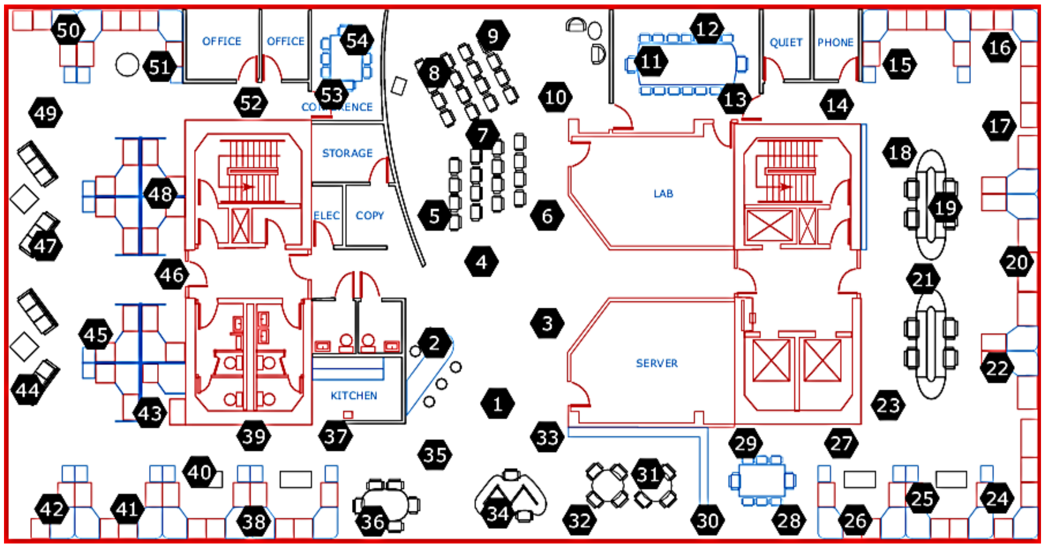

Figure 5.1: IntelBerkeley lab sensor deployment layout

problem [13] by creating a *compression* tree. However, existing approaches in this domain often encounter three major challenges: a) how to accommodate varying degree of correlation (particularly, temporal) among sensory data streams and; b) how to reconstruct original signals within a specified error bound. Here, we propose a novel framework that efficiently addresses the above challenges.

5.1 Motivating Example of Dynamic Correlation

The energy benefit of data compression mostly depends on the compressibility of data. In a sensor network, this eventually depends on the degree of correlation among data streams of neighboring sensors. This correlation can be categorized as spatial and temporal correlation [66]. Spatial correlation expresses how the data dependency varies based on the geometric distance of the sensors; the closer the sensors are, the higher the correlation among their sensed data. On the other hand, the degree of correlation between two static sensors may also vary with time, mostly due to inherent nonlinearity of the physical data.

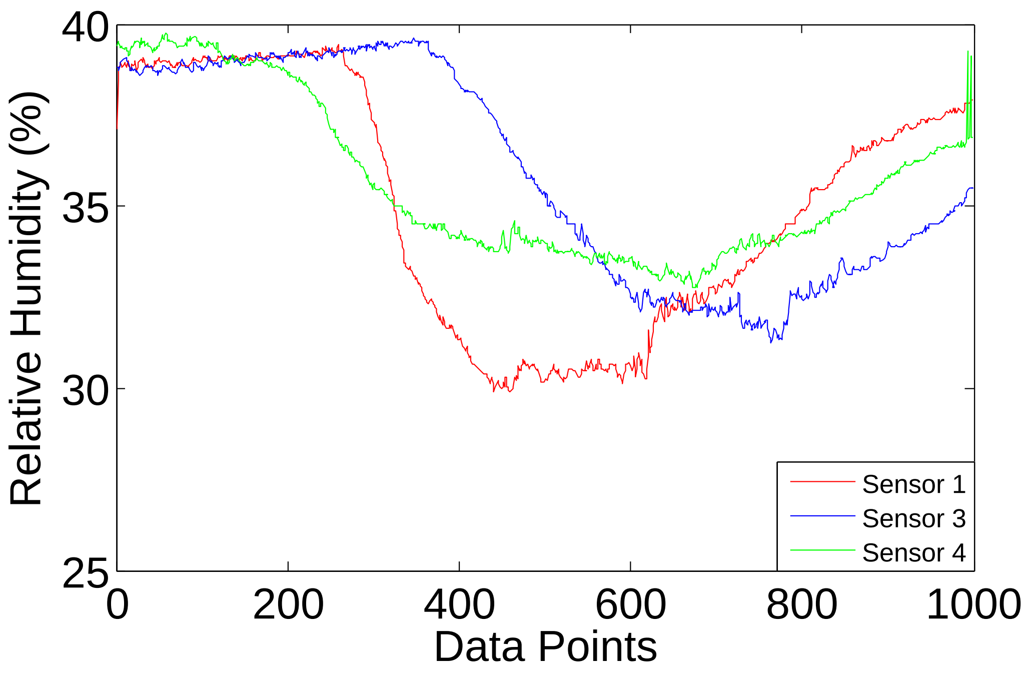

Figure 5.2: Sensor data as time series

We illustrate this by using sensory data from Intel Berkeley lab [67]. Figure 5.1 depicts their deployment with 54 sensors for collecting temperature, humidity, light, and voltage data over a period of one month. In Figure 5.2, we plot 1000 data points for relative humidity of three sensors, namely sensors 1, 3 and 4 from 5.1. We observe that, with time not only do the individual readings change, but also the degree of correlation among the readings vary. For instance, during the first 250 readings, data from sensors 1 and 3 are almost identical. Then the correlation decreases as suggested by a large gap between their respective lines from data points 300 to 600. In the last segment, this gap is slightly reduced, thereby denoting higher correlation.

From the above discussion, we see that even though the actual sensors are static, the temporal correlation among their data streams is inherently dynamic. Another aspect of data compression in WSNs is that different applications anticipate different levels of accuracy in the collected data. It is also intuitive that applying an arbitrary compression mechanism may not allow to reconstruct the original data from the compressed signals within a specific error bound. Thus, to compute a compression tree for energy efficiency, we need to use a framework that can accommodate the dynamic correlation of the data sensed by the sensor nodes and also guarantees that the original data streams can be reconstructed from the compressed signals within a given error bound.

The major contributions of this chapter are as follows.

- We propose a bucket approximation based framework called EFFECT to build energy-efficient compression tree for a given sensor network that accommodates both the spatial and the temporal correlation of the sensory data streams. The framework does not require any prior knowledge of the sensory signal statistics.
- EFFECT ensures that for a given $\theta > 0$, no more than θ error is introduced while reconstructing the original signals.
- With the help of extensive simulation experiments on real data traces collected by the Intel Berkley Lab [67], we demonstrate that our framework is highly energy efficient, providing higher lifetime for data collection trees.

The rest of this chapter is organized as follows. Section 5.2 formulates the problem of compression tree. The bucket approximation algorithm is presented in Section 5.3. Section 5.4 elaborates the concept of compression tree. The performance of our compression tree framework is presented in Section 5.5. Finally, Section 5.6 summarizes this chapter.

5.2 Problem Formulation

In this section, we first introduce the system model and underlying assumptions. Then we formally define the energy efficient compression tree problem.

Table 5.1: List of symbols

Symbol	**Description**
G	Sensor network graph
v_0	Sink node
v_i	i-th node
K	Total number of sensor nodes in G
N_i	Set of neighbors of v_i
e_i	Energy level of sensor v_i
S_i	Time series data of v_i
R_{ji}	Relative values of S_j with respect to S_i
f_i	Set of children of v_i
n_i	Children of v_i with relative signals
D_i	Amount of traffic v_i receives from its children
L_i	Bucket approximation of data stream S_i
Z_{ji}	Bucket approximation of R_{ji}
T	Number of data points in the stream S_i
α_i	Number of data points in bucket L_i
β_{ji}	Number of data points in Z_{ji}
θ	Approximation error bound to retrieve the original signals
E_t	Transmission cost of a single measurement
E_r	Reception cost of a single measurement
$q(.)$	Action-value function for Q-learning

5.2.1 System Model and Assumptions

As before, we use $G = (V, E)$ to denote a graph representing arbitrarily placed sensor nodes over a monitoring area A. Let N_i denote the neighbors of node v_i. Let the current energy level of v_i be referred to as e_i.

Definition 5.1 (Compression Tree). A *compression tree* in the sensor network is a spanning tree of G with v_0 as the root. Any node in a *compression tree* is capable of performing data compression on its incoming sensory data streams to reduce the amount of outgoing data stream.

Let a data collection round denote the time period during which sensors collect a set of measurements and send them to the sink [62, 2]. Specifically, we assume each

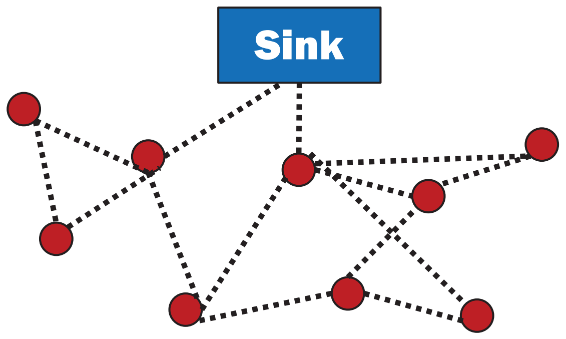

Figure 5.3: Connectivity graph, G

sensor periodically collects T measurements and sends these data points to the sink during one data collection round. Our first step to deal with the temporal and the spatial correlation in a single framework is to consider the sensory stream as a time series data. We define the time series S_i to represent the ordered set of sensed data collected during a data collection round by sensor v_i. Specifically, $S_i = \{s_i^1, s_i^2 \ldots, s_i^T\}$, where s_i^j is the measurement of some physical environment parameter of sensor v_i at time instant j.

In our compression model, the data streams from neighboring sensors are grouped and one base signal is selected from each group. We call the nodes with base signals as compression nodes. Data streams from non-compression nodes can be represented by some relative signal with respect to the base signals. Thus for a given sensor network G, we need to select a set of compression nodes, whose measurements will be used as base signals, and compute the relative signals for the rest. Then, these base signals and relative signals are compressed using bucket approximation scheme [68], and routed to the sink. In the bucket approximation scheme a time series is partitioned into several time segments (called as buckets), and from each segment one representative value is selected to approximate all values in that segment within a given error bound. Let $\hat{S}_i$ be the reconstructed signal from the approximation of

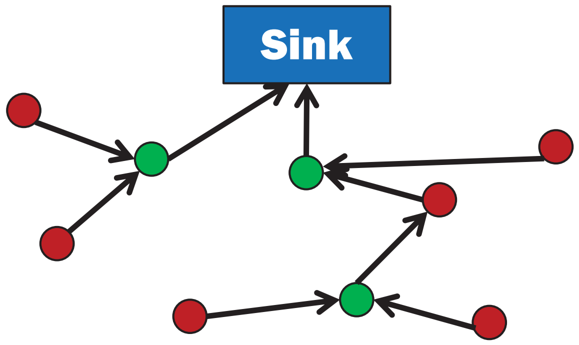

Figure 5.4: Compression tree 1

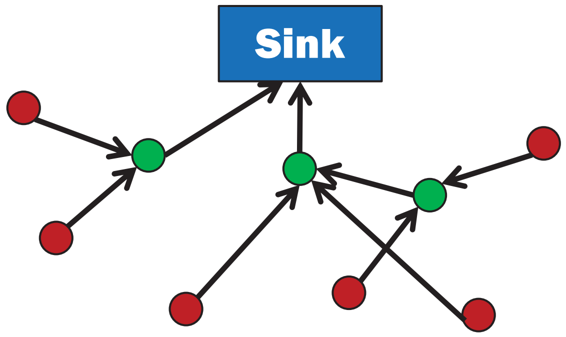

Figure 5.5: Compression tree 2

S_i. Our compression framework, EFFECT, applies bucket approximation to ensure that $\forall t \in T$, $|\hat{s_i^t} - s_i^t| \leq \theta$, for a given error bound $\theta > 0$.

Figure 5.3, illustrates a network G with 9 sensor nodes. Two possible compression trees of G are displayed in Figures 5.4 and 5.5. The green nodes represent the compression nodes, and both trees have three compression nodes. The signals of these nodes are considered as base signals. The signals of other nodes are approximated with base signals at the compression nodes.

5.2.2 Energy Model for Compression Tree

Assume in a given compression tree, f_i is the set of children of sensor node v_i. The nodes in f_i can be divided into two sets. For one set of nodes, the forwarded signals to v_i are approximated with S_i. We denote this set of nodes as $n_i \subseteq f_i$. For the other set of nodes (i.e., $f_i \backslash n_i$), the forwarded signals are not approximated with S_i. In both cases, the forwarded signal from a neighbor $v_j \in f_i$ has two parts. One part represents the signals that v_j received from its children in the current compression tree. Let this part be denoted with $D_j = \{d^1, d^2, \ldots, d^\gamma\}$. The other part is the signal of v_j itself, i.e., $S_j = \{s_j^1, s_j^2, \ldots, s_j^T\}$.

Now consider a node v_j from the set n_i. Consequently, S_j will be approximated with S_i. Let, $R_{ji} = \{r_{ji}^1, r_{ji}^2, \ldots, r_{ji}^T\}$ denote the data values of S_j relative to S_i, i.e., it is the relative signal of node v_j with respect to node v_i. Thus, we can reconstruct a compressed representation of S_j given compressed representations of S_i and R_{ji}. Now, assume we divide $S_i = \{s_i^1, s_i^2, \ldots, s_i^T\}$ into α_i time segments, and for each segment we select one value representing all values in that segment. In other words, we compress S_i with T data points, to a signal L_i with α_i data points, i.e., $L_i = \{l_1, l_2, \ldots, l_{\alpha_i}\}$. Similarly, we compress relative signal R_{ji} as $Z_{ji} = \{z_1, z_2, \ldots, z_{\beta_{ji}}\}$. We expect the relative signal to be more flat than the original signal, so that there would be fewer distinct intervals for the relative signals. We assume that the data stream D_j is forwarded by v_i to its parent without any change. The rationale behind this is, D_j represents signals from children of v_j. Some signals in D_j are relative signals, while the rest are base signals. All of these signals are assumed to be approximated either at node v_j or at some other downstream nodes. To keep the approximation error bounded, these signals are not approximated further.

Now consider a child of v_i, say v_k from the set $f_i \backslash n_i$. In this case, v_k compresses S_k directly and v_i receives data streams D_k, and L_k. Since, L_k will not be approximated by S_i in this case, v_i simply forwards L_k and D_k without any change.

We denote E_t and E_r as the data transmission and reception cost for the data packets containing a single measurement of S_i and R_{ji}. We derive the energy expenditure of v_i for nodes in n_i as $E_{i1} = E_r * (\sum_{v_j \in n_i} (\gamma_j + T)) + E_t * (\sum_{v_j \in n_i} (\gamma_j + \beta_{ji}))$. Similarly, the energy expenditure for nodes in $f_i \backslash n_i$ is derived as $E_{i2} = (E_r + E_t) * (\sum_{v_j \in f_i \backslash n_i} (\gamma_j + \alpha_j))$. Thus, the total energy spent by v_i in one data collection round is given by $E_i = E_t * \alpha_i + E_{i1} + E_{i2}$.

5.2.3 Energy Efficient Compression Tree Problem

Given a sensor network G, and an error bound θ, the energy efficient compression tree problem can be described as follows: select a set of nodes for base signals and others for relative signals such that a set of time intervals can be generated as $\{\alpha_i\}$, and $\{\beta_{ji}\}$, respectively, that minimizes the total energy consumption of the network and also all signals can be reconstructed within maximum error of θ. In other words, we want to minimize $\sum_{v_i \in N \backslash \{v_0\}} E_i$, with $|\hat{s_i^t} - s_i^t| \leq \theta$, $\forall v_i \in N \backslash v_0$ and $t \in T$. In the next section we present the compression framework in details.

5.3 A Bucket Approximation based Compression

We first illustrate how the data stream S_i of a node v_i can be compressed independently. Then we explain how relative signals are computed using S_i as the base signal. The compression of relative signals also follows the same technique as compression of a single data stream.

While sensing the physical world parameters such as temperature or humidity, consecutive measurements of a sensory data stream are correlated. Even though this correlation may change over time, the measurements collected during small time intervals will be highly correlated. Thus, if a sensor collects T measurements in a data collection round, we can divide these measurements into a number of time segments, $\alpha_i < T$ such that individual measurements in one segment are highly correlated. Consequently, all of the measurements in a segment can be approximated

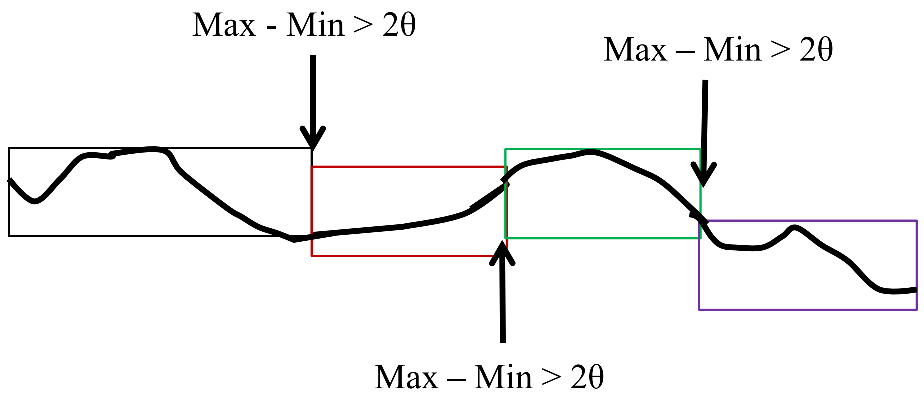

Figure 5.6: Partitioning a signal with bucket approximation

with a single measurement. Using a representative measurement for an interval to approximate all measurements of that interval introduces some error. We need to select the minimum number of intervals in such a way that this error is bounded by a specified threshold θ.

5.3.1 Compressing Single Data Stream

Let us assume node v_i collected T measurements in a data collection round, represented by $S_i = \{s_i^1, s_i^2, \ldots, s_i^T\}$. We need to find the minimum number of time segments α_i such that S_i can be mapped to $L_i = \{l_1, l_2, \ldots, l_{\alpha_i}\}$, where l_j denotes the approximation for all measurements between t_j and t_{j+1}, and $|l_j - s_i^t| \leq \theta$, for $t_j \leq t < t_{j+1}$.

Following the approach described in [68], a greedy bucketing strategy can be applied to find the minimum number of time segments, where each bucket represents a time segment, and contains a set of measurements collected during that segment. We start with an empty bucket and mark it as the *current bucket*. We read through the values $\{s_i^t\}$, and add them to the current bucket until the values in the current bucket violate the condition max$\{s_i^t\}$ - min$\{s_i^t\} \leq 2\theta$. When this constraint is violated, we create a new bucket, mark it as the current bucket and repeat the above steps. We continue with this procedure until all measurements are added to some bucket. Thus, after this process ends, the difference between the maximum and the minimum entries of any bucket is no more than 2θ. As a result, all measurements of a bucket can be approximated within θ error by taking the average of the maximum and the minimum entries as the representative value. Figure 5.6 illustrates this scenario.

After compression, v_i can transmit these α_i data points instead of T data points to save energy. We say the *compression ratio* achieved by v_i is $\frac{T-\alpha_i}{T}$.

5.3.2 Compressing Multiple Data Streams

In case of multiple data streams, we can achieve compression for individual sensor nodes using the above approach. However, we can also use signals of some of the sensors as base signals, and compute relative signals for the rest. We can then compress the base, and the relative signals using the bucket approximation method. This approach has significant advantages over compressing each of the data streams individually. First, since proximate sensors have strong correlation among their data streams, sharing a base signal for a group leads to significant energy savings. Second, individual signals can be approximately reconstructed using the information contained in the relative signals. Since relative signals are computed in a way to have lower variation, they result in fewer number of time segments after applying the bucket approximation method. Finally, finding appropriate base signals also helps with parent selection. If in a neighborhood, S_i is the optimal base signal for S_j, then v_j can select v_i as its parent, thus eventually leading toward building the compression tree.

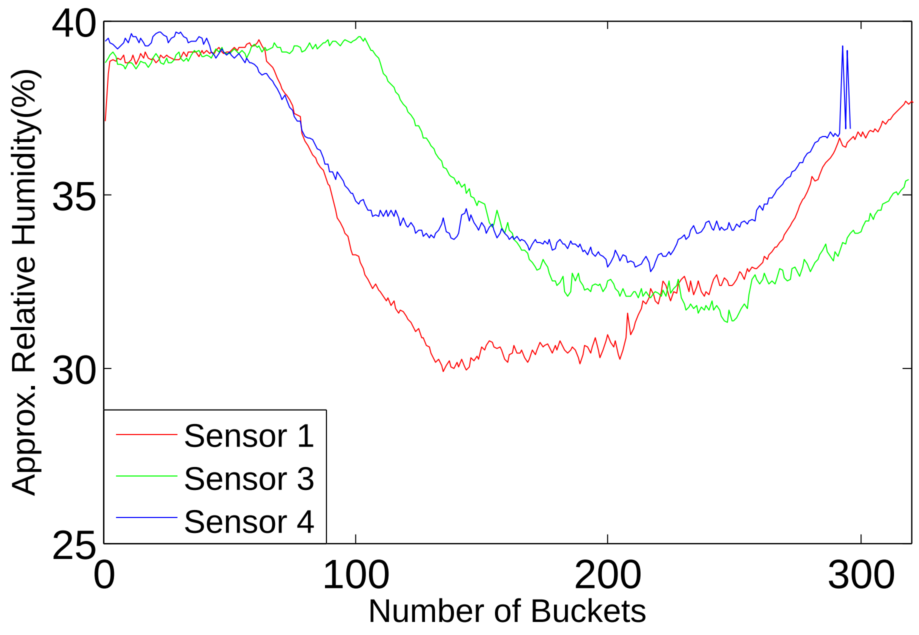

Figure 5.7: Base signals

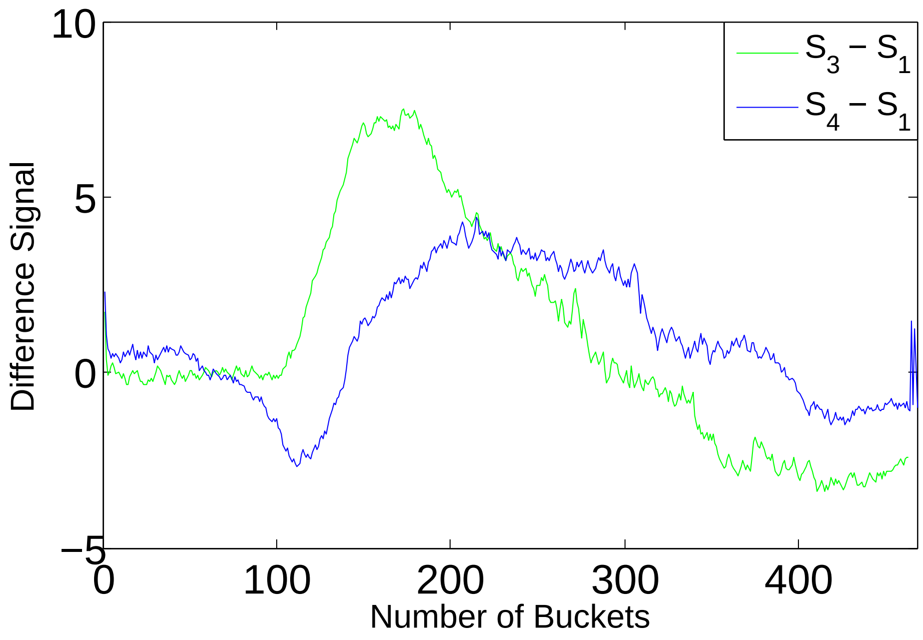

Figure 5.8: Difference signals

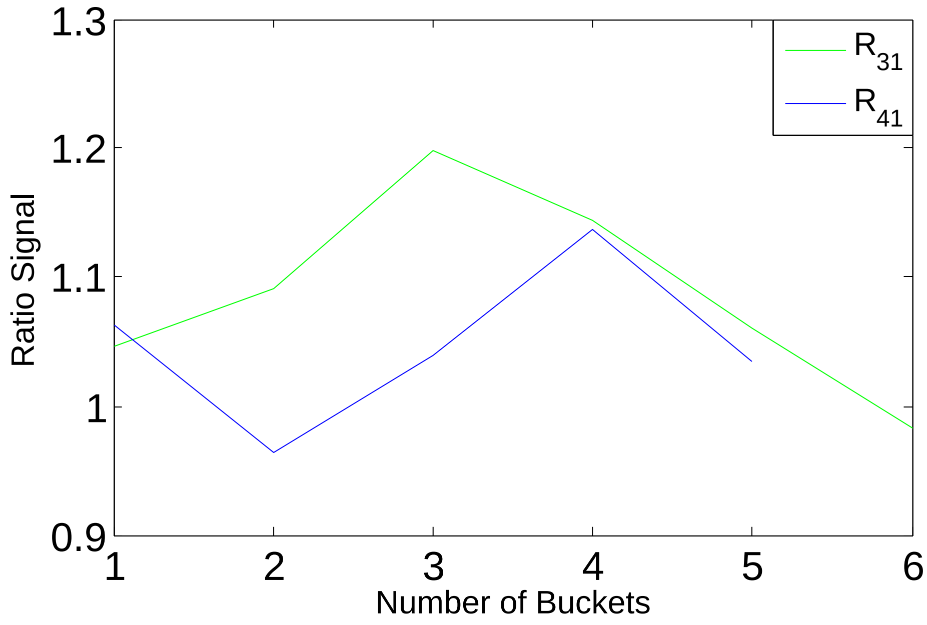

Figure 5.9: Ratio signals

5.3.2.1 Computing Relative Signals

Assume, v_i receives data streams from $|f_i|$ of its neighbors. Among these $|f_i|$ streams, we form a group with $|n_i| + 1$ streams, making S_i the base signal for this group. In the next section we detail a Q-learning based approach [69] to discover optimal grouping. For now, we focus on how to compute relative signals given a particular group, and a base signal.

The purpose of the relative signal is to achieve multi-stream compression for a group, using the bucket approximation strategy. We want to achieve minimal energy overhead for transmission of relative signals. Thus, for a given base signal, the relative signals should be computed in such a way that they do not vary significantly over time. In other words, these signals should be as flat as possible to reduce the number of resulting buckets in the bucket approximation method. It can be shown that the ratio signals are highly compressible for our purpose [70].

Given a base signal S_i, and another signal S_j, we compute the relative signal $R_{ji} = \frac{S_j}{S_i}$. That is, we express the relative signal as the ratio signal $\{r_{ji}^t\} = \{\frac{s_j^t}{s_i^t}\}$. We compare our choice of ratio signal as relative signal with two other alternatives. The first alternative is the regression model, e.g., linear regression, to express S_j in terms of S_i. In such a model, we have $S_j = a * S_i + b$, where coefficients a and b are determined to minimize the approximation error. However, since the degree of correlation between S_i and S_j changes over time, in practice we need to compute a, b not just once but for all segments that S_i and S_j are partitioned into. To make a and b constant over a segment, the segments have to be of shorter duration. This reduces overall compressibility of the regression model while adding complexity for computing the coefficients.

On the other hand, one may argue that the difference signal, i.e., $S_j - S_i$ would make a better relative signal in terms of compressibility. This would work fine, if v_i and v_j experience roughly the same amount of change of the physical parameter over an interval of Δ, i.e., $s_j^{t+\Delta} - s_j^t \approx s_i^{t+\Delta} - s_i^t$. However, most of the physical world parameters are interrelated and they change in a nonlinear fashion. Change of humidity, viscosity of fluids, and many other phenomena are all governed by inherently nonlinear equations [71].

In Figure 5.7, we illustrate the application of bucket approximation on the base signals, difference signals, and the ratio signals on three data streams from Figure 5.2. The difference and the ratio signals are computed with the signal of sensor 1 as the base. In all cases, we use an approximation error bound of $\theta = 0.05$. From Figures 5.8 and 5.9, we observe that the ratio signals can be approximated with much fewer number of buckets than the difference signals for the same error bound. Hence, ratio signals are much flatter than the difference signals, making them highly compressible using the bucket approximation method. In this example, the compression ratio achieved on the difference signals is 0.535, while on the ratio signals it is 0.994.

5.3.2.2 Approximating Group Signals

Now assume, in a compression tree, v_i has f_i nodes as its children. Thus, in each data collection round, v_i receives $|f_i|$ data streams. We form a group with $|n_i|$ signals from $|f_i|$ signals that are not already approximated, and select S_i as the base signal for this group. The rest $|f_i| - |n_i|$ signals are already approximated, and to maintain the error bound, these are not further approximated.

For all signals $S_j \in n_i$, we first compute the relative signals $\{R_{ji}\}$ with S_i as the base signal. Then the base signal S_i as well as each of the relative signals $\{R_{ji}\}$ are compressed using the bucket approximation. Assume, we obtain α_i buckets after compressing S_i, and the representative values of these buckets are expressed as $L_i = \{l_1, l_2, \ldots, l_{\alpha_i}\}$. All such values of S_i between t_k and t_{k+1} are approximated with $l_k \in L_i$. Similarly, we compress R_{ji} to $Z_{ji} = \{z_1, z_2, \ldots, z_{\beta_{ji}}\}$, and values of R_{ji} between t_k and t_{k+1} are approximated with $z_k \in Z_{ji}$. Thus, we can compute approximate value for $\{s_j^t\}$ in the range $t \in [t_k, t_{k+1})$ as $\{\hat{s}_j^{t_k}\} = \{l_{t_k} * z_{t_k} | l_{t_k} \in L_i \wedge z_{t_k} \in Z_{ji}\}$.

Assume the approximation error for the base signal S_i is θ_1, such that $|s_i^k - l_k| \leq \theta_1$, for $1 \leq k \leq \alpha_i$. Similarly, the approximation error for a relative signal R_{ji} is θ_2, implying $|r_{ji}^k - z_k| \leq \theta_2$, for $1 \leq k \leq \beta_{ji}$. It can be shown that [70], the approximation error for S_j with base signal S_i and relative signal R_{ji} is given by $\theta = \max_{1 \leq t \leq T}\{\frac{s_t^j}{s_t^i}\}\theta_1 + \max_{1 \leq t \leq T}\{s_t^i\}\theta_2 + \theta_1\theta_2$.

5.4 Computing Compression Tree

In this section, we present a Q-learning strategy [69], that computes an energy efficient compression tree for a given sensor network. Q-learning is a reinforcement learning technique where the agents (here sensors) explore a given environment by taking different actions. It has been extensively used in various optimization problems of wireless sensor networks [72].

Using Q-learning in our framework has several advantages that are suitable for time series data. First, it does not need any prior knowledge about the data that is being collected by the sensors. This is because we do not assume any static correlation among the data beforehand. Rather our framework monitors the current degree of correlation among the sensory data and groups them accordingly. The second advantage is, since Q-learning is inherently dynamic, the groups formed for compression can also be dynamically updated according to the deviation of degree of correlation.

In the Q-learning strategy, a set of states is defined for the agents, and agents can change their state through different actions. The goal of each agents is to reach to a desired state by taking a series of actions. Agents receive rewards in response to each action, and in turn learn about the appropriate actions to arrive at their desired state. The learning process assigns Q-values to each possible action, which represents the approximate benefit of the action. The agent repeatedly selects and executes a set of actions, then receives corresponding rewards, f to update the Q-values. Over time the agent learns the real action values, which it uses to select the most appropriate route.

In our compression tree problem, the agents are individual sensors. To form an efficient compression tree, sensors need to learn two things. First, whether the signal of a sensor would be used as a base signal, and second which parent node to forward to. The state of a sensor v_i is denoted as $< v_j, a_{ji} >$, where v_j is the parent of v_i while a_{ji} is a boolean variable that indicates if S_i is compressed with or without a base signal. If v_i chooses to compress its own signal S_i without any base signal, a_{ji} $= 0$. On the other hand, if v_i selects S_j as the base signal for S_i, then $a_{ji} = 1$.

The action denotes the next hop selected in the route, and the choice of base signal. The action of v_i is represented by λ^i_{kj}, which denotes a transition from state $< v_j, a_{ji} >$ to state $< v_k, a_{ki} >$. This indicates that v_i changes its parent from v_j to v_k, and also decides whether or not it wants to be approximated with S_k. By taking this action, v_i receives certain reward $\Re^i_{kj}$ and updates the corresponding Q-value $q(\lambda^i_{kj})$. The difference between the reward and the Q-value of an action is that, the reward denotes a measure of an immediate preference for an action. On the other hand, the Q-value denotes a long term reward associated with that action. Consequently, a low value of $\Re^i_{kj}$ does not confirm that the action λ^i_{kj} is undesirable, because depending on the actions from other sensors, the action λ^i_{kj} may prove to be really good for v_i. The goal of v_i is to learn better actions for the long term, i.e., for which it gets the highest Q-values. To ensure that these actions lead to energy efficient compression tree, we need to map the energy expenditure of different actions to appropriate rewards and Q-values.

5.4.1 Q-values for Compression Tree

Since a compression tree provides a routing structure leveraging data compression, we define Q-values with two components, one is routing, and the other is compression. In the routing part, we consider two parameters, namely path load and the hop count. Path load [10] for a sensor node v_i is the minimum load along the path from this node to the sink, where the load of node v_i is defined as the ratio of its energy requirement to forward traffic in the compression tree, and its residual energy, i.e., $\frac{E_i}{e_i}$. This load parameter denotes the desirability of a path selected by v_i. If a path has high load value, this implies there are bottleneck nodes along this path having high traffic load, or low residual energy, or both. On the other hand, the hop count denotes the total energy required to transport packets from v_i to the sink. Thus, v_i should consider paths with lower path loads and fewer hop counts to avoid bottleneck nodes and thus save energy.

For the compression part, we use the parameter compression ratio. Assume S_i has T data points. Now, if v_i selects v_j as the parent, and indicates that S_i will be approximated with S_j, i.e., $a_{ji} = 1$, then the compression ratio achieved by this action is given by $\frac{T-\beta_{ji}}{T}$. Otherwise, if v_i selects v_j as the parent but $a_{ji} = 0$, the compression ratio achieved is $\frac{T-\alpha_i}{T}$.

Let hc_j be the minimum number of hops from v_i to the sink through v_j. Let σ_j denote maximum load of all nodes along the path from v_i to the sink through v_j. Initially, σ_j is set to $\frac{1}{e_j}$.

5.4.1.1 Initialization

We initialize Q-values for node v_i taking action λ^i_{kj} as, $q(\lambda^i_{kj}) = \frac{1}{hc_j} * \frac{1}{\sigma_j} * \frac{T-(a_{ji}*\beta_{ji}+(1-a_{ji})*\alpha_i)}{T}$. Thus, the initial preference for a parent is given to the neighboring nodes with shorter paths to the sink, higher energy level, and higher compression ratio. Eventually, these values will be updated to learn the true preferences for the following dynamics: a) compression ratio may change due to varying degree of correlation, b) neighbors initially offering shorter routes may change their parents, and c) nodes with higher initial energy may have lower residual energy due to high traffic load.

5.4.1.2 Updating Q-values

Following the approach in [69], we first present the generic update rule for Q-learning: $q(\lambda^i_{kj})_{new}$=$q(\lambda^i_{kj})_{old} + \phi_1[\Re^i_{kj} + \phi_2 \text{max}_r\, q(\lambda^i_{rk}) - q(\lambda^i_{kj})_{old}]$. Here, $\phi_1 \in$ [0,1] is called the *learning rate* parameter. If ϕ_1 is closer to 0, it gives higher weight on the old Q-value, and convergence will be slow. On the other hand, making ϕ_1 closer to 1 drives faster learning of Q-values. If $\phi_1 = 1$, at each step the old Q-value is completely ignored.

On the other hand, $\phi_2 \in$ [0,1] is called the *discounting factor* for future rewards $q(\lambda^i_{rk})$ that can be achieved from state $< v_k, a_{ki} >$. If $\phi_2 = 0$, future rewards that can be accumulated by an action is completely ignored. As ϕ_2 gets closer to 1, future rewards play a greater role in measuring the goodness of an action.

We now discuss our reward function $\Re^i_{kj}$ corresponding to a state transition from $< v_j, a_{ji} >$ to $< v_k, a_{ki} >$. Our reward function basically measures the relative gain that v_i achieves by such a transition. This gain has two parts: one denotes the gain in the routing strategy, and the other quantifies the gain in the compression ratio. In the routing part, we measure gain across three parameters, the ratio of loads, the ratio of path loads, and the ratio of hop counts. The gain in compression is expressed with relative compression ratio. For this state transition, following four cases may arise:

Case I. $a_{ji} = a_{ki} = 0$: In this scenario, v_j is the old parent of v_i. The signal S_i was being compressed by itself without a base signal. Now v_i switches its parent from v_j to v_k. With this new parent, v_i still compresses its own signal without requiring S_k as the base. We define the reward for this action as $\Re^i_{kj} = (\frac{e_k}{E_k})(\frac{E_j}{e_j})(\frac{\sigma_j}{\sigma_k})(\frac{hc_j}{hc_k})$.

Case II. $a_{ji} = 0$, $a_{ki} = 1$: This is slightly different from the previous scenario in the sense that, S_i now needs to be approximated with respect to S_k. We define the reward for this action as $\Re^i_{kj} = (\frac{T-\beta_{ki}}{T-\alpha_i})(\frac{e_k}{E_k})(\frac{E_j}{e_j})(\frac{\sigma_j}{\sigma_k})(\frac{hc_j}{hc_k})$.

Case III. $a_{ji} = 1$, $a_{ki} = 0$: In this scenario, S_j has been the base for S_i, but after switching to v_k, node v_i now compresses its signal by itself. We define the reward for this action as $\Re^i_{kj} = (\frac{T-\alpha_i}{T-\beta_{ji}})(\frac{e_k}{E_k})(\frac{E_j}{e_j})(\frac{\sigma_j}{\sigma_k})(\frac{hc_j}{hc_k})$.

Case IV. $a_{ji} = 1$, $a_{ki} = 1$: In this scenario, S_j has been the base for S_i, and after switching to v_k, the signal S_k now becomes the new base signal for S_i. We define the reward for this action as $\Re^i_{kj} = (\frac{T-\beta_{ki}}{T-\beta_{ji}})(\frac{e_k}{E_k})(\frac{E_j}{e_j})(\frac{\sigma_j}{\sigma_k})(\frac{hc_j}{hc_k})$.

5.4.1.3 Action Selection Policy

In the Q-learning strategy, the agents must have a policy for an action selection. There is a tradeoff in selection policy as agents must balance the exploration of new actions and exploitations of learned actions. Since, agents want to maximize the Q-values, in a greedy approach the agents take the action with highest Q-value repeatedly. However, this can severely limit the solution quality by settling at local maxima. We use the well-known ϵ-greedy strategy [69], where the agents (i.e., sensors) select a random action with probability $\epsilon > 0$, and the action with the highest Q-value otherwise.

5.5 Performance Evaluation

In this section, we evaluate the performance of EFFECT framework on real traces from Intel Berkley Lab data [67].

5.5.1 Simulation Setup

We considered a network with 54 sensors that were deployed over a rectangular area of 40 m by 30 m. We placed the sensors according to the coordinate data from Intel Berkley lab. Each sensor was randomly assigned an initial energy between 1 to 5 Joules (J). We set the radio transmission range to 8 m. We assumed a packet containing a single data point had a size of 8 bytes. The energy required for transmission and reception were assumed to be 100 nJ/bit and 50 nJ/bit, respectively [10]. To apply bucket approximation for compression, we used humidity and temperature data from Intel Berkley lab. After some initial experiments, we set the learning rate parameter ϕ_1 to 0.65 and the discounting factor ϕ_2 to 0.7 for faster convergence. We measured the performance of the following metrics:

- *average number of data points*, denotes the number of samples after compression;
- *compression ratio*, measures the goodness of compression;
- *energy expenditure*, expressed as the average amount of energy spent per node in each data collection round;
- *lifetime*, measured as the number of data collection rounds until a certain percentage of nodes die.

In all of our experiments, we assumed the total error bound θ to be 0.1. To satisfy this value for θ, we varied the error bound for base signals, θ_1, from 0 to 0.01, according to the equation in the Subsection 5.3.2.2. On the other hand, θ_2 denotes the error bound on the ratio signal which is relatively much flatter. So, the variation of θ_2 did not show significant effect on the performance. For simplicity, we present results with θ_2 set to 0.01.

We compared our results on energy expenditure and lifetime with two other approaches. We used a random shortest path tree with no compression (SPT) as a base line. Since, we argue that Euclidean distance is not very useful to express the

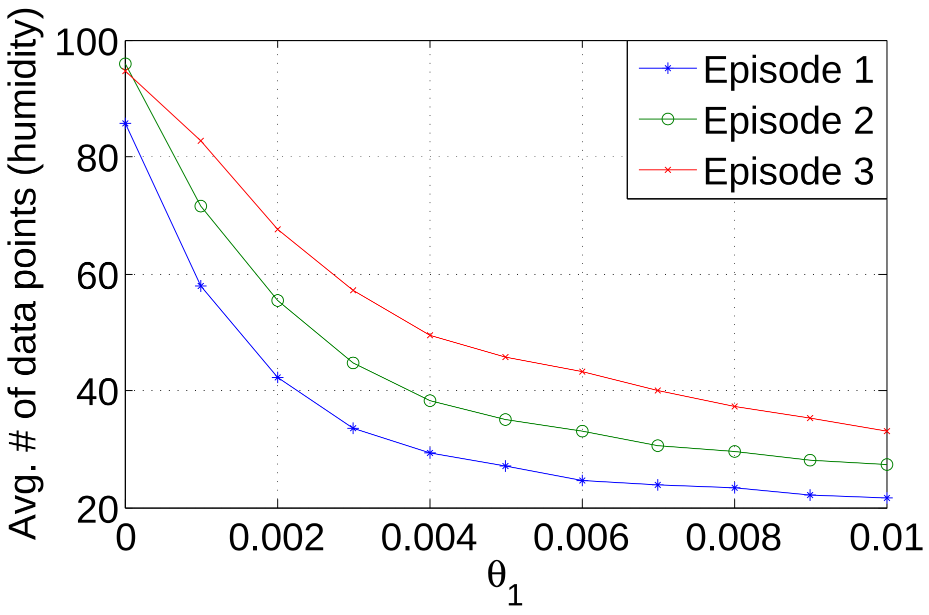

Figure 5.10: Humidity data points

varying degree of correlation, we modify our Q-learning strategy to form groups that minimizes mean distance to the nodes with base signals. We name this approach as CT-ED. To make fair comparison, we assume CT-ED uses the same bucket approximation based compression. Lastly, we employ EFFECT where the parent is selected with the Q-learning method described in the Subsection 5.4.1. We name the compression tree produced by EFFECT as CT-CR. The main difference between CT-CR and CT-ED is that, CT-CR uses the compression ratio of data streams to evaluate the reward of an action. In contrast, CT-ED assumes the data of proximate nodes are more correlated and replaces the compression ratio with the distance metric.

Finally, since bucket approximation works on streams of data, we take 10 different sets of data, where each set contains 100 measurements. We call each such set an *episode*. We apply our Q-learning algorithm with bucket approximation on one episode, and measure the performance by averaging the results over other 9 episodes.

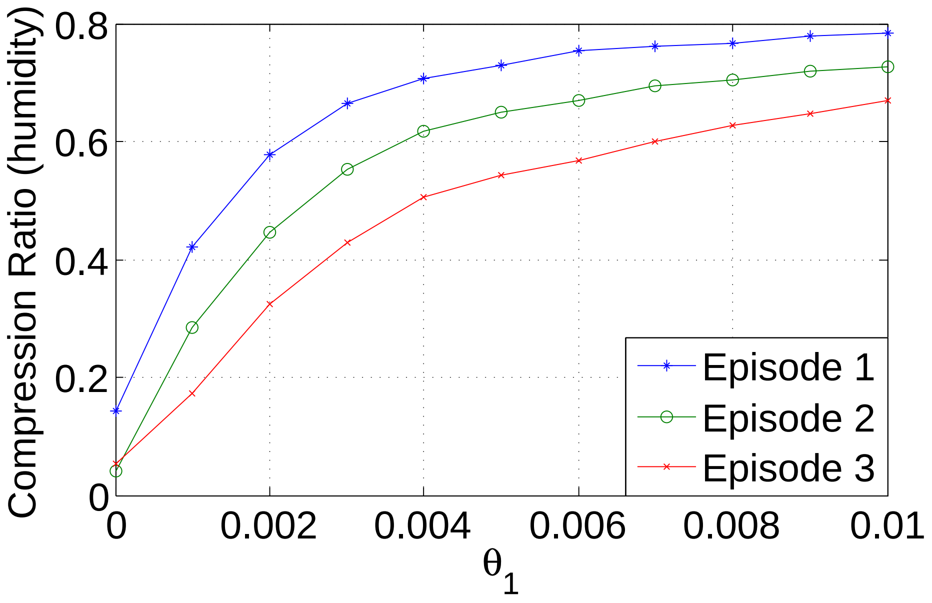

Figure 5.11: Compression ratio

5.5.2 Performance of Bucket Approximation

We first present the eligibility of bucket approximation as a compression method for data collection trees. After computing our compression tree CT-CR, we measure the average number of approximated data points per node and the average compression ratio achieved by the base signals. The results are presented with respect to different values of θ_1. We show two sets of results, one for the humidity data, and the other for the temperature data. In each set, we present the results for three different episodes.

From Figure 5.10, we observe the results for average number of data points after compression for humidity data. Note that, this actually represents the number of buckets produced by the bucket approximation while applied over an episode containing 100 original data points. We also observe that as the error bound θ_1 is relaxed, the number of approximated data points decreases. Initially, the decrement is very fast until θ_1 reaches 0.004. During this period, the number of data points reduces to about

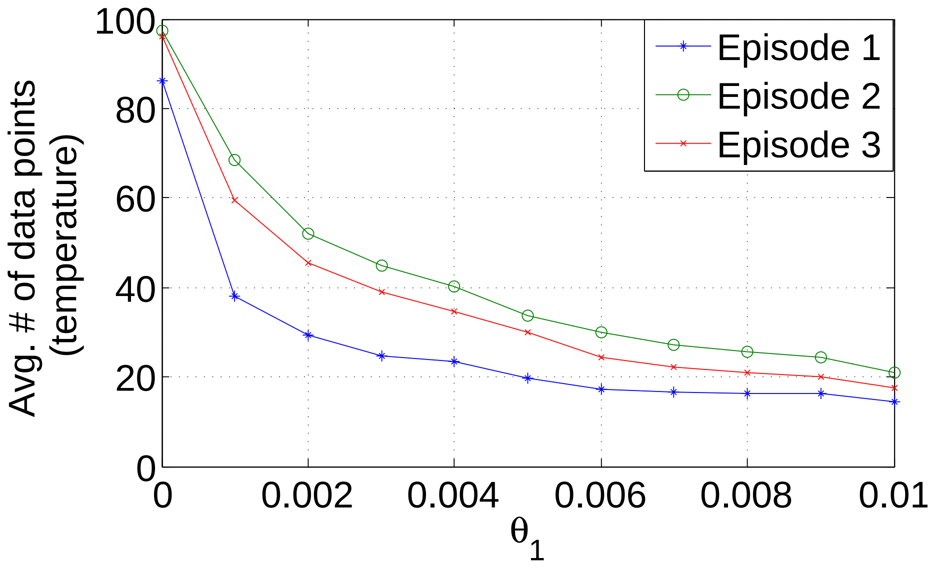

Figure 5.12: Temperature data points

50%. Increasing θ_1 further still reduces the number of data points but now the decrement rate slows down. This happens as the gain in the compression ratio tends to diminish with the relaxation of error bound as illustrated in Figure 5.11. This represents a tradeoff between the error bound and the compressibility of data. Figure 5.12 represents the average number of data points after compression for temperature data set. We observe a similar trend as in Figure 5.10. However, it appears that the temperature data set shows more correlation as the average number of compressed data points at $\theta_1 = 0.01$ is about 30% lower than that of the humidity data set. Figure 5.13 also supports this observation, as the compression ratio for the temperature data set is slightly higher.

From the above discussions, we conclude that the degree of correlation is different on different data sets due to the nonlinearity of physical data. Also, as the correlation varies with time on the same data set, we note that for different episodes, the average number of compressed data points and the compression ratios are also different.

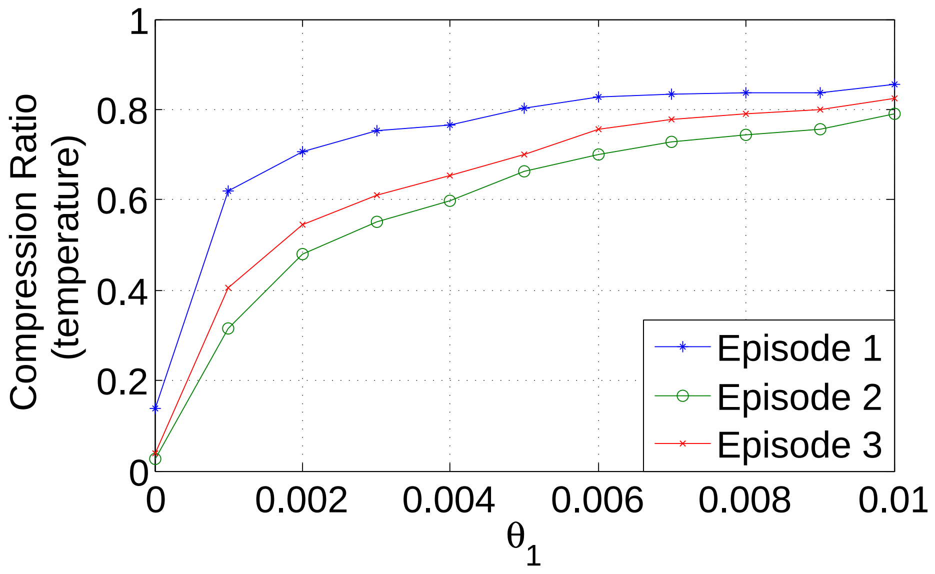

Figure 5.13: Compression ratio

5.5.3 Lifetime and Energy Consumption

We also show the network lifetime performance of CT-CR, and the average energy spent by a node per data collection round. We define *lifetime* as the time until 5% nodes die. Again we present our results for the humidity data (Set 1) and the temperature data (Set 2).

Figures 5.14 and 5.15 demonstrate the lifetime and the energy consumption results for the humidity data. Since, SPT does not employ bucket approximation, the lifetime and energy values do not change with the error bound θ_1. As Figure 5.14 shows, both CT-CR and CT-ED can produce compression trees with much higher lifetime than the SPT. This illustrates that the bucket approximation based compression trees are really useful for lifetime maximization. Note that at $\theta_1 = 0$, CT-ED results in slightly higher lifetime. However, CT-ED performs poorer than CT-CR as θ_1 increases. This is expected, since CT-ED does not consider the compression ratio while applying the bucket approximation on different episodes. As CT-ED considers the distance to be the representative of the correlation, it is unable to catch the variabil-

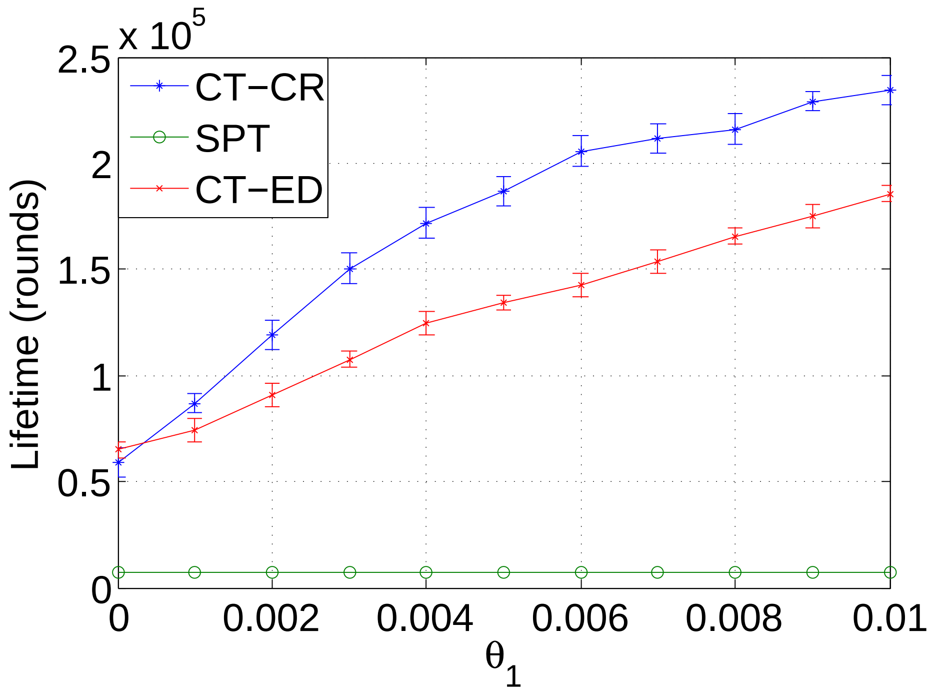

Figure 5.14: Lifetime (set 1)

ity of temporal correlation. Both CT-CR and CT-ED result in better lifetime as the error bound is relaxed. At $\theta_1 = 0.01$, the lifetime of the compression trees produced by CT-CR is about 31% higher than the lifetime of trees produced by CT-ED.

Figure 5.15 shows the average energy consumed by a node per data collection round. Once again, changing θ_1 does not have any effect on SPT. Both CT-CR and CT-ED can produce compression trees with lower energy consumption than SPT. Also, with increased θ_1, energy consumption reduces in the CT-CR and CT-ED schemes. Since CT-CR can adapt with temporal correlation, it is more efficient than CT-ED. Note that, the gap of energy consumption between CT-CR and CT-ED slightly decreases at higher values of θ_1. This happens as the relative gain in compression diminishes after a certain point. This observation is consistent with the results shown in Figure 5.11. Nevertheless, at $\theta_1 = 0.01$, CT-CR consumes about 35% less energy per node than CT-ED.

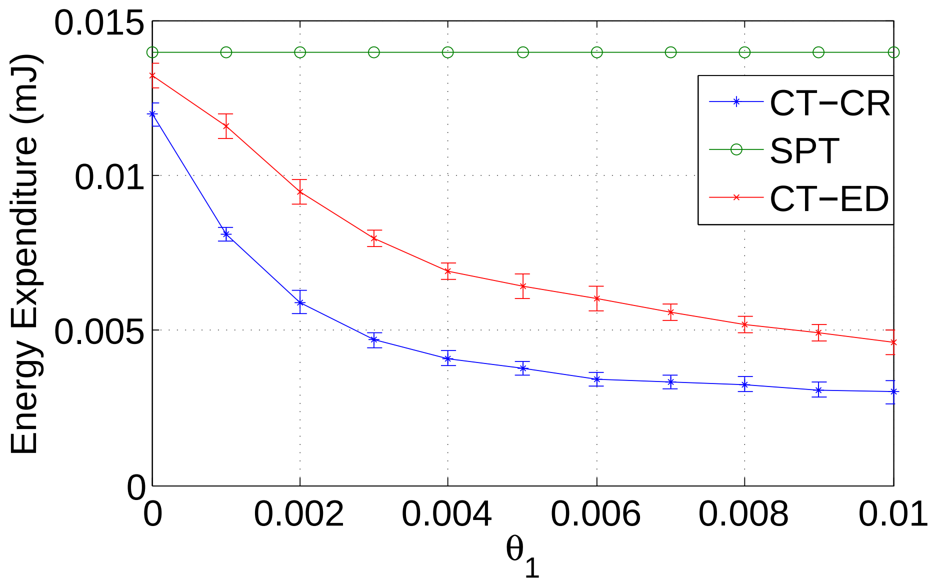

Figure 5.15: Energy consumption (set 1)

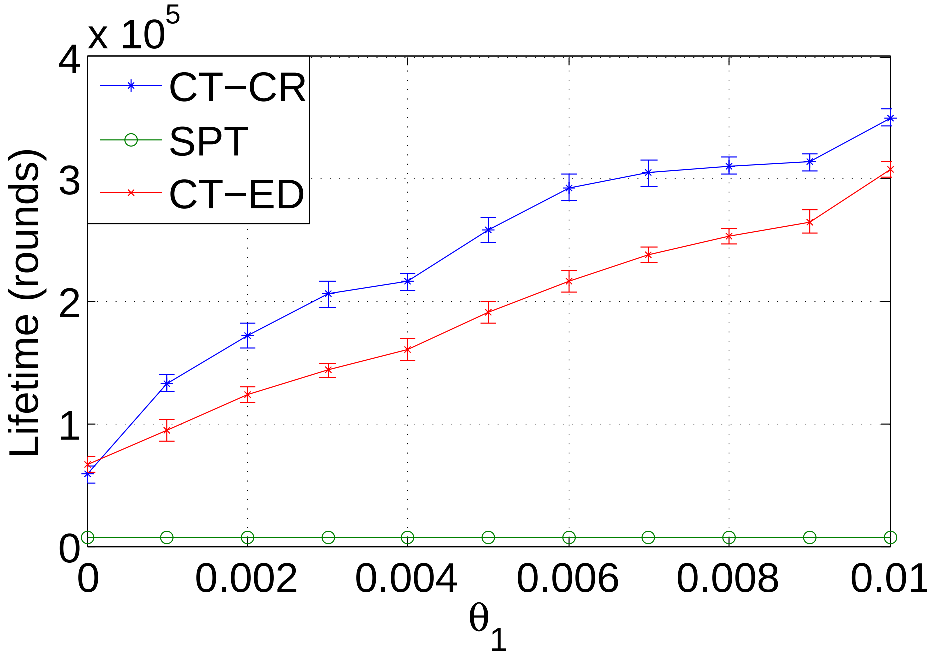

Figure 5.16: Lifetime (set 2)

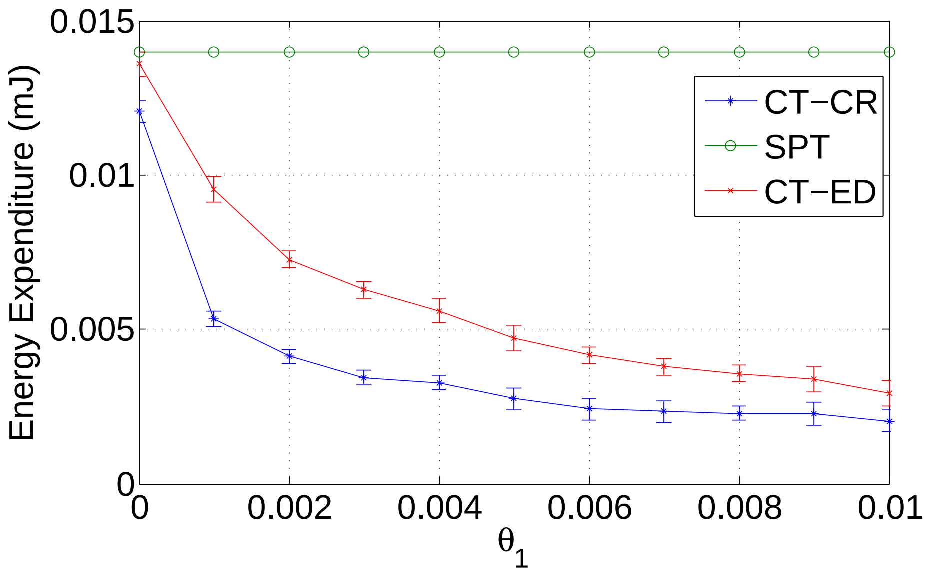

Figure 5.17: Energy consumption (set 2)

In Figures 5.16 and 5.17, we exhibit the lifetime and energy consumption values on the temperature data. Because of better compressibility of temperature data, the lifetime values are higher for both CT-CR and CT-ED. At $\theta_1 = 0.006$, CT-CR results in about 32% higher lifetime than CT-ED. However, as θ_1 increases, the relative gain is diminished, and at $\theta_1 = 0.01$, CT-CR results in just 13% higher lifetime than CT-ED. This indicates, the correlation among temperature data is more consistent with the distance metric than the correlation among humidity data. For this reason, the selection of base signals in CT-ED based on the distance performs quite well on the temperature data with larger error bound. This observation also supports the nonlinearity of the environmental data, i.e., the degree of correlation among different data sets is not the same. While the distance based CT-ED can perform well on one set of data, it can perform poorly on other sets of data. On the contrary, CT-CR is able to discover and adapt with the variation of correlation, and perform better in all cases than CT-ED. Finally, the average energy consumption is at least 22% less in CT-CR for temperature data.

5.6 Summary

In this chapter, we presented an energy-efficient compression tree framework named EFFECT that maximizes the lifetime of data collection trees in wireless sensor networks. Based on the concept of the bucket approximation for time series data, EFFECT can produce compression trees with a desired level of compression error. We provided a simple yet effective Q-learning based parent selection strategy for sensor nodes, that considers the compression ratio along with other network parameters. Incorporating the compression ratio for selection of the compression nodes intrinsically accommodates both the temporal and the spatial correlation. We conducted simulation on two real data sets, and our results confirm that leveraging dynamic correlation can significantly increase the lifetime of data collection trees with lower energy consumption per node.

As a future work, we seek to study the suitability of our framework on vector data like image, sound, etc. Furthermore, we are interested to study the applicability of our framework on multi-modal sensors.

Chapter 6

Conclusion and Future Work

In this dissertation, we disscussed the lifetime maximization problem of wireless sensor networks. From the beginning of the research in sensor networks, lifetime maximization has been a predominant research area. As different sensor application emerges and also the network size tends to grow larger this area has become even more important. We focused our study of lifetime maximization in the context of data collection trees as they are efficient routing structures for convergecast. We presented two novel frameworks for lifetime maximization, namely load balanced trees, and compression trees. We also extend our load balancing model for distributed implementation that can be used for real time tree construction and update. We validate our frameworks through analysis, extensive simulation, and also through real experiments. In the follwoing we summarize the insights we learned, and future scope of our work. From our work we present the follwoing insights on lifetime maximization of WSNs:

- *Mapping lifetime maximization to load balanced tree*: We showed that the lifetime maximization problem can be mapped to a general framework of load balanced tree problem. Such mapping helped us to develop parametrized algorithms like RaSMaLai and D-RaSMaLai.
- *Path load based non-myopic approach*: In our model, we used a novel and intelligent heuristic, namely *path load*. In our discussions from Chapters 2 and 3, we can conclude that this is indeed very effective for finding bounded balanced

trees. Path load enables our algorithms to behave in a non-myopic manner, since it gives an future estimate how good a path toward the sink is.

- *Probabilistic parent selection*: Our load balanced tree framework employs probabilstic parent selection to improve lifetime of the sensor nodes. Such selection strategy reduces number of selections made by any individual node. For the distributed implementation it results into lower energy overhead.
- *Faster convergence*: We showed that using path load based heuristic together with our novel probabilistic parent selection result into very fast convergence. For the distributed implementation this also helps to save energy over control packets.
- *Scalability*: Our load balanced tree framework is highly scalable, making it a suitable choice for large scale deployments.
- *Time series based compression model*: Our compression framework introduced in Chapter 4 is designed to accommodate time series sensory data. This gives our model flexibility to deal with dynamic correlation that is inherent to environmental data.
- *Bounded compression error*: The bucket approximation based in network compression ensures that the error introduced by compression remains bounded. This property is highly desirable when all data must be reconstructed at the sink with reliability guarantee.

6.1 Future Work

The models we present in this dissertation are shown to improve the lifetime of tree based WSNs significantly. There can be further research directions from our work.

6.1.1 Scopes for Load Balancing

One direction for the bounded balanced tree framework may be the consideration of multiple sinks for data collection. When there are multiple sinks, each sensor may have different paths to more than one sink. In such scenario, the definition of load and path load may need revised definition. Mobile nodes may be introduced to enhance connection between a sensor and a set of sink nodes. Deployment with heterogeneous sensor nodes with non-uniform data reporting frequency may make the problem even more interesting. Load balancing with coverage requirements may also be a promising topic of research.

6.1.2 Scopes for Compression Tree

The compression tree framework introduced in this thesis may be extended to incorporate multiple types of sensory data streams. In presence of different types of data how to compute a single compression ratio might be an interesting challenge. Application of bucket approximation framework on vectorial sensor data might be another interesting research direction. Finally, applicability of compression in the domain of mobile WSNs may provide interesting research challenges.

References

[1] F.-J. Wu and Y.-C. Tseng, "Distributed wake-up scheduling for data collection in tree-based wireless sensor networks," *Comm. Letters.*, vol. 13, pp. 850–852, November 2009.

[2] J. Liang, J. Wang, J. Cao, J. Chen, and M. Lu, "An efficient algorithm for constructing maximum lifetime tree for data gathering without aggregation in wireless sensor networks," in *IEEE INFOCOM*, 2010, pp. 1–5.

[3] U. Monaco, F. Cuomo, T. Melodia, F. Ricciato, and M. Borghini, "Understanding optimal data gathering in the energy and latency domains of a wireless sensor network," *Comput. Netw.*, vol. 50, no. 18, pp. 3564–3584, Dec. 2006.

[4] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 5, no. 1, pp. 5:1–5:39, Feb. 2009. [Online]. Available: http://doi.acm.org/10.1145/1464420.1464425

[5] O. Durmaz Incel, A. Ghosh, and B. Krishnamachari, "Scheduling algorithms for tree-based data collection in wireless sensor networks," *Theoretical Aspects of Distributed Computing in Sensor Networks*, 2010.

[6] G. Ghidini and S. K. Das, "Energy-efficient markov chain-based duty cycling schemes for greener wireless sensor networks," *J. Emerg. Technol. Comput. Syst.*, vol. 8, no. 4, pp. 29:1–29:32, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2367736.2367740

[7] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Netw.*, vol. 7, pp. 537–568, May 2009.

[8] T.-W. Kuo and M.-J. Tsai, "On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: Np-completeness and approximation algorithms," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 2591–2595.

[9] D. Kumar, T. C. Aseri, and R. B. Patel, "EECDA: Energy efficient clustering and data aggregation protocol for heterogeneous wireless sensor networks," *Communications*, vol. VI, no. 1, pp. 113–124, 2011.

[10] S. K. A. Imon, A. Khan, M. Di Francesco, and S. K. Das, "Rasmalai: A randomized switching algorithm for maximizing lifetime in tree-based wireless sensor networks," in *The 32nd IEEE International Conference on Computer Communications (INFOCOM 2013)*, April 2013, pp. 2913–2921.

[11] S. Toumpis and S. Gitzenis, "Load balancing in wireless sensor networks using Kirchhoff's voltage law," in *IEEE INFOCOM*, 2009, pp. 1656–1664.

[12] T. Srisooksai, K. Keamarungsi, P. Lamsrichan, and K. Araki, "Practical data compression in wireless sensor networks: A survey," *J. Netw. Comput. Appl.*, vol. 35, no. 1, pp. 37–59, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2011.03.001

[13] H. Luo, Y. Liu, and S. Das, "Routing correlated data with fusion cost in wireless sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 11, pp. 1620–1632, 2006.

[14] O. Durmaz Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi, "Fast data collection in tree-based wireless sensor networks," *IEEE Trans. on Mobile Computing*, 2011.

[15] Y. Liu, Y. He, M. Li, J. Wang, K. Liu, L. Mo, W. Dong, Z. Yang, M. Xi, J. Zhao, and X.-Y. Li, "Does wireless sensor network scale? a measurement study on greenorbs," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 873–881.

[16] S. K. A. Imon, A. Khan, and S. K. Das, "EFFECT: an energy efficient framework for data compression in tree-based wireless sensor networks," in *15th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2014)*, Sydney, Australia, June 2014.

[17] S. K. A. Imon, A. Khan, M. Di Francesco, and S. K. Das, "Analysis of RaSMaLai: A randomized approach for maximizing lifetime in tree-based wireless sensor networks," Dept. of Comp. Sci. and Eng., Univ. of Texas at Arlington, Tech. Rep. CReWMaN-2012-7, July 2012. [Online]. Available: http://crewman.uta.edu/resources/documents/technical-report/crewman-2012-7.pdf

[18] M. Di Francesco, C. M. Pinotti, and S. K. Das, "Interference-free scheduling with bounded delay in cluster-tree wireless sensor networks," in *Proceedings of the 15th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, ser. MSWiM '12. New York, NY, USA: ACM, 2012, pp. 99–106. [Online]. Available: http://doi.acm.org/10.1145/2387238.2387257

[19] S. Hariharan and N. Shroff, "Deadline constrained scheduling for data aggregation in unreliable sensor networks," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2011 International Symposium on*, 2011, pp. 140–147.

[20] G. Kasbekar, Y. Bejerano, and S. Sarkar, "Lifetime and coverage guarantees through distributed coordinate-free sensor activation," *Networking, IEEE/ACM Transactions on*, vol. 19, no. 2, pp. 470–483, April 2011.

[21] W. Choi, G. Ghidini, and S. K. Das, "A novel framework for energy-efficient data gathering with random coverage in wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 8, no. 4, pp. 36:1–36:30, Sept. 2012. [Online]. Available: http://doi.acm.org/10.1145/2240116.2240125

[22] D. Wang, B. Xie, and D. Agrawal, “Coverage and lifetime optimization of wireless sensor networks with gaussian distribution,” *Mobile Computing, IEEE Transactions on*, vol. 7, no. 12, pp. 1444–1458, Dec 2008.

[23] K. Xu, Q. Wang, H. Hassanein, and G. Takahara, “Optimal wireless sensor networks (wsns) deployment: minimum cost with lifetime constraint,” in *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, vol. 3, Aug 2005, pp. 454–461 Vol. 3.

[24] Q. Zhao and M. Gurusamy, “Lifetime maximization for connected target coverage in wireless sensor networks,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1378–1391, Dec. 2008. [Online]. Available: http://dx.doi.org/10.1109/TNET.2007.911432

[25] R. Katsuma, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, “Extending k-coverage lifetime of wireless sensor networks using mobile sensor nodes,” in *Wireless and Mobile Computing, Networking and Communications, 2009. WIMOB 2009. IEEE International Conference on*, Oct 2009, pp. 48–54.

[26] A. Chamam and S. Pierre, “On the planning of wireless sensor networks: Energy-efficient clustering under the joint routing and coverage constraint,” *Mobile Computing, IEEE Transactions on*, vol. 8, no. 8, pp. 1077–1086, Aug 2009.

[27] G. Ghidini and S. Das, “An energy-efficient markov chain-based randomized duty cycling scheme for wireless sensor networks,” in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 67–76.

[28] R. Madan, S. Cui, S. Lall, and A. Goldsmith, “Cross-layer design for lifetime maximization in interference-limited wireless sensor networks,” *Wireless Communications, IEEE Transactions on*, vol. 5, no. 11, pp. 3142–3152, November 2006.

[29] C. Vigorito, D. Ganesan, and A. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON '07. 4th Annual IEEE Communications Society Conference on*, June 2007, pp. 21–30.

[30] R. Subramanian and F. Fekri, "Sleep scheduling and lifetime maximization in sensor networks: Fundamental limits and optimal solutions," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, ser. IPSN '06. New York, NY, USA: ACM, 2006, pp. 218–225. [Online]. Available: http://doi.acm.org/10.1145/1127777.1127813

[31] F. Wang and J. Liu, "Duty-cycle-aware broadcast in wireless sensor networks," in *INFOCOM 2009, IEEE*, April 2009, pp. 468–476.

[32] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '05. Piscataway, NJ, USA: IEEE Press, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1147685.1147691

[33] J. Kim, X. Lin, N. B. Shroff, and P. Sinha, "Minimizing delay and maximizing lifetime for wireless sensor networks with anycast," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 515–528, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1109/TNET.2009.2032294

[34] W. Pak, J.-G. Choi, and S. Bahk, "Duty cycle allocation to maximize network lifetime of wireless sensor networks with delay constraints," *Wireless Communications and Mobile Computing*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/wcm.2215

[35] W. Zeng, A. Arora, and N. Shroff, “Maximizing energy efficiency for convergecast via joint duty cycle and route optimization,” in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–5.

[36] F. Yang and I. Aug-Blum, “Delivery ratio-maximized wakeup scheduling for ultra-low duty-cycled {WSNs} under real-time constraints,” *Computer Networks*, vol. 55, no. 3, pp. 497 – 513, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610003087

[37] Y. Zhao, J. Wu, F. Li, and S. Lu, “On maximizing the lifetime of wireless sensor networks using virtual backbone scheduling,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1528–1535, Aug 2012.

[38] T.-S. Chen, H.-W. Tsai, and C.-P. Chu, “Adjustable convergecast tree protocol for wireless sensor networks,” *Comput. Commun.*, vol. 33, pp. 559–570, March 2010.

[39] N. Sadagopan, M. Singh, and B. Krishnamachari, “Decentralized utility-based sensor network design,” *Mob. Netw. Appl.*, vol. 11, pp. 341–350, June 2006.

[40] A. Ranganathan and K. Berman, “Dynamic state-based routing for load balancing and efficient data gathering in wireless sensor networks,” in *CTS, 2010*, pp. 103–112.

[41] Z. Wang, E. Bulut, and B. Szymanski, “Energy efficient collision aware multipath routing for wireless sensor networks,” in *IEEE ICC*, 2009, pp. 1–5.

[42] M. Fyffe, M.-T. Sun, and X. Ma, “Traffic-adapted load balancing in sensor networks employing geographic routing,” in *IEEE WCNC*, 2007, pp. 4389–4394.

[43] Y. Wu, Z. Mao, S. Fahmy, and N. Shroff, “Constructing maximum-lifetime data-gathering forests in sensor networks,” *Networking, IEEE/ACM Transactions on*, vol. 18, no. 5, pp. 1571–1584, Oct. 2010.

[44] Y. Wu, S. Fahmy, and N. Shroff, "On the construction of a maximum-lifetime data gathering tree in sensor networks: NP-completeness and approximation algorithm," in *INFOCOM*, 2008.

[45] D. Luo, X. Zhu, X. Wu, and G. Chen, "Maximizing lifetime for the shortest path aggregation tree in wireless sensor networks," in *IEEE INFOCOM, 2011*, pp. 1566–1574.

[46] H. Dai and R. Han, "A node-centric load balancing algorithm for wireless sensor networks," in *IEEE GLOBECOM '03*, vol. 1, dec. 2003, pp. 548–552.

[47] C. Buragohain, D. Agrawal, and S. Suri, "Power aware routing for sensor databases," in *IEEE INFOCOM 2005*, vol. 3, pp. 1747–1757.

[48] Y. Wang, Y. Wang, H. Tan, and F. C. M. Lau, "Maximizing network lifetime online by localized probabilistic load balancing," in *Proceedings of the 10th international conference on Ad-hoc, mobile, and wireless networks*, 2011, pp. 332–345.

[49] A. Schumacher, P. Orponen, T. Thaler, and H. Haanpää, "Lifetime maximization in wireless sensor networks by distributed binary search," in *Proceedings of the 5th European Conference on Wireless Sensor Networks*, ser. EWSN'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 237–252. [Online]. Available: http://dl.acm.org/citation.cfm?id=1786014.1786035

[50] S.-J. Kim, X. Wang, and M. Madihian, "Distributed joint routing and medium access control for lifetime maximization of wireless sensor networks," *Wireless Communications, IEEE Transactions on*, vol. 6, no. 7, pp. 2669–2677, July 2007.

[51] V. Shah-Mansouri and V. Wong, "Distributed maximum lifetime routing in wireless sensor networks based on regularization," in *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, Nov 2007, pp. 598–603.

[52] L. Zhang, S. Chen, Y. Jian, and Y. Fang, "Distributed progressive algorithm for maximizing lifetime vector in wireless sensor networks," in *INFOCOM 2009, IEEE*, April 2009, pp. 2410–2418.

[53] F. Ren, J. Zhang, Y. Wu, T. He, C. Chen, and C. Lin, "Attribute-aware data aggregation using potential-based dynamic routing in wireless sensor networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 5, pp. 881–892, 2013.

[54] H. Luo, H. Tao, H. Ma, and S. K. Das, "Data fusion with desired reliability in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 501–513, 2011.

[55] K.-W. Fan, S. Liu, and P. Sinha, "Dynamic forwarding over tree-on-dag for scalable data aggregation in sensor networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 10, pp. 1271–1284, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1109/TMC.2008.55

[56] M. Bagaa, A. Derhab, N. Lasla, A. Ouadjaout, and N. Badache, "Semi-structured and unstructured data aggregation scheduling in wireless sensor networks," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 2671–2675.

[57] H. Luo, J. Luo, Y. Liu, and S. Das, "Adaptive data fusion for energy efficient routing in wireless sensor networks," *Computers, IEEE Transactions on*, vol. 55, no. 10, pp. 1286–1299, 2006.

[58] H. Luo, Y. Liu, and S. Das, "Distributed algorithm for en route aggregation decision in wireless sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 8, no. 1, pp. 1–13, 2009.

[59] S. Hariharan and N. Shroff, "Maximizing aggregated revenue in sensor networks under deadline constraints," in *Decision and Control, 2009 held jointly with the*

2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, 2009, pp. 4846–4851.

[60] Z. Ye, A. Abouzeid, and J. Ai, "Optimal policies for distributed data aggregation in wireless sensor networks," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1676–1684.

[61] J. Li, A. Deshpande, and S. Khuller, "On computing compression trees for data collection in wireless sensor networks," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 2115–2123. [Online]. Available: http://dl.acm.org/citation.cfm?id=1833515.1833797

[62] H. O. Tan and I. Körpeoğlu, "Power efficient data gathering and aggregation in wireless sensor networks," *SIGMOD Rec.*, vol. 32, no. 4, pp. 66–71, Dec. 2003.

[63] J. Gehrke and S. Madden, "Query processing in sensor networks," *IEEE Pervasive Computing*, vol. 3, pp. 46–55, 2004.

[64] D. West, *Introduction to Graph Theory.* Prentice-Hall, 2001.

[65] Oracle Labs, "SunSpot World," http://www.sunspotworld.com, retrieved on April 29, 2013.

[66] I. F. Akyildiz, M. C. Vuran, and O. B. Akan, "On exploiting spatial and temporal correlation in wireless sensor networks," in *Proceedings of WiOpt*, vol. 4, 2004, pp. 71–80.

[67] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *3rd International Symp. on Information Processing in Sensor Networks, IPSN 2004*, pp. 1–10.

[68] C. Buragohain, N. Shrivastava, and S. Suri, "Space efficient streaming algorithms for the maximum error histogram," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, 2007, pp. 1026–1035.

[69] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 1998. [Online]. Available: http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html

[70] S. Gandhi, S. Nath, S. Suri, and J. Liu, "Gamps: compressing multi sensor data by grouping and amplitude scaling," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 771–784. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559926

[71] D. van Velzen, R. L. Cardozo, and H. Langenkamp, "A liquid viscosity-temperature-chemical constitution relation for organic compounds," *Industrial Engineering Chemistry Fundamentals*, vol. 11, no. 1, pp. 20–25, 1972. [Online]. Available: http://pubs.acs.org/doi/abs/10.1021/i160041a004

[72] A. Forster, "Machine learning techniques applied to wireless ad-hoc networks: Guide and survey," in *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, 2007, pp. 365–370.

Biographical Information

Sk Kajal Arefin Imon was born in Khulna, Bangladesh, in 1984. He received his B.Sc. degree from Bangladesh University of Engineering and Technology, also known as BUET, in 2007. He then began his graduate studies in the Department of Computer Science and Engineering at the University of Texas at Arlington in Fall 2009. His current research interest are in Wireless Sensor Networks, Participatory Sensing, Network Modeling and Analysis, VVoIP (QoS/QoE). He has been actively involved with leadership and extracurricular activities both at the University of Texas at Arlington and in the community in DFW area.