PERFORMANCE MANAGEMENT FOR ENERGY HARVESTING WIRELESS SENSOR NETWORKS

by

Bo Zhang
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
(Computer Science)

Committee:

_____	Dr. Robert Simon, Dissertation Director
_____	Dr. Hakan Aydin, Dissertation Co-Director
_____	Dr. Jill Nelson, Committee Member
_____	Dr. Songqing Chen, Committee Member
_____	Sanjeev Setia, Department Chair
_____	Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering

Date: 04/18/2012 _____

Spring 2012
George Mason University
Fairfax, VA

Performance Management for Energy Harvesting Wireless Sensor Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Bo Zhang
Master of Science
University of Cincinnati, 2007
Bachelor of Science
Huazhong University of Science and Technology, 2004

Director: Dr. Robert Simon, Associate Professor
Co-Director: Dr. Hakan Aydin, Associate Professor
Department of Computer Science

Spring Semester 2012
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my wife, Xiaofei Pan whom without her full love, support, and sacrifice I never would have realized my full potential. I dedicate this dissertation to my grandparents and parents. There is no doubt in my mind that without their continued support and counsel I could not have completed this process.

Acknowledgments

I would like to thank the following people who made this possible, Dr Robert Simon, Dr Hakan Aydin, Dr Songqing Chen, Dr Jill Nelson, Baoxian Zhao, Lei Liu, Vinay Devadas, Anowarul Hassan, Atiq Haque, Ermo Wei, Yao Liu, Zhi Zhang, Huaming liu, Nan Li.

Table of Contents

List of Tables

List of Figures

Abstract

PERFORMANCE MANAGEMENT FOR ENERGY HARVESTING WIRELESS SENSOR NETWORKS

Bo Zhang, PhD

George Mason University, 2012

Dissertation Director: Dr. Robert Simon, Dr. Hakan Aydin

A Wireless Sensor Network (WSN) consists of spatially distributed sensor nodes which monitors environmental conditions such as temperature, humidity, sound or pressure, etc. Recently there is increasing need to design Wireless Sensor Network systems that support applications with intensive monitoring and control activities. This application class often has significant data collection and processing requirements, requiring increased levels of energy consumption as compared to other WSN applications. Further, many deeply embedded WSN systems with these data collection and processing requirements are expected to operate without manual battery recharging for several decades, and therefore require energy harvesting techniques. For this class of systems, there are currently few effective approaches that balance careful energy management with high performance communication and computation requirements.

My dissertation addresses the above problem. Specifically, I propose a set of algorithms and control methods for energy management in performance-sensitive WSN systems, and harvesting-aware rate allocation for application utility maximization. First I formally define the problem of energy harvesting-aware energy management as two optimization problems, one for individual sensor nodes and another for multi-hop sensor networks. I propose

energy management algorithm to solve both problems optimally and efficiently. These solutions combine two energy saving techniques, Dynamic Voltage Scaling (DVS), and Dynamic Modulation Scaling (DMS), alongside with energy harvesting techniques. I then address a harvesting aware rate allocation problem with the objective of utility maximization. The problem is solved with an optimal centralized algorithm and a distributed algorithm.

I conducted extensive simulation-based experiments to evaluate the effectiveness of my proposed algorithms. Specifically I developed simulation software using TOSSIM, the standard WSN simulator, and EPANET, a public domain, water distribution system modeling program. This software simulates in high fidelity the computation and communication activities of WSN nodes, and considers a variety of network setups, energy harvesting profiles (solar and water), and application scenarios, etc. My algorithms are implemented within this simulation environment and compared against a series of rival algorithms under various experimental setups. Extensive simulation results demonstrate the significant advantage of my algorithms over the rival algorithms.

Chapter 1: Introduction

There is increasing need to deploy deeply embedded Wireless Sensor Network (WSN) systems capable of lasting several decades in an unattended fashion. Examples of such applications range from structure health monitoring systems [1, 2], water distribution systems [3, 4] to underwater acoustic sensor networks [5, 6]. A major challenge faced by designers of this application class is that system lifetime is heavily constrained by the limitations of on-board battery power. As a result, there is significant interest in designing and analyzing efficient WSN energy management algorithms.

Common approaches for WSN energy management include ultra-efficient radio and chip design [7,8], energy-efficient Medium Access Control (MAC) protocol design [9–11], energy-aware routing protocols [12, 13] and dynamic voltage and modulation scaling [14, 15]. However, energy efficient algorithms by themselves are not adequate to achieve years or decades long unattended operation as the stored energy inside of a battery will ultimately be depleted. As a result, there has been a significant level of interest in designing *energy harvesting* WSN systems [16–18]. Energy harvesting is a technique to harvest energy from environmental sources, such as water flow, wind, vibration, and solar radiation to produce storable energy. For such systems a critical concept is that of *energy neutrality*: systems must consume less energy than that can be produced [16].

Because the availability of harvested environmental power is often limited and time-varying, energy neutrality cannot be achieved without deliberate management of node energy consumption. First, due to the limitation in energy harvesting availability, energy consumption must be carefully controlled not to exceed the amount of stored energy. Further, since for many environmental sources the harvesting availability varies significantly over time, energy consumption management must be carefully managed.

The challenge of designing energy-harvesting WSNs is further complicated by the fact that many embedded WSN applications such as lab-on-a-chip systems [3] and WSN-based multimedia processing [19] have significant data collection and processing requirements. As compared to earlier WSN application requirements, these intensive computational and communication requirements dramatically increase WSN energy demands. In addition to these computational and communication requirements many deeply embedded WSN applications will have unexpected workload bursts. For instance, a WSN used for monitoring water distribution systems needs to dramatically increase its activities when undergoing a biological attack. Based on the above description there is clearly a need for supporting *performance sensitive* energy harvesting systems. Moreover, many WSN applications may have stringent performance level requirements, such as deadlines for sensing task execution and data reporting, lowest acceptable sensing rate, etc. Such performance level requirements must be strictly guaranteed while performing energy management.

An important part of my research is to make combined use of two energy management techniques, Dynamic Voltage Scaling (DVS) [20] and Dynamic Modulation Scaling (DMS) [15]. The DVS technique saves computational energy by simultaneously reducing the CPU supply voltage and frequency and hence the computation speed. The DMS technique saves communication energy by scaling down the radio modulation level and hence the communication speed. For battery powered sensor nodes, the goal of energy management used to be minimizing energy consumption or maximizing lifetime. This is not suitable for energy harvesting WSNs due to unawareness of the need for harvested energy management. Observing this need, my work aims at maximizing energy reserves while meeting application performance requirements, so as to maximize the systems ability to effectively respond to an system uncertainty. Such uncertainties could be fluctuation of the system workloads or harvested environmental power which potentially result in unexpected lower energy storage.

Another important portion of my research is on utility-oriented design of energy harvesting WSN systems. The rapid introduction of new software and hardware functionalities has stimulated the development of complex WSN applications. For this new generation of

applications, maximizing the value or *utility* of sensed data, as perceived by end-users, is of paramount importance. For most WSN systems, a major constraint to utility maximization is the limited and time-varying energy availability of sensor nodes. Though utilizing energy harvesting techniques can alleviate this conflict, the goal of simple utility maximization is still constrained by the need for managing unpredictable energy supplies.

In my research, utility is related to the rate at which WSN nodes collect data from the target environment. The higher the data collection rate, the higher the utility. Unlike many existing works where utility is a linear function of rate, my research models utility as a concave function [21–23]. That is, the increase of utility value slows down as rate increases. This is because for many WSN applications increasing the level of sensed data reporting only increases the utility of the application in sub-linear fashion. Consider, for instance, video or motion sampling as part of an intrusion detection system. Since humans can only move at a certain speed, sampling above a specific threshold only marginally increases the utility of the application. The utility perceived by application end-user is the aggregate utility achieved jointly by all the nodes in the network.

As data collection rate increases, system utility can definitely increase however the energy consumption level of nodes also increases. A major design issue is the conflict between the need for utility maximization, and the limited battery capacity. This is because when utility is increased by collecting data faster, the energy consumption goes up as well potentially causing energy storage depletion. Therefore my goal is to maximize the aggregate utility over a WSN system through allocations of data collection rate to nodes, while ensuring energy neutral operations for any nodes. I formulate this goal as an concave optimization problem and propose MAX-UTILITY, an *epoch*-based rate allocation algorithm to solve it. The algorithm exploits the concavity of the utility function and a special property of tree-structured networks to allocate rates to nodes as evenly as possible, while maintaining the minimum data collection rate required by the application and the available energy and data forwarding capacity of the nodes. I formally prove the optimality of algorithm MAX-UTILITY in the sense of utility maximization. For a system with N nodes, my algorithm

has a time complexity of $O(N^3)$. I also develop a distributed version of this algorithm for use in systems without central control points. *To the best of my knowledge, this is the first optimal solution to maximize general network utility through rate assignments to individual nodes in tree-structured sensor networks while guaranteeing energy neutrality.*

1.1 Thesis Overview

The hypothesis of my dissertation can now be presented:

Thesis Hypothesis

Time and rate sensitive wireless sensor networks (WSNs) that utilize energy harvesting require new methods for effective performance engineering. Algorithms that use a joint Dynamic Voltage and Dynamic Modulation Scaling techniques will significantly improve the reliability and resiliency of such networks. Further, the sensing and reporting requirements of this class of WSNs can be effectively expressed as a concave utility function. Algorithmic techniques that exploit the special structure of this function will also substantially enhance network performance.

This dissertation describes a general device and application model for energy harvesting WSN systems. New energy management schemes utilizing DVS and DMS techniques are proposed with the objective of guaranteeing non-interruptive application services. Novel data collection rate allocation algorithm will be proposed to maximize utility of WSN systems while guaranteeing energy neutral operation for any sensor nodes. This work will be evaluated via both analytical methods and high-fidelity simulation.

My approach to harvesting-aware energy management and maximal-utility rate allocation involves several focus areas:

- An architecture of energy harvesting WSN systems which includes DVS-DMS capable WSN node model, energy harvesting model, network and application model.

- A careful quantification of the performance requirements of energy-harvesting WSN systems.
- A formal definition of utility of WSN applications.
- The use of an *epoch based* approach to manage harvested and consumed energy.
- The use of both DVS and DMS as a method for energy management.
- The use of rate allocation as a method for utility maximization.

The outcome of this work is (1) a set of algorithms and protocols capable of maximizing energy levels of sensor nodes, with an ultimate goal of improving emergency resilience of performance-senstive and mission critical energy harvesting WSN systems. (2) a set of rate allocation algorithms for maximizing application utility of WSN systems while guaranteeing energy neutrality for any sensor nodes.

1.2 Dissertation Roadmap

The rest of my dissertation continues as follows: first, I present a basic background and related work chapter (Chapter 2) on energy harvesting WSN systems, the use of DVS and DMS techniques for energy management, and utility maximization through sensing and reporting rate allocation. Next, I give the architecture of such systems including DVS-DMS capable WSN node model, energy harvesting model, tree-structured network model, and a formal definition of application utility (Chapter 3). In Chapter 4-6, I present the research work I have finished, including a series of DVS-DMS based energy management algorithms for loosely-coupled and tightly-coupled WSN systems, and rate allocation algorithms for maximizing utility of WSN systems. In Chapter 7, I propose how to implement the above algorithms on real WSN nodes as future work. Finally, Chapter 8 concludes my research work.

Chapter 2: Related Work

A wireless sensor network (WSN) comprises a set of spatially distributed nodes that are used to monitor the physical world [1, 3, 5]. Each node consists of one or more *sensors*, such as temperature, sound, vibration, chemical, etc. Each node is also equipped with a low-power wireless transceiver and micro-controller, a small amount of memory, and an energy source, typically a battery. Sensor nodes range in size from a can of soda down to the size of a grain of dust. They are meant to be easily deployable and relatively inexpensive. When deployed sensor nodes form an autonomously functioning wireless ad hoc network.

WSNs are used for many types of military, civilian and industrial applications, including battlefield surveillance [24], environmental monitoring [25], industrial process control and home automation [26]. WSN nodes generally report their readings to and receive instructions from one or more base stations. This chapter focuses on the background and directly related work in the area of performance sensitive WSNs that use energy harvesting. For a general overview of WSNs see [27].

The need for careful energy management has long motivated much activity in WSN research. Representative efforts include work in energy efficient link level protocols [11, 28], various methods of probabilistic or dynamic forms of radio and CPU duty cycling [9, 19, 29, 30], topology control [31] and in-network data aggregation [32], to name a few areas. There has also been increased interest in using both Dynamic Voltage Scaling (DVS) and Dynamic Modulation Scaling (DMS) as means of power control [14, 15, 33, 34].

Although numerous energy management techniques have been proposed to save energy and extend system lifetime, carried-on batteries have finite energy storage and will ultimately be exhausted. Further, in many circumstances it is difficult or impossible to replace the batteries. Therefore energy harvesting techniques are leveraged to sustain WSN nodes

perpetually. Because of the limited and time-varying environmental energy availability, existing energy and performance management approaches for battery-powered systems cannot be applied directly to energy harvesting WSN systems. This dissertation presents a series of harvesting-aware approaches to energy and performance management of WSN systems. One part of this work proposes DVS-DMS based energy management algorithms which maximizes energy reserve of sensor nodes, while meeting all the performance requirements of WSN applications. The ultimate aim is to improve system capacity to deal with energy shortage caused by emergency situations. This is in contrast to the traditional goal of energy management, i.e. energy consumption minimization. Another part of the work is a rate allocation algorithm which maximizes application utility of WSN systems through assignment of data collection rate to WSN nodes, while ensuring energy neutral operation. Utility maximization is of paramount importance to next-generation WSN applications, as opposed to merely maximizing throughput of data.

2.1 Performance Sensitive WSNs

The first few generations of WSNs were designed to operate with minimal performance requirements and best-effort service. The underlying assumption was that data loss and reporting delays were acceptable, under many circumstances. However, as WSN system capabilities evolve there is increasing interest in deploying applications that have stringent computational and communication requirements. Examples include systems for multimedia processing requiring actions such as in-network image analysis, distributed processing and timely communication [35], systems for micro-fluidic ("lab-on-a-chip") analysis [3], real-time systems for spot utility pricing [36] and battlefield management systems for target identification and tracking [37].

In addition to having computational and control constraints, many types of performance sensitive WSN applications should have reserved energy capacity to rapidly respond to unusual or emergency situations [38, 39]. Consider a WSN responsible for monitoring water quality for a metropolitan area such as Washington, D.C. Such system has over 1,300 miles

of pipes and if implemented would require ten's of thousands of monitoring nodes. The monitoring system would need to rapidly react throughout the pipe network to events such as a biological contamination by dramatically increasing the sampling rate, communication workload and actuator manipulation. Other WSN monitoring systems have similar requirements. One important aspect of my technical approach is addressing the need for maintaining sufficient energy capacity for emergency situations.

Another important class of performance sensitive systems will need to operate over the course of several decades. This class includes WSNs used to monitor critical infrastructures such as building health monitoring, highway maintenance, oil pipeline support and above mentioned systems for water supply monitoring and measuring [1,40]. For reasons of expense, accessibility and environmental friendliness systems in this application class will benefit from sustainable energy harvesting solutions.

One consequence of these applications increased workload demands is that power requirements are also increased. Further, even systems that use *prediction based* energy harvesting are subject to errors [16, 41]. An overestimation in energy availability (i.e. the predicted value is larger than the actual value) may induce overuse of the available energy, and potentially empty energy storage. One of the goals of my work is to maximize the energy storage level of WSN nodes, while meeting the application performance requirements.

Note that performance sensitive and real-time embedded systems have long been an area of study. Aydin et al. addressed power-aware scheduling for real-time periodic tasks using DVS. The authors first propose an offline algorithm to compute the optimal CPU frequency levels for all the tasks in the task set, while assuming a worst-case workload scenario. The derived frequency levels are optimal in that they minimize the total energy consumption, while guaranteeing the timely completion of all the tasks. Then, a dynamic reclaiming algorithm reduces CPU frequencies at run time to reclaim energy when the actual workloads of some tasks are less than the worst-case estimations. Finally, an aggressive speed reduction algorithm will further reduce the frequency levels to achieve even larger energy saving. Aydin et al. target system-level energy management for real-time periodic tasks.

The authors considered the CPU frequency-independent powers (or off-chip powers) that are consumed by memory access, I/O operations etc., in addition to frequency-dependent power consumption (on-chip power) by CPU, and determined that the system-level energy consumption does not monotonically decrease with the frequency level. They then formulate the energy minimization problem as a non-linear optimization problem, and solve it optimally by manipulating the Karush-Kuhn-Tucker optimality conditions. [42] and [43] focus on scheduling periodic computational tasks in a power aware manner. Both techniques perform well for systems which the computational energy dominates the overall energy consumption. However, their computation-oriented task model ignores communication energy, and is unsuitable for WSN systems.

Furthermore, [44] solved a reward maximization problem for real-time embedded systems utilizing energy harvesting. They considered real-time tasks with multiple versions, where each version has different workload demand and reward. The reward of a task version is modeled as a concave function of its required workload demands. Then, depending on the predictions of the amount of harvested power, their solution determines for each task a task version and task execution speed. While ensuring the timely completion of all the tasks, and also that the consumed energy is no larger than the harvested energy, the derived task versions and CPU frequencies maximize the total rewards obtained. Though [44] considered a simple energy harvesting model, it is one of the earliest works that exploiting the use of DVS for energy harvesting systems. Moreover, they assumed a system consisting of real-time tasks that are executed within periodically invoked frames, which motivated the use of frame-based task model in my work.

2.2 DVS and DMS for WSN Energy Management

Dynamic Voltage Scaling (DVS) and Dynamic Modulation Scaling (DMS) are two energy saving techniques widely used in wireless embedded systems. It is natural to consider these techniques for WSN energy management. With DVS the processor is operated at

a low supply voltage/frequency levels in order to save energy [45, 46]. DMS reduces the data transmission rate for a specific modulation technique, in order to exploit the convex dependence between the transmission rate and power [14, 15, 47], and thus saving energy. Both DVS and DMS techniques have been recently fine-tuned by the research community by incorporating frequency-independent or rate-independent power components and avoiding energy inefficient operation points [15, 43, 48].

As an example of related work in DVS and DMS, Kumar et al. addressed a resource allocation problem with the aim of minimizing the overall CPU and radio energy consumption[14]. They assume a system containing a mixed set of computation and communication tasks. An initial speed schedule is assumed to be given. This initial speed schedule preserves the precedence relation between tasks, and satisfies applications timing constraints. By analyzing the power consumption characteristics of different tasks and also the available time before deadlines, the overall energy consumption is minimized by adjusting compute and communicate speeds. In [49], the energy management problem is formulated as a convex optimization problem. Genetic algorithms are proposed to find out the optimal computation and communication speeds for maximum energy savings, while ensuring the timely completion of the workloads.

Both [15] and [50] explored the use of DMS in multi-hop wireless sensor networks. In [50], Yu et al. proposed a DMS-based approach for a multi-hop chain of sensor nodes. Their goal is to balance the energy consumption of sensor nodes on the chain by scaling the communication speed, while ensures timely routing of data between the two ends of the chain. In [15], Yu et al. extended their approach to tree-based WSNs. They assumed a data gathering application in which a remote base station periodically collects sensed data from sensor nodes over a tree-based routing structure. They proposed centralized and distributed algorithms to minimize the total energy consumption across all nodes in the network by adjusting nodes communication speeds, while guaranteeing that the base station gathers sensed data from all the nodes within an end-to-end timing constraint.

The work presented in [14,15,49,50] all assume battery-powered systems without energy

harvesting capabilities. Their ultimate goal for using DVS and/or DMS is to prolong system lifetime by reducing energy consumption, without considering issues such as ensuring perpetual operation through energy harvesting. Further, [15] and [50] consider only communication energy, while ignoring the fact that computational activities are quite time and energy consuming in modern WSN applications.

2.3 Energy Harvesting

The current generation of WSN nodes are generally powered by batteries. A typical TelosB node, using two AA cells, will exhaust its power supply after a few days running at a full duty cycle. As a result a number of prototype energy harvesting WSN nodes had been designed and implemented. In [16], the authors described *Heliomote*, a solar powered sensor node. Heliomote harvests solar energy to sustain its operation during daytime, the unused energy is stored in batteries for use in the evening. The work described in [16] is the first to propose a theoretical model of environmentally powered sensor networks. The authors of [17] discuss *Prometheus*, which utilizes a two-level energy storage architecture. Its first-level energy buffer consists of a super-capacitor which is characterized by its slow deteriorating process and large energy storage capacity. These two properties make super-capacitors the ideal primary energy buffer which may experience high energy throughput. The second-level buffer consists of a Lithium battery which have the benefit of low power leakage, and hence can hold harvested energy longer. The authors claim that Prometheus can survive 43 years under a 1% load, and 4 years under a 10% load, and 1 year under a 100% load. In [18], the authors proposed *Everlast*, a super-capacitor operated sensor nodes. Everlast adopted a power transfer hardware which enables the solar cells to charge the super-capacitors at the maximum power point, hence achieves high power transfer efficiency.

Many existing studies have explored algorithmic support for energy harvesting WSNs. In [33], Moser et al. proposed the LSA algorithm (Lazy Scheduling Algorithm) for scheduling periodic real-time tasks in the context of energy harvesting. They noticed that completing tasks before their deadlines has no benefit, but rather steals energy from successive tasks.

This potentially leads to future deadline misses due to overhasty energy usage. Motivated by this, the LSA defers task execution as late as possible (but before deadline), thus avoids unnecessary deadline misses. Liu et al. ([34]) proposed EA-DVFS (Energy-Aware Dynamic Voltage and Frequency Scaling) which improves the energy efficiency of LSA algorithm by using DVS, thus further reducing deadline misses. The EA-DVFS achieves this by reducing the computation speed when high speeds could potentially leads to energy depletion. However, both LSA and EA-DVFS manage only the CPU energy, while ignoring radio energy.

Other research ([51,52]) considered the maximization of application performance. Specifically performance is correlated with sensing and reporting rate, task workload in [52] and [51] respectively. Both of these papers formulate the problem in terms of an optimization goal, with an objective of maximizing performance. An important constraint is to ensure that nodes energy reserve never drops to zero. The authors then solve the problem using standard optimization algorithms. The ultimate goal of these works is to fully utilize the harvested energy in order to maximize system performance. I notice that many WSN applications do not have this requirement. Rather, the number of sense tasks and associated computation and communication activities are fixed by the environment and application. Additionally, they consider only individual nodes, and cannot be extended to a multi-hop network environments.

Additional related work includes [53] and [54], which target harvesting-aware design for multi-hop sensor networks. The work in [53] proposed a distributed approach for finding the optimal lexicographic rate assignment in the context of multi-hop, energy harvesting sensor networks. They considered a tree-based multi-hop network with a set of data source nodes that periodically generates data packets, and forwards them to a base station. In [54], the authors designed a novel energy harvesting aware routing algorithm for sensor networks with renewable energy. Their work is distinguished from the existing WSN routing schemes in that it is aware of the spatial differences in nodes energy harvesting abilities and instant residual energy storage. By considering the disparity among nodes energy availability, they

defined a harvesting aware cost function, and apply the classic shortest-path algorithm for determining the routing paths. The goal is to balance routing load and hence resulted energy consumption among sensor nodes all over the network, so as to avoid overloading and exhausting a small group of "hot-spot" nodes.

Finally, [55] and [56] propose energy budgeting algorithms. An energy budget defines the maximum energy that can be consumed by a node over a given time interval. Their goal is to reduce the variance of energy budget over time. Specifically, [55] describes a probabilistic observation approach for harvesting solar energy that attempts to minimize energy allocation variance among nodes. [56] presented an approach for reducing variance of node's duty cycle using adaptive control theory. The motivation of both studies is particularly important to applications that requiring operating under a constant performance level.

The fundamental difference between the above work in energy harvesting and my proposed work is that I focus on mission-critical and performance sensitive applications.

2.4 Rate allocation for utility maximization

Rate allocation for wireless sensor networks has been explored in [21,23,41,53,57]. The work proposed by [53] is reviewed in 2.3. [41] proposes a rate control approach for a single energy harvesting node to achieve a series of objectives including the maximization of average sensing rate over time. These objectives are formulated as optimization problems and solved using multi-parametric control algorithms. [55, 57] propose a flow control algorithm for energy harvesting WSNs. As mentioned in 2.3, [55] finds an energy budget assignment that minimizes the variance of energy assigned over time. Using this formulation the flow control algorithm in [57] maximizes the amount of data collected over the network, given that no node consumes more energy than the assigned budget.

An implicit assumption in [41, 53, 57] is that the system utility increases linearly with the rate of nodes. However, for many applications the increase of utility slows down as rate

increases (the *diminishing returns* principle). Using this observation, [21–23] model system utility as a concave and non-decreasing function of rate, and propose primal-dual based algorithms to maximize utility over the entire network. Among these works, [21] is the most related one to mine that target utility maximization for energy harvesting WSNs. However, [21–23] all assume specific utility functions that are continuously differentiable, which may limit their application for a more general class of functions. Moreover, their proposed solutions are not optimal, and can incur high control overhead and unpredictable running time, thus potentially limiting their practical implementation within resource-constrained WSN systems.

Research presented in [58,59] formulate the Network Utility Maximization (NUM) problem for Internet congestion control. Primal-dual based algorithms are proposed to solve the problem. The issue I target shares the same structure with the NUM problem, but is for a WSN environment. The utility maximization problem has been also extensively studied for real-time embedded systems. [60] addresses utility maximization for energy-constrained systems that execute periodic real time tasks. The objective is to maximize the total utility obtained from execution of these tasks, while satisfying the deadlines of all the tasks, the tight system energy budget and the minimum system performance requirement. In [44], the authors extend the approach in [60] for solar-powered embedded systems. However, [44] only considers two different epochs in each day. [51, 61] address a similar problem as [44] but with a more complex system model. They assume a highly dynamic energy harvesting model, and assume that WSN applications have multiple discrete service and performance levels, and each has different utility values and energy demands. The problem is then to decide when to select which service level to run in order to maximize the total utility, without over-using the available energy.

Chapter 3: Device Model

This chapter describes the basic sensor node model used within the dissertation. Without loss of generality, I assume that each node has several functional units, including an energy harvester head, an energy storage module, a DVS capable CPU, a DMS capable radio, as well as required sensor suites. The harvester head is energy source-specific, such as a solar panel or wind generator. This model is shown in Figure 3.1. The rest of this chapter provides the technical details of this model.

3.1 Energy Consumption Model for DVS-DMS Capable Wireless Sensor Nodes

The node consumes power via either processing, wireless communication or sensing. I now describe how to model energy consumption for an individual node. The basic time interval over which energy consumption is calculated is called a *frame*, defined differently in 4 and 5. I assume each DVS-enabled CPU has m discrete frequencies $f_{min} = f_1 < ... < f_m = f_{max}$, and each DMS-enabled radio has n discrete modulation levels, $b_{min} = b_1 < ... < b_n = b_{max}$. I use the terms frequency and computation speed interchangeably. In practice, the modulation level represents the number of bits encoded in one signal symbol [15]. To understand this relationship, let R be the fixed symbol rate. Then modulation level b is associated with communication speed d expressed as:

$$d = R \cdot b \tag{3.1}$$

Let e^{sen} represents the energy required for each sensing which is a constant. The computation energy is a function of the computation speed f and supply voltage V_{dd} [42].

Communication energy is a function of the communication speed d [15]. Let e^{cp} and e^{cm} represent the computation and communication energy in a frame, respectively, and are given below:

$$e^{cp} = [\alpha f V_{dd}^2 + P^{ind,cp}] \cdot (C/f) \tag{3.2}$$

$$e^{cm} = [\beta R(2^{d/R} - 1) + P^{ind,cm}] \cdot (M/d) \tag{3.3}$$

Above, C and M are the computation and communication workloads in a frame. C is the number of CPU cycles to be processed, while M is the number of bits to be transmitted. The α in Eq. (3.2) is the CPU switching capacitance, which is a constant. The β in Eq. (3.3) is a constant determined by the transmission quality and noise level [15]. The terms $\alpha f V_{dd}^2$ and $\beta R(2^{d/R} - 1)$ give the *speed-dependent power* of CPU and radio which vary with f, V_{dd} and d. $P^{ind,cp}$ and $P^{ind,cm}$ are two constants representing the *speed-independent power* of CPU and radio. By using DVS, the supply voltage V_{dd} can be reduced linearly alongside with f to obtain energy saving (i.e. $f \propto V_{dd}$), making the speed-dependent CPU power a *cubic* function of f. My model assumes a sufficient level of coordinated sleeping and transmission scheduling, so that the radio energy consumed by listening channel activities is not a significant factor. Finally, the total energy consumed in a frame, e^c is given by:

$$e^c = e^{sen} + e^{cp} + e^{cm} \tag{3.4}$$

3.2 Energy Harvesting and Storage Model

The energy storage unit (e.g. rechargeable battery or super-capacitor) has a maximum energy capacity of Γ^{max} joules. This unit receives power from energy harvester, and delivers power to the sensor node. I take the commonly used approach that the amount of harvested power is uncontrollable, but reasonably predictable, based on the energy source type and

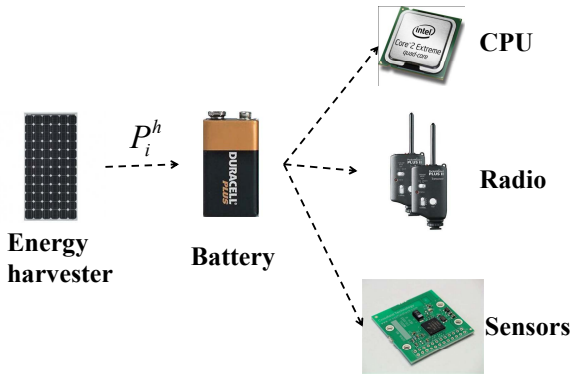


Figure 3.1: Device model of an energy harvesting sensor node

harvesting history [16]. To capture the time-varying nature of environmental energy, time is divided into *epochs* of length S. Harvested power is modeled as an epoch-varying function denoted by P_i^h, where i is the epoch sequence number. P_i^h remains constant within each epoch i, but changes for different epochs. To be precise, P_i^h is the actual power received by energy storage including the loss during power transfer from energy harvester to energy storage, and the power leakage of energy storage. The time unit used for harvesting prediction is one epoch. The *prediction horizon*, H, is an interval comprising a number of epochs for which predictions can be reasonably made. The system needs to know the harvested power prediction of only the coming epoch, at the epoch start.

When a sensor node is executing tasks, it draws power from the energy storage. When the energy level drops to zero, the energy storage is forced to stop discharging. On the contrary, when the energy level approaches Γ^{max}, the energy storage stops charging to avoid energy overflow. In [16], the authors proposed the concept of *energy neutrality* condition which basically requires sensor nodes to maintain positive energy storage at all time. This is a necessary condition for an energy harvesting sensor node to operate non-interruptively.

Depending on the types of energy storage device, power usage and power harvesting may happen concurrently or non-concurrently. Several papers assumed that concurrent harvesting/usage capability is commonly available ([17], [16]). However, [62] pointed to the need for special hardware mechanisms to separate charge and discharge currents, which may be expensive for sensor nodes. In such systems, energy cannot be consumed (i.e. no sensing, computation or computation activity can take place) while harvesting energy.

3.3 Performance-Sensitive WSN Applications

In my work the basic unit for WSN application execution is referred as *task*. The sensor nodes execute task within periodic invoked frames. There are a total of three task types: *sensing*, *computation* and *communication*. I assume in each frame that a sensor node performs sensing first, then processes the sensor reading (computation), and finally transmits the processed data (communication).

In order to model a time-critical WSN application, I assume the execution of tasks must be completed within a specified deadline. For low-power, multi-hop WSN applications where end to end data delivery delay is often excessively large, this requirement is necessary to guarantee the freshness of sensed data.

I model two types of time-critical WSN systems, loosely-coupled and tightly-coupled systems. Due to the differences in how sensor nodes are coupled, timing requirement for these two types of systems are different. For loosely-coupled systems, the task execution deadline is specified for individual nodes while for tightly-coupled system where distributed nodes must collaborate to accomplish a joint task, it makes more sense to specify an end to end deadline. The formal definition of *deadline* for both system types is given in Chapter 4 and 5 respectively. Moreover, Chapter 4 and 5 describes the ways for calculating task execution delay at individual nodes and the end-to-end delay.

3.4 Utility-Oriented WSN Applications

The rapid introduction of new software and hardware functionalities has stimulated the development of complex wireless sensor network (WSN) applications. For the next-generation WSN applications, maximizing *utility* that are perceived by end users is of paramount importance. I model utility as a function of how frequently sensor nodes sense and report sensor readings. Much previous work in this area assumes that utility increases linearly with sensing and reporting rate [41, 52, 53, 57], therefore the maximization of the network-wide data collection rate leads to utility maximization.

However, for many WSN applications increasing the level of sensed data reporting only increases the utility of the application in sub-linear fashion. Consider, for instance, video or motion sampling as part of an intrusion detection system. Since humans can only move at a certain speed, sampling above a specific threshold only marginally increases the utility of the application. For this reason, I models utility as a non-decreasing concave function, such that its rate of increase (*marginal utility*) decreases as the sensing and reporting rate increases [21–23]. The utility perceived by application end-user is the aggregate utility achieved jointly by all the nodes in the network. The formal definition of utility is given in Chapter 6.

The remaining three chapters present my research. Chapter 4 and 5 propose DVS-DMS based energy management solutions for time-critical energy harvesting WSN systems. Chapter 4 targets loosely-coupled systems while Chapter 5 targets tightly-coupled systems. Chapter 6 addresses utility maximization for energy harvesting WSNs through rate allocation.

Chapter 4: Joint DVS-DMS Energy Management for Loosely-Coupled Systems

To accurately define the problem, I first give the task model for a single node scenario. Due to the simplicity in managing sensor activities and energy usage, I organize the operations of a sensor node as periodically invoked tasks. I model three sensor operations: sense, computation and communication. A *sense* operation measures a physical quantity and generates raw reading. A *computation* operation may involve processing the raw data, aggregating data from other nodes or performing management of networking activities. A *communication* operation involves sending or receiving data packets. I will denote each one of these three basic operations as a subtask.

The above three subtasks are combined to form three task types. Sense subtasks are performed periodically and most frequently by a sensor node, followed by compution and then communicatation. Based on this observation the three task types are *sense-only* (SO), *sense-computation* (SC), and *sense-computation-communicatation* (SCC). I refer to one invocation of task as a task instance. A new task instance is invoked every π time units (i.e., with period π) and its type is known when it arrives. The task instances are executed on *frame* basis [44]. A frame refers to a time interval of length π during which a task instance is invoked, executed and completed. In order to maintain acceptable system performance, each task instance must be completed within the frame period π. For example, the jth task instance is invoked at the beginning of the jth frame (i.e. at time $(j-1)\cdot\pi$) and must complete its execution within that frame (i.e. by time $j\cdot\pi$). Since each frame contains only one task instance, I use the terms *frame* and *task instance* interchangeably for convenience.

The tuple $\{I_j^{sen}, I_j^{cp}, I_j^{cm}\}$ identifies the type of a frame j. The elements in the tuple are binary-valued with "1" indicating the existence of the specified subtasks, and "0" if

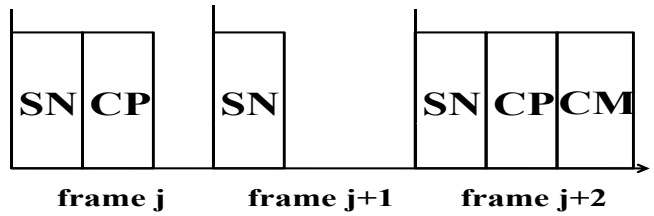


Figure 4.1: A sequence of frames of different types

not. For instance, a sense-computation frame with frame type {1,1,0} consists of a sense subtask followed by a computation subtask, but no communication. I assume the types of frames are determined by the application. Fig. (4.1) illustrates a sequence of frames of different types. SN, CP and CM represent sense, computation and communication subtasks respectively.

Using the notation from Chapter 3, the execution time of the computation and communication subtasks in frame j equal C/f_j and M/d_j respectively. I make a common assumption that the effective data transmission time dominates the overall communication time thereby enabling us to ignore the carrier sense time [14, 15, 49]. Sensing time is denoted as a constant t^{sen}. The total execution time t_j^{exe} in frame j of type $\{I_j^{sen}, I_j^{cp}, I_j^{cm}\}$ at computation speed f_j and communication speed d_j is given as:

$$t_j^{exe} = I_j^{sen} \cdot t^{sen} + I_j^{cp} \cdot (C/f_j) + I_j^{cm} \cdot (M/d_j) \tag{4.1}$$

4.1 Energy Management with Energy Harvesting

Energy management is required to improve the systems resilience to emergency or fault-driven situations which may deplete a nodes energy storage. At the same time we still need to meet the applications performance (timing) requirements. Such energy depletion might result from many system uncertainties, e.g. workload burst or errors in prediction of harvested power. Motivated by this, I aim at *maximizing the minimum energy level observed over time*. The benefit of this goal is that the extra energy can be used to service unexpected workloads or new application tasks that are introduced into the system, and to cover epochs or time periods where the amount of harvested power is less than expected.

In an attempt to capture all the parameters of the problem, I start with a couple of definitions. This will allow me to formulate the problem in a precise manner. The energy level at the end of epoch i Γ_i is given by:

$$\forall i \in [1, N], \Gamma_i = \Gamma^{init} + \sum_{k=1}^{i} E_k^h - \sum_{k=1}^{i} E_k^c \tag{4.2}$$

where Γ^{init} is the initial energy level in a horizon. N is the number of epochs in a prediction horizon. E_k^h and E_k^c are the harvested and consumed energy in epoch k respectively. Starting with Γ^{init}, Γ_i may increase or decrease depending on the consumed and harvested energy in intermediate epochs.

In my epoch-based approach, the jth frame of an epoch i is denoted by the pair (i, j). Now, within an epoch i, the energy level at the end of a frame j, is:

$$\gamma_{i,j} = \Gamma_{i-1} + \sum_{k=1}^{j} e_{i,k}^h - \sum_{k=1}^{j} e_{i,k}^c \tag{4.3}$$

In other words, starting with the ending energy level Γ_{i-1} in epoch $i-1$ (which is also the starting energy level in epoch i), $\gamma_{i,j}$ is determined by the harvested and consumed energy

in frame (i, j) and all its preceding frames, $e^h_{i,k}$, $e^c_{i,k}$, $k \in [1, j]$. $e^h_{i,k}$ is defined in 3.2, 3.3 and 3.4 while $e^h_{i,k}$ is defined later in this section.

Note that there are $\lfloor S/\pi \rfloor$ frames in an epoch. S is defined in 3.2. The consumed energy in an epoch, E^c_i is given as:

$$E^c_i = \sum_{j=1}^{\lfloor S/\pi \rfloor} e^c_{i,j} \tag{4.4}$$

$e^c_{i,j}$ is the energy consumption in frame (i, j) (Eq. (3.4)). The harvested energy E^h_i is given as:

$$E^h_i = \sum_{j=1}^{\lfloor S/\pi \rfloor} e^h_{i,j} \tag{4.5}$$

$$e^h_{i,j} = P^h_i \cdot t^h_{i,j} \tag{4.6}$$

$e^h_{i,j}$ is the harvested energy in frame (i, j). As mentioned in 3, the harvested power P^h_i is a known constant and fixed over all frames in epoch i. $t^h_{i,j}$ is the effective energy harvesting time in frame (i, j). In concurrent harvesting model, the system can continuously harvest power throughout a frame, hence:

$$t^h_{i,j} = \pi \tag{4.7}$$

On the other hand, in non-concurrent harvesting model, task execution and energy harvesting cannot occur concurrently. Since $t^{exe}_{i,j}$ is the total execution time in frame (i, j) (Eq. (4.1)), then $t^h_{i,j}$ equals:

$$t^h_{i,j} = \pi - t^{exe}_{i,j} \tag{4.8}$$

At this point, I can start formulating my objective as an optimization problem. The

objective is to maximize the minimum energy level over all the frames in a horizon:

$$\gamma_{min} = min\{\gamma_{i,j} \mid \forall i \in [1, N], j \in [1, \lfloor S/\pi \rfloor]\} \tag{4.9}$$

The variables of the problem are the computation and communication speeds $f_{i,j}$, $d_{i,j}$, used in any frame (i, j) in the horizon. Recall that by managing $f_{i,j}$ and $d_{i,j}$, one can adjust the harvested and consumed energy and hence regulate the energy levels. Thus, I need to determine the optimal speeds $f_{i,j}$, $d_{i,j}$ for each frame (i, j) in the horizon that achieve this objective. I will later show that the optimal communication speed $d_{i,j}$ is unique for a given (entire) epoch (i.e. it does not change from frame to frame). Similarly, it will turn out that for a given epoch one needs to derive only two computation speeds (one for SC and one for SCC frames, respectively).

The optimization problem is called *Energy Management with Energy Harvesting (EMEH)* and given by Eq. (4.10) to (4.14).

$$Max. \quad \gamma_{min} \tag{4.10}$$

$$s.t. \quad \forall i \in [1, N], j \in [1, \lfloor S/\pi \rfloor]$$

$$0 < \gamma_{i,j} \leq \Gamma^{max} \tag{4.11}$$

$$t_{i,j}^{exe} \leq \pi \tag{4.12}$$

$$f_{min} \leq f_{i,j} \leq f_{max} \tag{4.13}$$

$$d_{min} \leq d_{i,j} \leq d_{max} \tag{4.14}$$

The objective is maximizing γ_{min} 4.9. The constraint (4.11) enforces that the energy level $\gamma_{i,j}$ in any frame is confined to the range $(0, \Gamma^{max}]$. $\gamma_{i,j} > 0$ must hold in order to ensure energy neutrality. Also, I require that $\gamma_{i,j} \leq \Gamma^{max}$ to model the energy storage capacity of the sensor node. The constraint (4.12) ensures the timely completion of workloads in a given frame. The constraints (4.13) and (4.14) give the lower and upper bounds for computation

and communication speeds, respectively.

Notice that the problem *EMEH* is essentially a non-linear program, because the frame energy level $\gamma_{i,j}$ (Eq. (4.3)) depends on the non-linear energy consumption function, $e^c_{i,j}$ (Eq. (3.4)). My strategy to solve this problem is given as follows. I first focus on designing an energy management algorithm for any single, given epoch with known initial energy level and harvested power. Then, I show that by iteratively invoking this algorithm for each epoch I can solve the horizon-based problem *EMEH* optimally. I start by proposing Theorem 1 as follow.

Theorem 1. *Starting with arbitrary initial energy level in an epoch i, iteratively maximizing the increment of energy level of each frame (i, j), $\Delta\gamma_{i,j} = \gamma_{i,j} - \gamma_{i,j-1}$, $j \in [1, \lfloor S/\pi \rfloor]$ beginning with the first frame, maximizes the energy level at the end of any frame in epoch i.*

The proof for this theorem is given in the Appendix. Since applying Theorem 1 maximizes $\gamma_{i,j}, \forall j \in [1, \lfloor S/\pi \rfloor]$ in epoch i, the following corollary is easily justified.

Corollary 1. *Iteratively maximizing the energy level increment in each frame (i, j), $\Delta\gamma_{i,j}$, maximizes the minimum energy level observed in any frame of epoch i.*

Theorem 1 implies the existence of an algorithm which maximizes the energy level at the end of any epoch i, Γ_i by greedily accumulating energy over each frame in epoch i. I refer to this optimal algorithm as *DVMS*.

Then, I can now observe the following. Starting with the initial energy level in the horizon, i.e. Γ^{init}, the ending energy level in epoch 1, Γ_1 is maximized by invoking algorithm *DVMS* for epoch 1, which in turn supplies the maximum possible initial energy level for epoch 2. The same reasoning would apply to the 2nd, 3rd,...,N^{th} epochs as well, as long as the new harvesting rate is fed into the algorithm *DVMS* at the start of each new epoch. Therefore, I conclude that by iteratively invoking algorithm *DVMS* for each epoch, I can achieve the objective of problem *EMEH* which maximizes the minimum energy level observed in any frame of the horizon, γ_{min}.

The optimality of DVMS and Corollary 1 also imply the following:

Corollary 2. *If the algorithm DVMS cannot find a feasible solution to a specific instance of the problem EMEH, then that instance does not admit any feasible solution.*

Finally, I notice that violation of the constraint $\gamma_{i,j} \leq \Gamma^{max}$ will never happen in practice, simply because the energy harvester is assumed to stop charging the storage device when the energy level approaches the capacity Γ^{max}.

4.2 Epoch-Based Energy Management

While maximization of energy increments over consecutive frames is optimal as indicated by Theorem 1, I still need to determine the optimal computation and communication speeds to achieve that objective. Since the harvesting rate changes only from epoch to epoch, a natural strategy is to solve the problem for each epoch separately. In this way I can focus on designing the epoch-based algorithm *DVMS*. As mentioned in 4.1, the basic idea of algorithm *DVMS* is to accumulate as much energy as possible in each frame of a given epoch. This will lead to the maximum possible stored energy at the end of the epoch. I achieve this objective by iteratively solving a Single-Frame Energy Management (*SFEM*) problem for each frame in the epoch.

The problem *SFEM* has effectively two variants. In my analysis, I consider only the solution for the SCC frame because it is the most general one; the SC type is a special case of the SCC type where $M = 0$. Note that since its energy consumption function is not controllable through DVS and DMS, I do not include SO frames in the analysis. Although one epoch contains $\lfloor S/\pi \rfloor$ frames, I claim that the above problem needs to be solved only once for the first SCC frame in each epoch, that is, the optimal computation and communication speeds derived can be fixed for all SCC frames within the epoch. This claim is supported by the observation that the harvested power and workloads are identical for all SCC frames in one epoch. One parameter that may vary is the *initial energy level* for different frames in the epoch. At first, it looks like different starting energy levels may

result in different frame-level computation and communication speed assignments while trying to enforce the maximum energy level constraint $\gamma_{i,j} \leq \Gamma^{max}$. However, recall that the storage device automatically stops charging when the energy level approaches Γ^{max}, hence the maximum capacity does not need to appear as a constraint in the *frame level* energy management problem. The fixed speeds yield also a benefit on the implementation side: in general, a sensor node will need to convey every change in its modulation level b (communication speed d) to its receiving node. Therefore, fixing d within each epoch makes a practical implementation possible. Finally, note that the computation speed f could be different for the computation subtasks in SC and SCC frames, since voltage scaling is the only energy management tool for SC frames. The problem *SFEM* is specified as follows:

$$
\begin{aligned}
Max. \quad & \Delta\gamma_{i,j} \\
s.t. \quad & t_{i,j}^{exe} \leq \pi \\
& f_{min} \leq f_{i,j} \leq f_{max} \\
& d_{min} \leq d_{i,j} \leq d_{max}
\end{aligned}
$$

The objective is to maximize the energy level increment in a frame, while satisfying the application performance requirements.

4.2.1 Concurrent harvesting model

In this case, the stored energy e^h is constant (Eq. (4.6) and (4.7)). Thus, maximizing $\Delta\gamma = e^h - e^c$ is equivalent to minimizing e^c which is a function of f and d (Eq. (3.4)). In

this case, the problem becomes:

$$Min. \quad e^c = e^{sen} + e^{cp}(f) + e^{cm}(d) \tag{4.15}$$

$$s.t. \quad t^{sen} + C/f + M/d \leq \pi \tag{4.16}$$

$$f_{min} \leq f \leq f_{max} \tag{4.17}$$

$$d_{min} \leq d \leq d_{max} \tag{4.18}$$

Considering the nature of computation and communication energy consumption functions, the objective function can be seen to be convex. The problem has two unknowns (f and d), three inequality constraints. I denote this problem as *SFEM-C*.

4.2.2 Non-concurrent harvesting model

In this case, the harvesting time t^h is variable with the execution speeds f and d, i.e., $t^h = \pi - t^{sen} - (C/f) - (M/d)$ (Eq. (4.8)). Thus, saving energy by reducing speed will sacrifice harvesting opportunity, and may lead to even smaller $\Delta\gamma$. Hence, unlike the concurrent case, minimum energy consumption does not imply the maximum energy level increment. In this case, the problem becomes:

$$\begin{aligned} Max. \quad & e^h - e^c = t^h P^h - [e^{sen} + e^{cp}(f) + e^{cm}(d)] \\ s.t. \quad & t^{sen} + C/f + M/d + t^h = \pi \\ & 0 \leq t^h \leq \pi \\ & f_{min} \leq f \leq f_{max} \\ & d_{min} \leq d \leq d_{max} \end{aligned}$$

This problem has a concave objective function, three unknowns (f, d and t^h), one equality constraint, and three inequality constraints. Since maximizing a concave objective function $h()$ is equivalent to minimizing a convex function $-h()$, this problem leads to a convex

program as well. I denote the problem as *SFEM-N*.

4.3 Frame-Level Energy Management

In order to solve *SFEM-C*, I temporarily ignore the constraint (4.16) (timing requirement). By ignoring it, f and d can be scaled arbitrarily within their available ranges. Thus, the overall energy consumption e^c is minimized by minimizing the CPU energy e^{cp} and radio energy e^{cm} separately. The speeds f^*, d^* which yield the minimum e^{cp} and e^{cm} can be found by equalizing their first derivatives to zero:

Now, I take constraint (4.16) into consideration. If f^*, d^* satisfy constraint (4.16), I consider two special cases:

- if $f_{min} \leq f^* \leq f_{max}$, $d_{min} \leq d^* \leq d_{max}$, then the optimal solution is $f^{opt} = f^*$, $d^{opt} = d^*$.
- if $f^* < f_{min}$ and/or $d^* < d_{min}$, then $f^{opt} = f_{min}$ and/or $d^{opt} = d_{min}$. This is because e^{cp} and e^{cm} are monotonically-increasing in $f \in [f^*, +\infty]$ and $d \in [d^*, +\infty]$.

In [43], Aydin et al. derived f^* under an equivalent CPU energy model and referred it as the *energy-efficient frequency*. Also, d^* can be called the *energy-efficient communication speed*. If f^*, d^* violate constraint (4.16), the solution is more complicated. I note that problem *SFEM-C* can be rewritten as:

$$
\begin{aligned}
Min. \quad & e^{sen} + e^{cp}(t^{cp}) + e^{cm}(t^{cm}) \\
s.t. \quad & t^{sen} + t^{cp} + t^{cm} \leq \pi \\
& C/f_{max} \leq t^{cp} \leq C/f_{min} \\
& M/d_{max} \leq t^{cm} \leq M/d_{min}
\end{aligned}
$$

I use the computation and communication time (in a frame), $t^{cp} = C/f$ and $t^{cm} = M/d$ as the new variables. This makes problem *SFEM-C* a separable optimization problem with

the following structure:

$$
\begin{aligned}
Min. \quad & \sum_{k=1}^{p} F_k(x_k) \\
s.t. \quad & \sum_{k=1}^{p} x_k \leq \pi \\
& \forall k, x_{k,min} \leq x_k \leq x_{k,max}
\end{aligned}
$$

Above x^ks are equivalent to f and d. p is the number of variables which equals 2 for my optimization problem. In [43] and [63], it has been shown that any problem with the above structure can be solved in time $O(p^3)$ by manipulating Karush-Kuhn-Tucker optimality conditions ([64]). Since in problem *SFEM-C*, $p = 2$, it can be solved in constant time. The same method can be extended to solve problem *SFEM-N* which is also a separable problem. The detail of this method can be found in [43] and [63]. The derived t^{cp}, t^{cm} are then used to compute the optimal speeds f^{opt}, d^{opt}. Finally, f^{opt} and d^{opt} might not be available on the target hardware with discrete speed levels. However, I can use the lowest f and d which satisfy $f \geq f^{opt}$, $d \geq d^{opt}$, to guarantee the timely completion of the workloads.

4.4 Numerical Evaluation

Though I have demonstrated the optimality of algorithm *DVMS* for energy level maximization, I ran a set of experiments to determine the actual improvement in stored energy and γ_{min}, compared to the schemes that use either no or one of the voltage and modulation scaling techniques, under both concurrent and non-concurrent harvesting models. The simulations are conducted using TOSSIM, the standard WSN simulator [65]. In addition to the normal workload conditions where the worst-case application demand is constant, I also considered an emergency mode where there are sudden, unexpected peaks in the demand. The emergency mode is introduced to assess the scheme's capacity to cope with run-time

uncertainties and minimize service interruptions.

4.4.1 Node architecture and workload models

I consider two types of workloads, *normal* and *emergency*. The normal computation and communication workloads are generated randomly according to uniform distribution within the ranges C=[1, 2000000] CPU cycles, and M=[1, 128] bytes. The emergency mode is simulated by increasing the workload of frames by u times, where u ranges in $[1, 2]$. I assume that the sensor node encountered v emergencies in the horizon, each of which lasts w consecutive epochs. v and w are both random integers in the range of $[0, 10]$. My simulations used SCC frames.

Table 4.1: Specification of Intel Xscale Pxa27x

$Freq.(MHz)$	104	208	312	416	520	624
$Power(mW)$	116	279	390	570	747	925

The hardware basis for a DVS-DMS capable platform is the widely available iMote-2 sensor node [66]. The iMote-2 platform has a Intel Xscale PXA27x CPU [67] and a ChipCon CC2420 radio [68]. The frequency and power specification of PXA27x processor is given in Table 4.1. I derived the radio speed-independent power $P^{cm,ind} = 26.5$mW, radio symbol rate $R = 62.5k$ symbols/sec, and $\beta = 2.74 \times 10^{-8}$ based on CC2420 specification [68] and Eq. (3.3). I note that the CC2420 is not DMS-capable, so as in [15] I assume four modulation levels, $b = \{2, 4, 6, 8\}$ which give four communication speeds: $d = \{125, 250, 375, 500\}$ kbps (Eq. (3.1)). The radio energy is calculated using Eq. (3.3). I assume a light sensor TSL2561 [69] which takes 12ms to get one reading and consume 0.72mW. I assume the harvested energy is obtained from solar radiation, and use the solar power harvesting trace over one day provided in [16] as my harvesting profile. Solar energy can be harvested in either concurrent or non-current mode. I use the results in [16] to fix the harvesting cycle at

$H = 24hours$. This horizon is then divided into 96 epochs, each has a length $S = 15mins$. I simulated the execution of each invoked frame and set the frame period at $\pi = 30ms$. Each sensor node uses a rechargeable battery with capacity $\Gamma^{max} = 4000$ joules. All nodes start with the same initial energy level, $\Gamma^{init} = 2400$ joules.

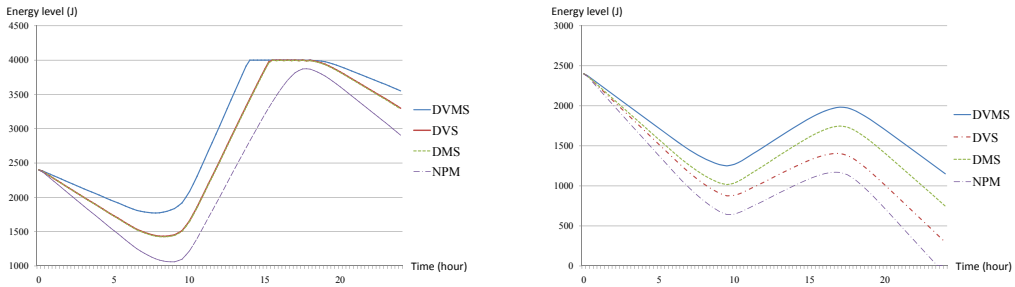
Although there are no existing schemes that are directly comparable to my approach, I have defined three new baseline schemes for comparison purposes. First, the *NPM* (No-Power-Management) scheme fixes both frequency and modulation level at the maximum across all epochs. Second, the *DVS* scheme scales only the frequency optimally, while fixing the modulation level at its maximum level. Third, the *DMS* scheme scales the modulation level optimally, while fixing the frequency at the maximum level. I use the metric *frame skip ratio* to measure the capacity of different schemes to cope with uncertainties, defined as the percentage of failed frames (missed deadline) due to empty energy storage in the horizon. Notice that this ratio also captures the scope of service interruption time: the higher the frame skip ratio, the longer the service interruption time.

For each of the experiments below, I ran 96 full epochs 1000 times. I then computed the average stored energy at the end of each epoch and plotted it as a data point.

4.4.2 Impact of joint voltage and modulation scaling

In Fig. 4.2a, I compare different schemes in stored energy of a sensor node executing normal workload, while harvesting concurrently. Among all the schemes, the energy level increases in the daytime as the sunlight intensity increases, and decreases in the evening due to the absence of sunlight. In all schemes, *DVMS* achieves the highest energy level. At the γ_{min} point (appeared around $8:00am$), *DVMS* stores 1800 joules (45.0% full) which is significantly higher than 1100 joules (27.5% full) for *NPM*, 1400 joules (35% full) for both *DVS* and *DMS*. As opposed to *DVS* or *DMS* schemes, the *DVMS* stores more energy since it has wider power scaling range, and always selects the most energy efficient speeds which yield the smallest energy consumption. All schemes have zero frame skip ratio which means no service interruption occurred.

In Fig. 4.2b, I ran the same experiment, but assumed non-concurrent harvesting. Again, *DVMS* stores more energy than all other schemes. The γ_{min} point (appearing at midnight) is about 1200 joules (30.0% full) for *DVMS* which is higher than 0 joule (empty) for *NPM*, 300 joules (7.5% full) for *DVS*, and 800 joules (20.0% full) for *DMS*. The *DVMS* stores the most energy as it uses the speeds which optimally balancing energy consumption and harvesting. Only *NPM* suffers 2.1% frame skip ratio.



a. concurrent harvesting

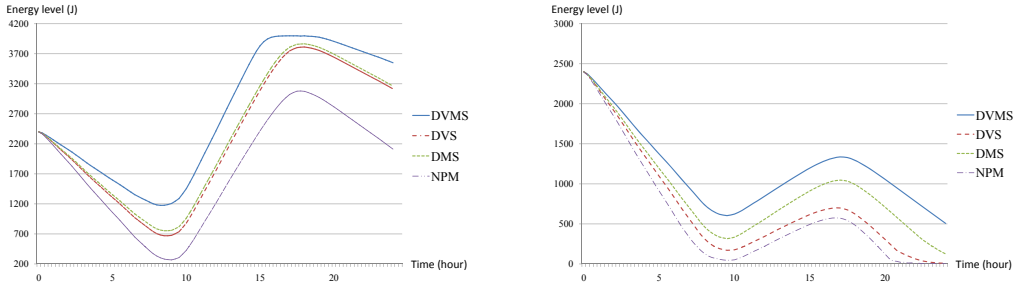
b. non-concurrent harvesting

Figure 4.2: Energy level - Normal mode

In Fig. 4.3a and 4.3b, I compare different schemes for a node executing emergency workloads, while harvesting concurrently and non-concurrently. The stored energy by all schemes decreases significantly compared to Fig. 4.2a and 4.2b due to the extra energy used by emergency workloads. The *DVMS* beats all other schemes again in stored energy and causes no service interruption. In Fig. 4.3a, no schemes suffers service interruption, while in Fig. 4.3b the *DVS* and *NPM* have skip ratio of 4.2% and 10.4%.

In all the above figures, the *DVMS* scheme stores significantly more energy than all other schemes, and never suffers service interruption. Under emergency mode, some schemes ran out of energy in the middle of operation for up to 10% time of service which potentially causes disasters to mission-critical applications. My experiments indicate the benefits of

my algorithm in term of both stored energy and resilience to system uncertainties.



a. concurrent harvesting

b. non-concurrent harvesting

Figure 4.3: Energy level - Emergency mode

Chapter 5: Joint DVS-DMS Energy Management for Tightly-Coupled Systems

I next turn attention to the energy management problem over an multi-hop network of sensor nodes. This network-level problem will share the same basic device model as the single-node problem. To precisely formulate the problem, I start by giving the definition of my network and application model. Note that some of the definitions were previously given in the discussion of the single-node problem, but are restated here in the context of a multi-hop system.

5.1 Network and Application Model

The system consists of N sensor nodes and the set of wireless communication links connecting them. A sensor node is denoted as V_i. Base stations or control points are denoted as BS. The N nodes are further divided into two different types. *Source* nodes perform sensing, processing and communication operations, while *relay* nodes only perform processing and communication. My data processing architecture is quite general, and supports systems that perform some levels of aggregation at each node, as well as systems that do not allow any aggregation. I represent a time-critical and performance sensitive WSN application by requiring that all source nodes report their readings, which may or may not be aggregated into other readings, every π time units. The time interval π is the length of a data collection *frame*. In other words, all sensed, processed or aggregated data must reach BS by the end of each frame. For example, at the start of the kth frame (i.e., at time $(k-1) \cdot \pi$), each source node senses the target environment and sends the sensed data to BS. The data must reach BS by the end of the kth frame, at time $k \cdot \pi$.

On a per-frame basis, energy consuming activities within each node are represented using a task-based model. In this way, frame-based energy consumption is determined by examining the power demands of individual tasks. There are a total of three task types: a *sensing* task, a *computation* task and a *communication* task. Without loss of generality and in order to simplify the modeling process, I assume the three tasks are executed in the order of *sense*→*computation*→*communication*. That is, in each frame, a node performs sensing first, then processes the sensor reading, then transmits the processed data. Note that the sense task is performed only by the source nodes. The workloads of the computation and communication tasks of any node V_i are fixed over any frame in a given epoch, and denoted as C_i and M_i, respectively.

I assume that each node uses standard WSN energy management techniques for transitioning to *sleep* states when there is no active task. In this chapter, I consider only concurrent energy harvesting model. I also assume that computation and communication speeds only change at the start of an epoch. This design decision reduces the required level of control and synchronization overhead. Using this analysis I can calculate the time required by each node V_i to carry out all activities during frame k, referred as the *per-node latency*, $l_{i,k}$. The per-node latency depends upon the computation speed $f_{i,k}$ and the communication speed $d_{i,k}$. Then $l_{i,k}$ is given by

$$l_{i,k} = t^{sen} + \frac{C_i}{f_{i,k}} + \frac{M_i}{d_{i,k}} \tag{5.1}$$

t^{sen} is the sensing time which is a constant. t^{sen} equals zero for relay nodes. I make a common assumption that the effective data transmission time dominates the overall communication time so I ignore the carrier sense time [14], [49], [15]. Thus, the communication time is inversely proportional to $d_{i,k}$.

The system is organized into a data collection and processing tree rooted at BS. In order to support time-critical operation I must define and calculate the *maximum data collection latency* and individual *path latency*. These two values are used in the optimization

formulation in 5.2 to ensure that all latency requirements are maintained. A node V_i receives data from a set of child nodes that are denoted as $Children(V_i)$. Node V_i forwards packets to its parent node, denoted as $Parent(V_i)$. Then the maximum data collection latency $L_{tot,k}$ of frame k is the time interval between the start of frame k, and when BS collects all sensed data, given by

$$L_{tot,k} = Max.\{L_{i,k} + l_{i,k} | V_i \in Children(BS)\} \tag{5.2}$$

Above, $L_{i,k}$ is the latency of the subtree rooted at node V_i, i.e. $L_{i,k} = Max.\{L_{j,k} + l_{j,k} | V_j \in Children(V_i)\}$. The subtree rooted at a leaf node contains only the leaf itself, and hence incurs zero latency.

Next, I define the *path* ρ_i from a node V_i to the root BS as the series of nodes and wireless links connecting V_i and BS. The notation $V_j \in \rho_i$ signifies that node V_j is an intermediate node on path ρ_i. The latency $H_{i,k}$ of ρ_i is given by

$$H_{i,k} = \sum_{j:V_j \in \rho_i} l_{j,k} \tag{5.3}$$

Note that by resolving the recursion in Eq. (5.2), $L_{tot,k}$ actually equals to the latency of the longest path in the tree, i.e., $Max.\{H_{i,k} | \forall \rho_i\}$.

5.2 Harvesting Aware Speed Selection

Based on the network model presented in 5.1, I now formally define the *Harvesting Aware Speed Selection (HASS)* problem. My goal is to maintain end-to-end performance while maximizing system's resilience to abnormal or emergency situations. This is accomplished by maximizing the minimum energy level of any node.

The computation and communication speeds at individual nodes are adjusted at the start of each epoch, and remain fixed throughout that epoch. As defined in 3.2, an epoch

is a time interval over which an energy harvesting prediction can be reasonably made. For an arbitrary epoch, the energy consumption $e^c_{i,k}$ (Eq. 3.4) and performance latencies $L_{i,k}$, $H_{i,k}$ of node V_i are fixed over any frame k. For simplicity I therefore rewrite them as e^c_i, L_i and H_i. Then the energy level Γ_i of a node V_i at the end of a given epoch is given as:

$$\Gamma_i = \Gamma_i^{init} + P_i^h \cdot S - \lfloor S/\pi \rfloor \cdot e_i^c \tag{5.4}$$

Γ_i^{init} is the starting energy level of V_i in the epoch. Recall that S is the epoch length. $\lfloor S/\pi \rfloor$ gives the number of frames in an epoch. Using this notation, I define Γ_{min} as

$$\Gamma_{min} = Min.\{\Gamma_i | \forall V_i\} \tag{5.5}$$

Then the goal of my approach is to maximize Γ_{min}. The variables of the problem are the computation and communication speeds f_i, d_i used by any node V_i in an epoch. Given N nodes in the tree, there are $2N$ unknowns in the problem. The optimal solution to this problem consists of N *speed configurations* (f_i, d_i), one for each node which maximize Γ_{min}. The problem *HASS* is given as:

$$Max. \quad \Gamma_{min} \tag{5.6}$$

$$s.t. \quad \forall \rho_i, H_i \leq \pi \tag{5.7}$$

$$\forall V_i, f_i \in [f_{min}, f_{max}], d_i \in [d_{min}, d_{max}] \tag{5.8}$$

$$\forall V_i, 0 < \Gamma_i \leq \Gamma^{max} \tag{5.9}$$

The constraint (5.7) ensures that the latency of any path ρ_i in the tree is smaller than the frame period π. As mentioned in 5.1, this is equivalent to ensuring that the latency of the entire tree is smaller than π. The constraint (5.8) gives the available ranges of f and d. The constraint (5.9) requires that the energy level of any node V_i must be confined to the range $(0, \Gamma^{max}]$. The left hand side of constraint (5.9) (called the *positivity constraint*) must hold

in order to ensure energy neutrality, while the right hand side (called the *capacity constraint*) is used to model energy storage capacity. Given known and constant harvested power, and fixed speeds and power consumption, the variation of energy level also fix throughout an epoch, i.e. either monotonically increase or decrease at a fixed rate. Therefore, ensuring a positive energy level at the end of an epoch also ensures positive energy level at the end of any frame in that epoch.

5.3 Centralized and Distributed Solutions

Now I present both centralized and distributed solutions to the *HASS* problem. The centralized solution solves the problem optimally, while the distributed version is appropriate for systems that need to avoid single control point.

I first give Lemma 1 which states that solving problem *HASS* with full constraint set is equivalent to solving the same problem but without constraint (5.9). This enables me to remove constraint (5.9) and focus on a new problem obtained in this manner, denoted as *HASS-N*. Note that the objective function and all other constraints are retained in *HASS-N*.

Lemma 1. If in the optimal solution to HASS-N, Γ_{min} is strictly positive, then the solution to HASS is identical to that of HASS-N. Otherwise, HASS does not have a feasible solution.

The proof of Lemma 1 is given in the Appendix. Now I can focus on solving problem *HASS-N*. Solving HASS-N requires non-linear optimization techniques, since it has a non-linear objective function (Eq. (5.6)). Such costly methods are difficult to implement on resource-constrained sensor nodes. I will show how to obtain an optimal solution efficiently.

A naive approach to solve *HASS-N* is to exhaustively search over all possible solutions. For a system with N nodes where each node has m computation speeds and n communication speeds, there are $(mn)^N$ possible solutions, making brute force search impractical. However, I notice that many different solutions yield identical Γ_{min} which is justified later in the next paragraph. Using this observation I can simply enumerate each possible Γ_{min}, check if there exists a feasible solution that yields a minimum energy level (among any node)

equaling the enumerated Γ_{min}, while satisfying constraints (5.7) and (5.8). The highest Γ_{min} that passes this check is by definition the maximum Γ_{min} that I am looking for.

For each node, mn speed configurations correspond to mn different power consumption levels. Since each node's power consumption is fixed throughout an epoch, a node has exactly mn energy consumption levels over an epoch. Thus, given a known starting energy level and a fixed prediction for how much energy can be harvested, a sensor node could end with at most mn different energy levels in an epoch. Given N nodes, at the end of an epoch, there could be at most mnN different energy levels in the network, and Γ_{min} can be only of these possible values. The set of possible Γ_{min}s is referred as EL (Energy Level), and has a size of mnN.

5.3.1 Centralized solution

I call the centralized *HASS* algorithm *CHASS*, given in Algorithm 1. It runs on the base station, and assumes that BS must collect Γ^{init} from each node in the system, and is aware of the available speed configurations of sensor nodes. *CHASS* first computes the possible energy levels of all the nodes using Eq. (5.4) to build the set EL, then sorts EL in non-increasing order (line 1). *CHASS* proceeds iteratively over the sorted EL starting from the first element (i.e. the highest energy level in EL) (line 2). In each iteration p, it solves a decision problem, called *Feasible Solution* denoted by FS_p, by calling algorithm *Is-Feasible* (line 3). The pth element in EL, $EL[p]$ is input to *Is-Feasible*. The problem FS_p is specified as "Is there a solution which yields Γ_{min}=$EL[p]$, while satisfying constraints (5.7-5.8)?"

The loop in line 2-9 iterates through all the elements in EL. It continues if the answer to problem FS_p, ans_p is negative, and terminates once it met a FS_p with positive answer, i.e. in iteration z where FS_z is the first problem encountered with positive answer, $z = Min.\{p \in [1, |EL|] | ans_p = TRUE\}$ (line 4-8). By definition of problem *HASS-N* and *FS*, and the ordering of EL, $EL[z]$ is the maximum Γ_{min} that can be achieved (line 5), while satisfying all the constraints. If *CHASS* proceeds to the end of EL and never received a positive answer to any of the FS_p, this implies problem *HASS-N* has no feasible solution.

The algorithm *Is-Feasible* for solving problem FS_p is given in Algorithm 2. The algorithm has one input, the energy level enumerated in iteration p of *CHASS*, $EL[p]$. It has three returned values, the answer to problem FS_p, ans_p, and two speed sets of length N, F^*, D^* which contain f and d derived for all the nodes in the current iteration. F^*, D^* are returned only if ans is positive, otherwise they are empty.

Algorithm 1 CHASS

```
1: Compute and sort EL (in non-increasing order)
2: for p = 1 to |EL| do
3:    [ans_p, F*_p, D*_p] = call Is-Feasible(EL[p])
4:    if ans_p == TRUE then
5:       Max_Γ_min = EL[p]
6:       [F^opt, D^opt] = [F*_p, D*_p]
7:       Break from for-loop
8:    end if
9: end for
```

First, by making $\Gamma_{min} = EL[p]$ (line 1), $\Gamma_i \geq \Gamma_{min} = EL[p]$ must hold for any node V_i. Then the algorithm calls function *find_fastest* for each node (line 2-4) to search over all its mn speed configurations for the fastest one, while yielding $\Gamma_i \geq EL[p]$. Let $\Gamma_i(f, d)$ and $l_i(f, d)$ represent the energy level and per-node latency achieved using speed configuration (f, d). Then, *find_fastest* returns a speed configuration for V_i, $(F^*[i], D^*[i])$ which satisfies:

$$F^*[i] \in \{f_1, f_m\}, D^*[i] \in \{d_1, d_n\} \tag{5.10}$$

$$\Gamma_i(F^*[i], D^*[i]) \geq EL[p] \tag{5.11}$$

$$\forall (f^{'}, d^{'}) \in \{(f, d) | \Gamma_i(f, d) \geq EL[p])\}, \tag{5.12}$$

$$l_i(F^*[i], D^*[i]) \leq l_i(f^{'}, d^{'})$$

Since there are only mn combinations of f and d, *find_fastest* can search for $(F^*[i], D^*[i])$ using brute force. *find_fastest* also returns the per-node latency l_i^{min} at V_i achieved by using the derived $(F^*[i], D^*[i])$. Note that l_i^{min} is the least achievable latency according to

Eq. (5.12). Next, for each path ρ_i, I compute its latency H_i by summing up any $l_j^{min}, V_j \in \rho_i$ (line 5). Since (F^*, D^*) minimizes the per-node latency at any node, it also minimizes the latency of any path H_i. Therefore, if $H_i \leq \pi, \forall \rho_i$, the constraint (5.7) is met, then the answer to problem FS_p is positive (line 6-7). Otherwise, constraint (5.7) can never be met, hence the answer is negative (line 8-9). Note that it is possible that function $find\_fastest$ does not return an answer, as there may exist some nodes having no possible energy level larger than the input $EL[p]$. In this case, the algorithm immediately rejects $EL[p]$. The speed sets F^*, D^* found in iteration z is set to be the optimal solution to problem *HASS-N* and also *HASS* (line 6 in Algorithm 1). $EL[z]$ is set to be the maximum achievable Γ_{min} (line 5 in Algorithm 1).

Algorithm 2 Is-Feasible - Input: $EL[p]$

1: $\Gamma_{min} = EL[p]$
2: **for** $i = 1$ to N **do**
3: $\quad (F^*[i], D^*[i], l_i^{min}) =$ call $find\_fastest(\Gamma_{min})$ on V_i
4: **end for**
5: Compute $H_i = \sum_{j:V_j \in \rho_i} l_j^{min}$ for any path ρ_i
6: **if** $\forall \rho_i, H_i \leq \pi$ **then**
7: $\quad ans = TRUE$
8: **else**
9: $\quad ans = FALSE,\ F^*, D^* = \emptyset$
10: **end if**
11: **return** $[ans, F^*, D^*]$

It is possible to reduce the runtime of the above algorithm. In order to do so I present Lemma 2 and Corollary 3, which is used as the basis for $CHASS^*$, the faster algorithm. The key idea of $CHASS^*$ is to implement a binary search for FS_z. This will reduce the number of iterations in *CHASS* from $O(|EL|)$ to $O(\log(|EL|))$.

Lemma 2. *For any node V_i, the least per-node latency found by invoking algorithm* Is-Feasible *with Γ_1 as input is no smaller than the one found with Γ_2 as input, where $\Gamma_1 \geq \Gamma_2$.*

The proof of Lemma 2 is given in the Appendix. Given Lemma 2, the following corollary stands:

Corollary 3. *For any node V_i, the least per-node latency found in iteration p is no smaller than the one found in iteration q, $\forall q \geq p$.*

Corollary 3 holds because $EL[p] \geq EL[q]$, given that EL was sorted in non-increasing order. Corollary 3 implies that the latency of any path found in iteration p is also no smaller than the one found in iteration q, $\forall q \geq p$. Therefore, I can implement the search for FS_z using binary search. The search starts from the pth element of EL, $p = \frac{|EL|}{2}$, and

- continues on the left half ($[1, p-1]$) if FS_p has positive answer. Due to smaller path latency found in iteration q, $q > p$, any FS problem on the right half must also have positive answer, hence it is unnecessary to search that half. Rather, on the left half, I may find a problem FS with larger achievable Γ_{min}.
- continue on the right half (i.e. $[p+1, |EL|]$) if FS_p has negative answer. Due to even larger path latency found in iteration q where $q < p$, any FS on the left half must violate constraint (5.7), thus has negative answer.

The binary search continues on either half depending on the answer to FS_p, until FS_z is found. The binary search based implementation reduces the number of iterations in *CHASS* from $O(|EL|)$ to $O(\log(|EL|))$.

Complexity analysis: Given mn speed configurations, the runtime of *find_fastest* is $O(mn)$. Given N nodes, the loop in line 2-4 of Algorithm 2 has runtime $O(mnN)$. Also, computing the latency for all paths (line 5) takes $O(N)$ time since there are N nodes. Therefore, the runtime of *Is-Feasible* is $O(mnN)$. Since $CHASS^*$ iterates for $O(\log(|EL|))$=$O(\log(mnN))$ rounds, its total complexity is $O(mnN \log(mnN))$. Since m and n are typically much smaller than N, the algorithm can be seen as an efficient one in practice. In terms of the communication overhead, the gathering of initial energy levels from all the nodes at BS requires one round of data collection.

5.3.2 Speed reduction for nodes on non-critical paths

I note that the function $find\_fastest$ in Algorithm 2 produces node speed assignments that run at the highest possible speed configuration that satisfies $\Gamma_i \geq \Gamma_{min}$. This speed assignment is not always desirable or necessary, and I now present a scheme to reduce speed for the unnecessarily fast nodes, given that Γ_{min} has been maximized by *CHASS*. First I define the *critical path*. Given the speed assignment derived by *CHASS*, a critical path in the tree is any path on which no nodes can further reduce their computation or communication speeds without violating constraint (5.7). Note that such critical paths must exist. This is because if the tree does not contain a critical path, then the speed of nodes can be further reduced without violating the constraint (5.7). This will increase the energy level of these nodes, hence *CHASS* will proceed to a new iteration and find a higher Γ_{min}. This contradicts the optimality of *CHASS* that Γ_{min} is already maximized.

Recall that H_i is the latency for packets from V_i to reach BS, and L_i is the time for V_i to collect packets from all its descendent source nodes. For any node V_i on a critical path, H_i is fixed since the speed of any node on ρ_i cannot be further reduced. Since the latency from V_i to BS is fixed at H_i, in order for V_i to collect and forward packets sensed by its descendent nodes to BS within the latency constraint π, any of those packets must reach V_i in no more than time $\pi - H_i$. In other words, the collection latency of V_i, L_i must be smaller than $\pi - H_i$. Therefore $\pi - H_i$ can be interpreted as the latency constraint at V_i, denoted as LC_i.

I refer to the speed reduction scheme as *SpeedReduction*, and give it in Algorithm 3. *SpeedReduction* reduces speeds of nodes based on the speed assignment derived by *CHASS*. Line 1 identifies the path with the largest latency in the tree, denoted as ρ^{max}. Notice that ρ^{max} must be a critical path, and the base station is also on ρ^{max}.

Lines (2-7) iterate over every node V_i on ρ^{max} (from BS down to the leaf node on ρ^{max}) to reduce speeds of their descendent nodes, while ensuring that the resulted collection latency L_i is smaller than the latency constraint at V_i, LC_i. Specifically, in each iteration of the for-loop, procedure *DoSpeedReduction* is called over any immediate children of V_i, except

the one resided on ρ^{max}. The procedure *DoSpeedReduction* given in Algorithm 4 executes in recursive fashion. In this way, lines (2-7) will ultimately visit every node in the tree exactly once, in a top-down fashion.

Algorithm 3 SpeedReduction

1: Identify ρ^{max} in the tree.
2: **for** any node V_i on ρ^{max}, from BS down to the leaf node on ρ^{max} **do**
3: $LC_i = \pi - H_i$
4: **for** any child V_j of V_i **do**
5: **If** $V_j \notin \rho^{max}$ **then** $DoSpeedReduction(LC_i, V_j)$;
6: **end for**
7: **end for**

Algorithm 4 DoSpeedReduction

1: Input: LC_i, V_j
2: $(\tilde{f}, \tilde{d}) = arg\,min\{E^c(f,d) | L_j + l_j(f,d) \leq LC_i\}$
3: Assign $(\tilde{f}, \tilde{d})$ to V_j
4: Set $LC_j = LC_i - l_j(\tilde{f}, \tilde{d}) = \pi - H_i - l_j(\tilde{f}, \tilde{d})$
5: **for** any child V_k of V_j **do**
6: $DoSpeedReduction(LC_j, V_k)$;
7: **end for**

Now I specify Algorithm 4, *DoSpeedReduction*. *DoSpeedReduction* has two inputs, the node V_j whose speed is to be reduced, and the collection latency constraint of its parent, LC_i. *DoSpeedReduction* reduces speed for V_j, while fixing the speeds of V_j's children. That is, $l_j(f,d)$ is to be increased, while L_j remains fixed. Specifically lines (2-3) reduce the speed configuration of any node V_j to a level $(\tilde{f}, \tilde{d})$ that yields the minimum energy consumption $E^c(\tilde{f}, \tilde{d})$ (Eq. 3.4), among any (f,d)s that give $L_j + l_j(f,d) \leq LC_i$. This will yield the maximum energy level increment at these nodes. The reduction of speed to $(\tilde{f}, \tilde{d})$ will yield a new latency at V_j, $l_j(\tilde{f}, \tilde{d})$.

After the speed reduction at V_j, lines (5-7) continue reducing the speeds of V_j's children V_ks, where the latency constraint LC_j at V_j is set to $\pi - H_i - l_j(\tilde{f}, \tilde{d})$ in line 4. As procedure *DoSpeedReduction* executes recursively until the leaf nodes are reached, it visits every descendent node of V_i, and reduces their speeds.

DoSpeedReduction has the desirable property of reducing the speed of nodes it visits

earlier in the procedure. This is desirable because in any tree-structured network, high level nodes, nodes which will be visited earlier by the procedure, have more descendant nodes than the low level nodes, and hence are more critical to the system in terms of network connectivity. Therefore, one need to obtain more energy for these nodes in order to prevent them from energy depletion and consequently network partition and service interruption.

Complexity analysis: As mentioned earlier in this subsection, *SpeedReduction* and the recursive procedure *DoSpeedReduction* visit every node in the network exactly once. For each node visited, *DoSpeedReduction* finds the speed configuration $(\tilde{f}, \tilde{d})$ according to line 2 of *DoSpeedReduction* which has time cost of $O(mn)$ given mn speed configurations. Therefore, given N nodes in the network, the total complexity of *SpeedReduction* is $O(mnN)$.

5.3.3 Distributed Version

Next I describe the distributed *HASS* solution called *DHASS*. The purposes of the distributed version is to enable any node in the network to act as the base station, thereby enabling that node to make command and control decisions.

The algorithm *DHASS* also proceeds in binary-search fashion. It requires one initialization round during which each sensor node sends an *initialization* message containing two pieces of information, its estimated lowest and highest energy levels at the end of the epoch, denoted as Γ^{low} and Γ^{high}. After the initialization round, all the nodes agree on the global lowest and highest achievable energy levels (within the entire tree). The continuous range between the two energy levels is the starting binary search space. It then runs for Y computation rounds, each of which corresponds to one iteration of binary search, and solves one instance of problem FS using the distributed *Is-Feasible*. In each computation round, the midpoint of the search space is used as the input energy level to *Is-Feasible*. Given that input, each node calls function *find_fastest* individually to derive its fastest speed configuration and associated per-node latency. It then computes the accumulative latency at it, i.e. $L_i + l_i$ and sends to its parent as a *latency* message. The parent computes its accumulative latency as well based on the received latency messages from its children. By making all

the nodes compute and report their latencies accumulatively, the latency of the entire tree L_{tot} will be ultimately computed at the root. The root then compares L_{tot} to π in order to determine the answer to problem FS, and disseminates it to all the nodes as a *decision* message. Note that any node in the network can be the root.

In the initialization round, each node estimates its local Γ^{low}, Γ^{high}, then forwards to its parent as an initialization message. Γ_{low} is calculated using Eq. 3.4 and 5.4 while assuming use of the highest speed configuration. Γ_{high} is also calculated using these two equations but assuming the lowest speed configuration. A node receives initialization messages from its children, and compares Γ^{low}, Γ^{high} received to its own values, in order to derive Γ^{low} and Γ^{high} among its children and itself. The derived Γ^{low}, Γ^{high} are sent to its parent as well. When the root receives initialization messages from all its children, it derives the global Γ^{low} and Γ^{high} which actually equals to the minimum and maximum elements in EL, $Min(EL)$, $Max(EL)$. Then the root disseminates the global Γ^{low} and Γ^{high} to all the nodes for the use in the first computation round. Their values will be updated in each computation round according to a rule given in the following paragraphs. The initialization round ends when all the nodes have received the global Γ^{low} and Γ^{high} values.

The procedure of the distributed *Is-Feasible* is different for non-root and root nodes. Algorithm 5 presents the non-root node case, while Algorithm 6 presents the root node case. Both proceed iteratively for Y computation rounds. Y is tunable parameter defined by system designers. Each computation round starts by requiring the leaf nodes to send latency messages upwards over the tree. These latency messages will then trigger the computation procedure on the non-leaf nodes.

Lines 3-26 in Algorithm 5 specify the procedure of the distributed *Is-Feasible* in one computation round at the non-root nodes. Line 26 starts a new computation round by directing the execution to line 3. Algorithm 5 has two inputs Γ^{low} and Γ^{high} which have been derived in the initialization round, and outputs the optimal speed configuration and the maximum Γ_{min} found, i.e. MAX_Γ_{min}. It also uses a variable *round* to keep track of the current computation round. *round* is initialized to 0.

Algorithm 5 Procedure of Distributed Is-Feasible at a non-root node V_i

1: Input: Γ^{low}, Γ^{high}
Output: $(F^{opt}[i], D^{opt}[i])$, MAX_Γ_{min}
2: Initialization: $round = 0$
3: $round = round + 1$
4: **if** V_i is a non-leaf node **then**
5: Wait to receive $L_j + l_j^{min}$ from every child V_j
6: **end if**
7: Compute $L_i = Max.\{L_j + l_j^{min} | V_j \in Children(V_i)\}$
8: Compute $X = \frac{\Gamma^{low}+\Gamma^{high}}{2}$
9: $(*, *, l_i^{min}) = find\_fastest(X, V_i)$
10: Send $L_i + l_i^{min}$ to $Parent(V_i)$
11: Wait to receive decision message (containing ans) from the root
12: **if** $ans == TRUE$ **then**
13: $\Gamma^{low} = X$
14: **else**
15: $\Gamma^{high} = X$
16: **end if**
17: **if** $round == Y$ **then**
18: **if** $ans == TRUE$ **then**
19: $MAX\_\Gamma_{min} = X$
20: **else**
21: $MAX\_\Gamma_{min} = \Gamma^{low}$
22: **end if**
23: $(F^{opt}, D^{opt}, *) = find\_fastest(MAX\_\Gamma_{min}, V_i)$
24: Return
25: **end if**
26: Goto line 3

Algorithm 6 Procedure of Distributed Is-Feasible at the root

1: Wait until $L_j + l_j^{min}$ has been received from all children
2: $L_{tot} = Max.\{L_j + l_j^{min} | V_j \in Children(Root)\}$
3: **if** $L^{tot} \leq \pi$ **then**
4: $ans = TRUE$
5: **else**
6: $ans = FALSE$
7: **end if**
8: Disseminate ans across the tree

Algorithm 5 can be explained as follows: Line 3 sets the current computation round. If node V_i is a non-leaf node, then before it starts computation, it must wait to receive latency messages from each of its children V_j (line 4-6). A latency message contains $L_j + l_j^{min}$ computed by its child V_j. Using the received $L_j + l_j^{min}$, V_i computes its collection latency L_i using Eq. 5.1 in line 7. If V_i is a leaf node which has no child, then it has $L_i = 0$ and does not wait for latency messages. Next in line 9, V_i derives the smallest per-node latency l_i^{min} while satisfying $\Gamma_i \geq X$, by calling function $find\_fastest$ specified in 5.10-5.13. $X = \frac{\Gamma^{low}+\Gamma^{high}}{2}$ is the input energy level in the current computation round (line 8). Then V_i adds up l_i^{min} to L_i, sends to its parent (line 10), and waits to receive decision message to be disseminated from the root (line 11).

In Algorithm 6, the specification of the root-side procedure of distributed *Is-Feasible*. As each node V_i receives $L_j + l_j^{min}$ from its children, it computes and sends its own $L_i + l_i^{min}$, the root will compute the total collection latency L_{tot} after receiving $L_j + l_j^{min}$ from all its children (line 1-2). In line 3-7, the root compares L_{tot} to the latency constraint π. If $L_{tot} \leq \pi$, the root sets the answer to the associated problem FS in the current round, i.e. ans as TRUE; otherwise it sets ans to FALSE. Finally, the root disseminates ans across the tree (line 8).

Returning to Algorithm 5, when V_i receives the decision message from the root (line 11), it updates the binary search range (represented by Γ^{low} and Γ^{high}) for the next computation round. This is based on the answer ans enclosed in the decision message. If $ans == TRUE$, V_i sets Γ^{low} to X which directs the binary search to the higher half of the current search range, otherwise it sets $\Gamma^{high} = X$ which directs the search to the lower half (line 12-16). Then V_i starts a new computation round by directing the execution to line 3. Finally, if the current computation round is the Yth round (line 17), then V_i determines MAX_Γ_{min} as follow:

- If $ans == TRUE$, MAX_Γ_{min} is set to be the X in the Yth round (line (18-19)).

- Otherwise, MAX_Γ_{min} is set to be the Γ^{low} in the Yth round (line (20-22)).

Using the derived MAX_Γ_{min}, V_i calls function *find_fastest* to determine the optimal speed configuration $(F^{opt}[i], D^{opt}[i])$ (line 23), and terminates (line 24).

I now present Theorem 2, showing that the performance of *DHASS* is very close to optimal.

Theorem 2. *The maximum Γ_{min} found by algorithm DHASS is smaller than the optimal value, by at most $(Max(EL) - Min(EL))/2^{Y-1}$.*

The proof of Theorem 2 can be found in the Appendix. As seen from Theorem 2, using a larger Y for *DHASS* achieves closer performance to the optimal, however this comes at cost of higher complexity shown as follow.

Complexity analysis: In a computation round, the major time cost of a node comes from function *find_fastest* which equals $O(mn)$. Given Y computation rounds, the total cost is $O(mnY)$. The root compares L_{tot} to π in each computation rounds, this causes a time cost of $O(Y)$. In the initialization round, each node sends exactly one initialization message. In any computation round, each node sends exactly one latency message. The optimal speed configurations are computed on each node individually, hence there is no dissemination cost.

5.4 Performance Evaluation

I performed a series of simulations to evaluate the effectiveness of my *HASS* approaches. Specifically, the goal of the evaluation is to determine how well both the *CHASS* and *DHASS* algorithm maximize the minimum energy level across the system. I considered two WSN application types: aggregating and non-aggregating application. The evaluation examined a number of workload scenarios, including several emergency scenarios where there are sudden, unexpected peaks in the demand.

5.4.1 Experimental methodology

Without loss of generality I evaluated my approaches within a WSN system designed for residential monitoring of water usage and quality. My approach can also be applied to WSN systems that utilizing different energy sources, such as solar or wind energy. Each customer (residence) is coupled to a supply pipe through a water meter. The purpose of the water meter may be to measure the amount of water used by each customer, monitor the water quality, and alike. Future water meters will be used to provide customers and companies instantaneous pricing information, measurements of water quality, detecting the presence of containments, and alike. My simulation environment assumes that each water meter is coupled with a DVS-DMS enabled node. Energy is harvested from the flow of water. The amount of harvested energy is therefore dependent upon the rate at which the customer uses water. To my best knowledge, my work is the first to simulate energy harvesting WSN systems utilizing water energy source.

I have developed simulation software upon TOSSIM: the standard high-fidelity WSN simulator, combined with EPANET [70], a public domain, water distribution system modeling program developed by the US Environmental Protection Agency's Water Supply and Water Resources Division. My simulator can take as input a variety of WSN topologies, water distribution system configurations and customer usage patterns. Based on water utilization and water quality patterns, the software simulates energy harvesting, and various WSN processing and communication activities. The presented results are based upon a 100 node residential water distribution topology. The topology is derived from an existing suburban area of 100 houses with nodes spaced approximately 50 to 75 ft. apart. The 100 nodes installed in the houses then form a WSN system. I use the *Collection Tree Protocol* [71] to organize the nodes into a data collection tree.

Due to the repetitive water usage pattern with a cycle of 6 hours as supported by EPANET, I fix the harvesting horizon at $H = 6$ hours. A horizon is then divided into 24 epochs with equal length $S = 15$ minutes. I run EPANET for 48 hours, containing 8 horizons or equivalently 192 epochs, and obtained hydraulic simulation reports. Using

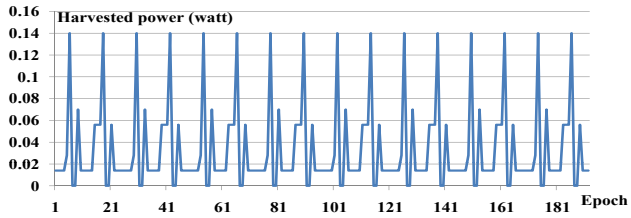


Figure 5.1: Water energy harvesting profile

these reports I generate water energy harvesting profile for each node based on the observed water usage at the customer. Note that the difference in the amount of water used across residences will lead to quite different power harvesting profiles across nodes. Fig. 5.1 shows the harvesting profile of one selected node. I assume concurrent energy harvesting model for all the experiments below. The frame period is set to $\pi = 240ms$, and therefore each epoch contains 3750 frames.

Both algorithm *CHASS* and *DHASS* were implemented in my simulation environment. Although there are no schemes that are directly comparable to my algorithms, I implemented a baseline scheme called No-Power-Management (*NPM*). Unlike the *HASS* approaches, *NPM* scheme is harvesting-unaware in the sense that it uses the highest frequency and modulation level on all the nodes in order to guarantee data collection timing constraint. My experiments considered two basic application types: applications that support *complete-aggregation* and applications that do not require any aggregation (*non-aggregation*). By complete-aggregation, I mean that each node aggregates multiple packets received into one single packet, while in the non-aggregation case a sensor node forwards all packets to its parent without aggregation. The packet size is randomly selected between M=[64, 128] bytes following uniform distribution, and the computational workload is randomly selected between C=[0, 3000000] cycles. These two scenarios produce highly different levels of workload and network traffic.

I assume almost the same hardware platform as specified in 4.4, including a DVS-capable

CPU, a DMS-capable radio, a light sensor, except that I model a water energy harvester instead of solar panel. Also, due to relatively narrower range of energy level variation in water-powered WSN systems, I reduce the battery capacity from 4000 joules to 1000 joules, and each node has a starting energy level of 600 joules. Nodes operate in either *normal* or *emergency* mode. I represent the emergency mode by increasing the frame-based workload by w times upon the normal mode, where w is a tunable parameter. This reflects the fact that nodes will need to perform additional duties during those times. I simulate emergency scenarios by introducing contaminants into the system at random time. This can be done by deteriorating the water quality at the water reservoir or certain residences in EPANET (such as a biological attack on a water supply). As the contaminant spreads out, the water quality in the residences will decrease and finally been detected by sensor nodes. A sensor node then switches to emergency mode and perform additional workloads over a series of epochs, until the water quality returns to normal.

I consider three different types of emergency scenarios which affecting the system in different patterns. The first type is *random* (RAND) attack. In this case, nodes fail according to a negative exponential distribution, and are picked according to a random uniform distribution from among all the nodes still operating in *normal* mode. This represents an event that suddenly impacts the system, such as a power blackout, industrial catastrophe, etc. The second mode is a *spreading attack* (SPRD). This represents an emergency that increases its area of impact over time. I introduce contaminant into the system from one randomly selected contamination source node. The contaminant spreads out of the system with the flow of water, and lasts a few epochs until water valves are shut off to stop further spreading. One example is a terrorist attack on a water supply, where a contaminate is introduced via a reservoir, and is spread throughout the system. The third mode is *area instant* (INST) attack. Under this scenario a large contiguous area of the network is affected. I prevent spreading of contaminant by shutting off the water valves. I simulated one emergency in each horizon, while an emergency started in one horizon may continue to affect multiple successive horizons.

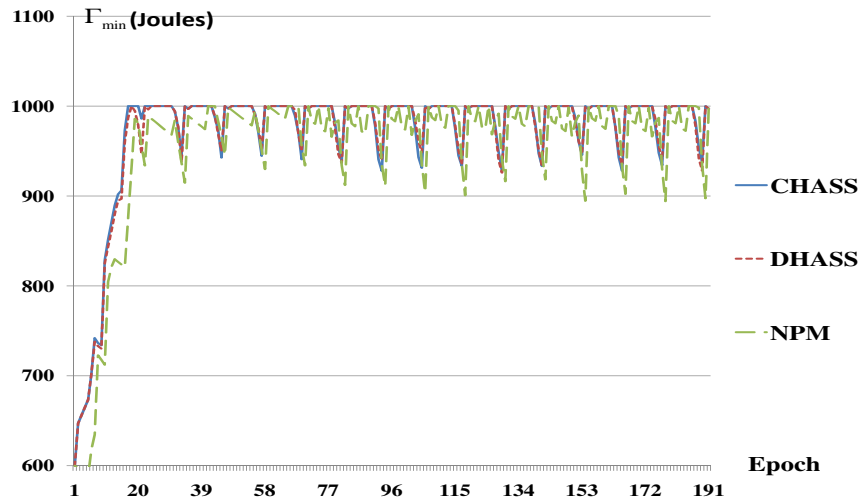


Figure 5.2: Min. energy level, non-aggregation: Normal mode

5.4.2 Results

In *CHASS* scheme, the set EL contains 2400 elements, given 6 CPU frequencies, 4 modulation levels, and 100 nodes. In *DHASS* scheme, we set the number of search iterations to be $\log(|EL|) \approx 11$. In the *NPM* scheme, I fix both the frequency and modulation level at the maximum across all epochs. I evaluated the performance of my algorithms under normal and all three emergency modes. Each scenario was tested using complete aggregation and no aggregation. I assume the emergency workloads of computation and communication are w times the normal workloads. I first fix w at 2.0, and compare different schemes in term of the achieved Γ_{min}. Then, I gradually increase w from 1.5 to 3.0 (with step size of 0.5) to evaluate the resilience of our schemes against various intensity levels of emergency.

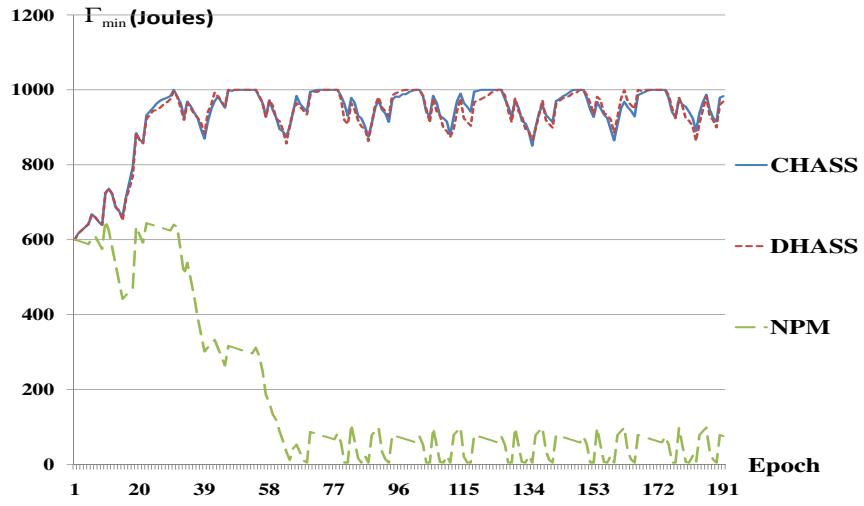


Figure 5.3: Min. energy level, non-aggregation: RAND mode

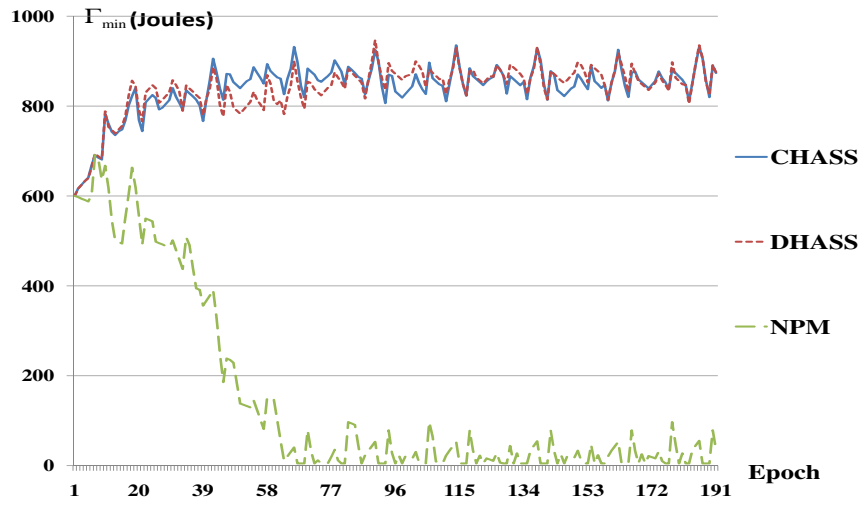


Figure 5.4: Min. energy level, non-aggregation: SPRD mode

Non-aggregating applications

In Fig. 5.2-5.4, I compare different schemes in terms of the achieved Γ_{min}, while assuming non-aggregating applications. I fix the emergency level w at 3.0, meaning that the emergency workload is three times the normal workload. In normal mode (Fig. 5.2), the Γ_{min} value can be seen to vary semi-repetitively. It stays close to full capacity at most time, however drops down twice in every horizon (24 epochs). This is because the workload and energy demand in normal mode is relatively low, such that the energy level is dominantly affected by the amount of harvested water energy which varies in repetitive pattern. However in Fig. 5.3-5.4, the significantly increased workload demand (in RAND and SPRD emergency modes) turns to have a dominant effect on energy level, therefore one emergency in each horizon leads to one drop of Γ_{min} in each horizon. This observation demonstrates the effects of harvested energy and workload demand over energy level when operating in different work modes.

As seen from all above figures, in the normal and emergency modes, the *CHASS* scheme achieves the highest Γ_{min}, followed by *DHASS* with slightly lower Γ_{min}, and then *NPM*. In normal mode (Fig. 5.2), *NPM* scheme also achieves a high Γ_{min}. This is because the harvested energy is much larger than energy demand in normal mode. As workload demand increases in emergency modes (Fig. 5.3-5.4), the performance of *NPM* drops drastically; its achieved Γ_{min} drops to zero around the 58th epoch for all three emergency modes. This implies that at least one node in the network fails to maintain non-empty energy storage and is forced to stop operation. The failure of these nodes will cause service interruption to the entire data collection application during the rest of the epochs. Such lasting service interruption is clearly unacceptable for mission-critical applications. In comparison both *HASS* approaches achieve much higher Γ_{min} than *NPM* (Fig. 5.3-5.4). This is because the *HASS* method allows energy-rich nodes run at faster speeds and therefore permits to allow the harvesting-poor nodes to slow down. The reduced speeds allow the weak nodes to maintain higher energy storage level, hence enhance the system's capacity to deal with emergencies.

Notice that *DHASS* achieves a Γ_{min} level very close to that of *CHASS*. In normal mode (Fig. 5.2), the achieved Γ_{min} of *CHASS* and *DHASS* almost overlap. In emergency modes, the performance difference between them increases slightly due to the increased influence of workload demands over energy level. This observation indicates that *DHASS* scheme can achieve near-optimal performance when it runs enough number of binary search iterations, as claimed in Theorem 2.

Table 5.1: Percentage of interrupted nodes: NPM, Non-aggregation

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	27.2%	31.6%	33.4%
$SPRD$	0%	0%	38.7%	41.6%	43.2%
$INST$	0%	0%	32.8%	53.9%	55.2%

I then conducted a *stress test* over the system while using different schemes. I raise the intensity of emergency by increasing the value of w from 1.5 to 3.0 with an increment of 0.5. The aim of this stress test is to evaluate the resilience of different schemes to various emergency intensities. I measure the system resilience to emergency in terms of the percentage of nodes that are interrupted by energy depletion. Note that if a node runs out of energy, all its descendent nodes would lose contact with BS, hence are also interrupted. The smaller the percentage of interrupted nodes under the same emergency intensity, the higher resilience supported by a scheme compared to others.

Table 5.1 gives the percentage of interrupted nodes in all three emergency modes under various emergency intensities, using *NPM* scheme. As seen from Table 5.1, as emergency intensity increases, the percentage of interrupted nodes increases noticeably in all modes when using *NPM*, implying the low resilience of the harvesting-unaware NPM scheme to emergency situations. I then found that though there are very limited number of nodes who ran out of energy in the network, large amount of nodes are also interrupted. This is because the depleted nodes are mostly close to BS which have large group of descendent

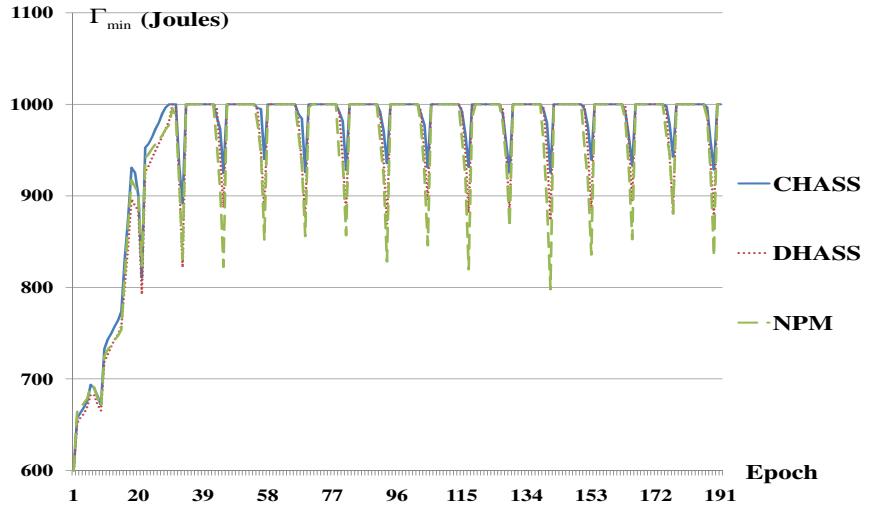


Figure 5.5: Min. energy level, complete aggregation: Normal mode

nodes and thereby high workload demands. The depletion of these nodes interrupts all their descendent nodes. While using *CHASS* and *DHASS*, the same increase in emergency intensity interrupts no node at all in the network. The results of the stress test demonstrates the benefit of my harvesting-aware approaches in mitigating the impact of emergencies over the system.

Aggregating applications

For aggregating applications, I repeat the same set of experiments conducted for non-aggregating applications. In Fig. 5.5-5.7, I also fix the emergency intensity at $w = 3.0$ and plotted the Γ_{min} achieved by using different schemes. In all the modes, *CHASS* and *DHASS* schemes again achieve much higher Γ_{min} than *NPM*. *DHASS* performance is very close to the *CHASS* scheme. As seen from all figures, the Γ_{min} achieved by the *HASS* approaches never drops to zero, while the achieved Γ_{min} by *NPM* schemes drops to zero many times. I then repeat the stress test. No nodes are interrupted when *HASS* scheme is used, while

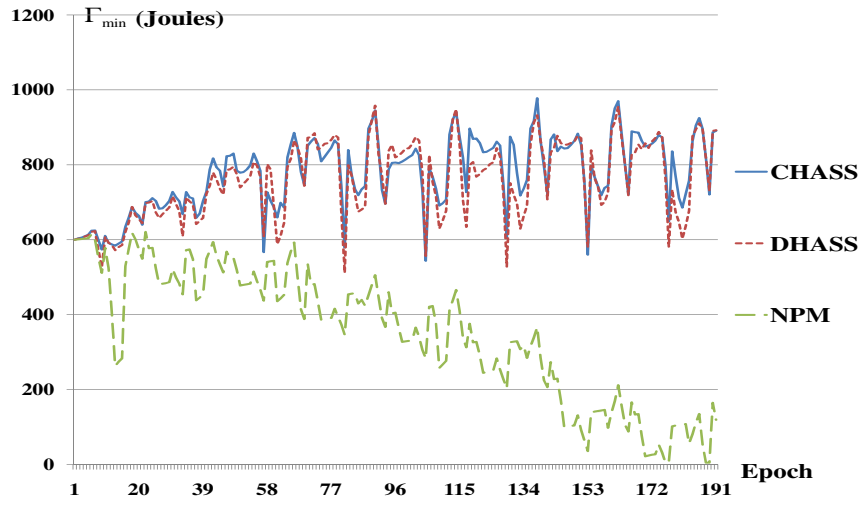


Figure 5.6: Min. energy level, complete aggregation: SPRD mode

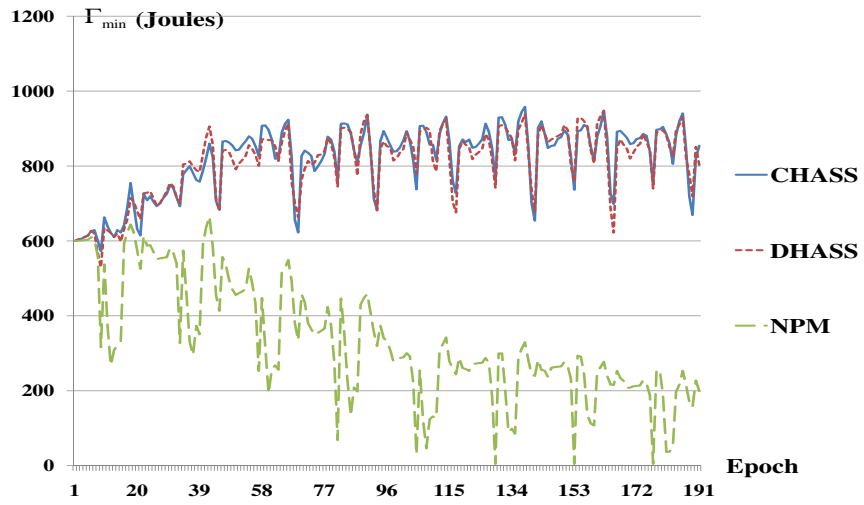


Figure 5.7: Min. energy level, complete aggregation: INST mode

large percentage of nodes are interrupted when emergency intensity increase to $w = 3.0$ for the INST and SPRD modes.

Table 5.2: Percentage of interrupted nodes using NPM - aggregation

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	0%	0%	26.9%
$SPRD$	0%	0%	0%	0%	49.1%
$INST$	0%	0%	0%	0%	47.5%

Next I present in Fig. 5.8 the achieved Γ_{min} using *CHASS* scheme for networks of different sizes (with 36, 64 and 100 nodes). As seen from the figure, the 36-nodes network has the highest Γ_{min}, followed by 64-nodes and 100-nodes networks. Although the difference of Γ_{min} among different networks is small, I observe that the smaller the network, the higher the value of Γ_{min}. This is because given the fixed end to end latency constraint (240 ms), nodes in large networks have more hops in a path to BS, thus the intermediate nodes have less time slack for processing and communication, thereby must use high computation and communication speeds.

In Fig. 5.9, I assume inaccurate prediction of harvested energy and workloads, and plot Γ_{min} values for SPRD attack, at emergency level $w = 3.0$ for aggregating applications. That is, the actual amount of energy harvested is up to 25% higher or lower than the harvesting prediction. Also, the actual computation and communication workloads, C and M are 25% lower than the workload prediction, as both C and M are the worst-case workloads and tasks may complete earlier. As seen from the figures, the HASS approaches still outperforms NPM under assumption of probabilistic energy harvesting and workloads, however the performance difference decreases as compared to Fig. 5.6 which share the same experiment setting but with accurate harvesting and workload predictions. This is because due to prediction inaccuracy the HASS schemes outputs sub-optimal speed assignment,

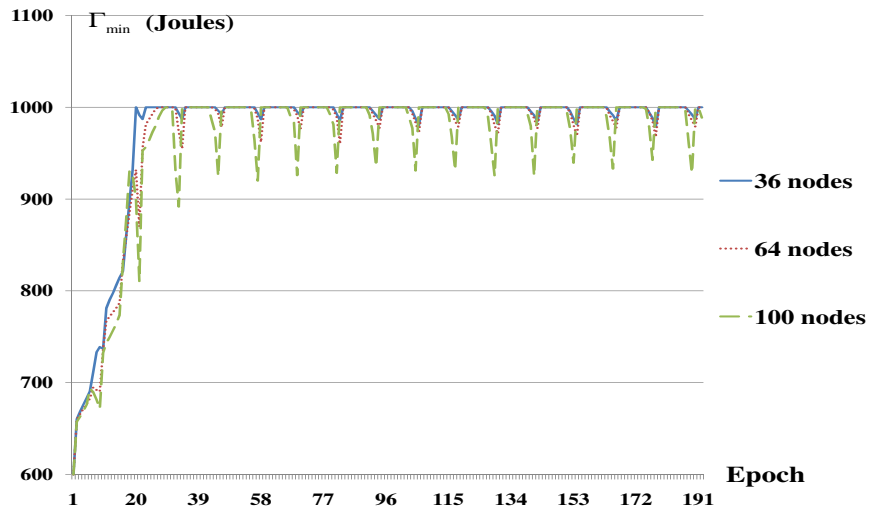


Figure 5.8: Min. energy level as a function of different network sizes - Normal, complete aggregation

while NPM scheme is not affected by prediction inaccuracy as it always uses the maximum speed configuration.

I calculated the communication overhead of *DHASS*. In the experiments, *DHASS* requires each node to send 1 initialization message, and 11 latency messages for the 11 computation rounds. Since *DHASS* needs to run only once in each epoch, each node sends in total 12 control messages in an epoch. Given a frame period $\pi = 240$ ms, and epoch length $S = 15$ minutes, each node executes 3750 frames and sends 3750 data messages. Therefore I derive the communication overhead to be $12/(3750+12) \approx 0.32\%$. This implies that *HASS* approaches have also the benefit of high efficiency.

Finally I derived that the CHASS algorithm outperforms the baseline algorithm NPM with close to 100% confidence. The computation and communication workloads in one frame are selected randomly (following uniform distribution) from the range $C = (0 - 3000000)$ cpu cycles and $M = (64 - 128)$ bytes respectively. The simulation is repeated 100 times. I collected the value of Γ_{min} achieved by both algorithms at the end of an epoch over all 100 simulation runs. I take the *difference* of the values of Γ_{min} by CHASS and NPM for

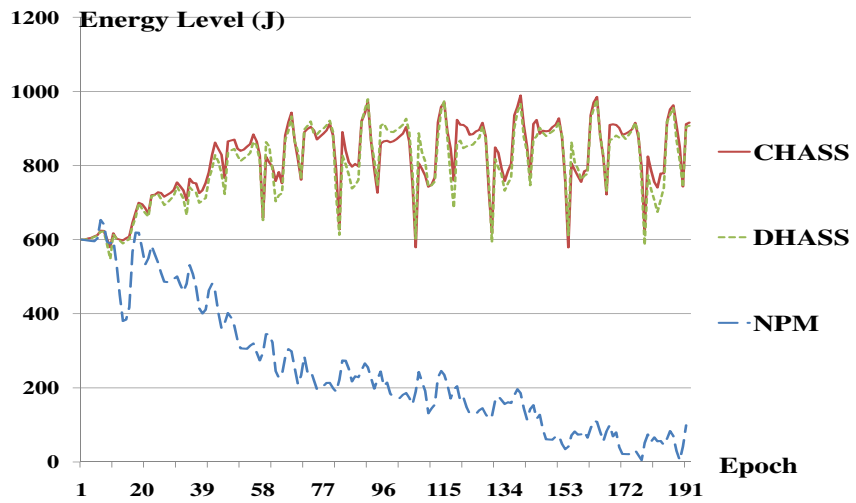


Figure 5.9: Min. energy level with probabilistic harvesting profile and workloads - SPRD, complete aggregation

each simulation run, and computed the confidence intervals of these Γ_{min} differences with different confidence levels. Since I repeated the experiment for 100 times, I used z-value for computing the confidence intervals [74]. I then found that zero is never included in the computed confidence interval even with close to 100% confidence level. This implies the CHASS algorithm outperforms NPM with close to 100% confidence [74].

The reason which the CHASS algorithm outperforms NPM with high confidence is that CHASS is formally proved to be optimal in term of maximization of Γ_{min}, and that the calculation of Γ_{min} and collection latency is based on Eq. 5.5 and Eq. 5.2 respectively (TOSSIM does not support DVS and DMS, and does not provide ways for computing these values). Therefore the achieved Γ_{min} by the baseline algorithm NPM cannot be higher than that is achieved by CHASS with any input values of computational or communication workloads.

Chapter 6: Maximal-Utility Rate Allocation

Chapter 4 and 5 target maximizing energy storage level of sensor nodes using DVS and DMS techniques while ensuring application performance requirements. This chapter targets maximizing the utility of WSN applications by assigning data collection rates to sensor nodes, given limited energy harvesting availability and minimum application performance requirement. It shares the same energy harvesting model, network topology and data collection paradigm to the work proposed in Chapter 4 and 5. However we do not use DVS and DMS to manage energy consumption. Rather we assume energy consumption is fixed for each invocation of sensing-processing-communication task while epoch-based energy consumption can be controlled by varying the invocation rate of tasks.

6.1 Network and application model

I consider general WSN applications that periodically collect data from N sensor nodes populated over the target environment. The nodes are organized into a data collection tree using any tree construction protocol, such as the Collection Tree Protocol (CTP) [71]. A sensor node is denoted as V_i, and the base station is denoted as BS. Within a tree-based routing structure, at any time each node V_i is connected to BS by a single path ρ_i consisting of zero or more intermediate nodes. I use the notation $V_i \in \rho_j$ to indicate that V_i resides on a given path ρ_j.

A node V_i senses the environment and sends the resulting data in packets towards BS along the path ρ_i, at rate r_i. These packets are referred as *internal packets*. Each node V_i may also forward *external packets* at a certain rate. These are received from the set of descendant nodes $\{V_j\}$, where V_i resides on V_j's path to BS, i.e. $V_i \in \rho_j$. Therefore the total outbound traffic at V_i equals $r_i + \sum_{j:V_i \in \rho_j} r_j$. I refer to r_i as V_i's *internal packet rate*,

and $\sum_{j:V_i \in \rho_j} r_j$ as its *external packet rate*.

Sensor nodes produce utility by sensing and reporting data to BS. Therefore I define the utility accrued by any node V_i as a function of its packet rate r_i, $U(r_i)$. U is a positive, non-decreasing and concave function. Then I define the aggregate utility U^{tot} accrued jointly by all N nodes in the network as:

$$U^{tot} = \sum_{i=1}^{N} U(r_i) \tag{6.1}$$

Now I define the energy consumption model which is different to that is defined earlier in 3.1 which assumes DVS and DMS capabilities. My energy model assumes, without loss of generality, that sensing and processing energy costs are fixed, relatively negligible and can be ignored. I therefore consider the radio transceiver as the main consumer for energy consumption. Packet routing results in two types of energy consuming activities, packet transmission and reception. I denote the energy spent on transmitting an internal or external packet as e^{tx}, where e^{tx} can be measured in advance. Packet reception also consumes energy. My architecture assumes that reception energy is controlled by the MAC layer, using techniques such as TDMA or duty-cycled LPL approaches such as B-MAC [11]. Therefore I model per-epoch reception energy as a constant E^{rx} which is dependent on the epoch length. I can calculate, at each node V_i, the total energy consumed E_i^c by handling internal and external packets in an epoch of length S as:

$$E_i^c = e^{tx} \cdot (r_i + \sum_{j:V_i \in \rho_j} r_j) \cdot S + E^{rx} \tag{6.2}$$

I also assume that each node V_i has a pre-assigned per-epoch energy budget B_i. Such energy budgets can be produced by algorithms such as [51,55,57], based on the predictions of harvested energy for any of the epochs in the horizon. Further, related work such as [16,21] require that a sensor node can consume no more than the amount of energy harvested in

any epoch. For these models a node's allocated energy budget in any epoch is set to the amount of energy it can harvest. Any of these energy budgeting algorithms guarantees the energy-neutral operation across all epochs.

6.2 The Maximal Utility Rate Allocation Problem

In this section, I define my rate allocation problem for energy harvesting WSN systems. My objective is to maximize the network utility U^{tot} (Eq. (6.1)), given the limited energy harvesting ability of nodes. I achieve this goal by adjusting packet rate r_i. Since the harvested power changes from epoch to epoch, the rates of nodes need to be re-adjusted in every epoch. I formulate this objective as an optimization problem called *Network Utility Maximization with Energy Harvesting* (*NUM-EH*) as:

$$Max \quad U^{tot} \tag{6.3}$$

$$s.t. \quad \forall V_i,$$

$$E_i^c \leq B_i \tag{6.4}$$

$$r_i \geq r^{min} \tag{6.5}$$

$$r_i + \sum_{j:V_i \in \rho_j} r_j \leq R_i^{cap} \tag{6.6}$$

The optimal solution to the above problem consists of the rates for all N nodes, i.e. $\{r_1, \ldots, r_N\}$ that maximizes U^{tot}. The constraint (6.4) enforces that the energy consumption E_i^c of any node V_i must be smaller than the assigned energy budget B_i in an epoch. The constraint (6.5) enforces that the rate of any node must be higher than the minimum required value r^{min}, in order to maintain the basic service level at individual nodes. Finally, to avoid packet congestion the total packet rate at any V_i, i.e. $r_i + \sum_{j:V_i \in \rho_j} r_j$ must be smaller than V_i's packet forwarding capacity denoted by R_i^{cap}.

I observe that *NUM-EH* is a concave maximization problem with three linear constraints. This is seen from Eq. (6.2), where E_i^c is a linear function of r_i and the set of external rates $\{r_j\}$. I notice that this problem is a special case of the well-known network utilization maximization problem [21, 58, 59] which can be solved using primal-dual based algorithms. However, such algorithms are typically too computationally expensive for resource-constrained sensor nodes. I instead propose a polynomial-time algorithm to solve the problem optimally and cost-effectively.

6.3 Rate Allocation Algorithm

In this section, I propose the algorithm MAX-UTILITY that optimally solves problem *NUM-EH*. MAX-UTILITY is applicable to arbitrary utility functions that are concave and non-decreasing. I first propose a centralized version of this algorithm that can be run on a base station. I then show how to implement MAX-UTILITY in a fully distributed fashion so resource-constrained sensor nodes can collaboratively produce optimal rate assignments.

First, I show that the constraint (6.4) and (6.6) can be combined into one single constraint as follows. Substituting the equality (6.2) into constraint (6.4) gives:

$$E_i^c = e^{tx} \cdot (r_i + \sum_{j:V_i \in \rho_j} r_j) \cdot S + E^{rx} \leq B_i \tag{6.7}$$

yielding:

$$r_i + \sum_{j:V_i \in \rho_j} r_j \leq \frac{B_i - E^{rx}}{e^{tx} \cdot S} \tag{6.8}$$

The right-hand side of the above inequality, $\frac{B_i - E^{rx}}{e^{tx} \cdot S}$, is a constant as B_i, E^{rx}, e^{tx} and S are all known constants. This can be combined with constraint (6.6) into one single

constraint, $r_i + \sum_{j:V_i \in \rho_j} r_j \leq CAPACITY_i$ where:

$$CAPACITY_i = Min\left\{R_i^{cap}, \frac{B_i - E^{rx}}{e^{tx} \cdot S}\right\} \tag{6.9}$$

I refer to $CAPACITY_i$ as the *rate capacity* of node V_i. I can re-write problem *NUM-EH* concisely as:

$$\begin{aligned} Max \quad & U^{tot} = \sum_{i=1}^{N} U(r_i) \\ s.t. \quad & \forall V_i, r_i \geq r^{min} && (6.10) \\ & \forall V_i, r_i + \sum_{j:V_i \in \rho_j} r_j \leq CAPACITY_i && (6.11) \end{aligned}$$

I refer to constraint (6.10) as the *min-rate constraint*, and constraint (6.11) as the *capacity constraint*.

6.3.1 The centralized version

Now I present the centralized version of algorithm MAX-UTILITY. Before the algorithm starts, the BS collects two pieces of information from each node. The first is the rate capacity $CAPACITY_i$, computed locally by each node using Eq. (6.9). The second is the node id of parent of each node, and the BS uses the parenthood relation of nodes to derive the structure of the existing data collection tree. MAX-UTILITY allocates rates as evenly as possible to nodes, while also satisfying constraints (6.10-6.11). Given an arbitrary concave, non-decreasing functions U, this will maximize network utility U^{tot}. This property is formally given later in Proposition 1. MAX-UTILITY runs in multiple iterations, and assigns rates to a subset of nodes in each iteration. The iteration ends when rates are assigned to all N nodes.

The algorithm has two lists of inputs and one output. The first input list contains N

rate capacities $CAPACITY_i$, one for each node V_i. The second list contains N vectors, one for each V_i. Denote the subtree rooted at node V_i by τ_i. The vector of each V_i contains all the nodes in the subtree τ_i rooted at V_i. This vector is derived based on the previously discovered tree structure. For notational simplicity I re-use τ_i to represent this vector. The output is the rate assignment vector derived by MAX-UTILITY, denoted by $\tilde{R}[]$.

MAX-UTILITY uses one global variable and three per-node variables that are updated from iteration to iteration. The global variable $ASSIGNED\_SET$ is a set containing all the nodes in the network that have been assigned rates so far. As will be seen later, in each iteration MAX-UTILITY adds at least one node to $ASSIGNED\_SET$. The algorithm terminates when all N nodes are in $ASSIGNED\_SET$. Next I declare three per-node variables. First, $CAP[i]$ is the remaining capacity of node V_i, initialized to $CAPACITY_i$. Second, $unassigned[i]$ is a subset of τ_i containing any nodes in τ_i that have not yet been assigned rates. $unassigned[i]$ includes V_i itself, and is initialized to τ_i. Finally, $r_c[i]$ is the maximum common rate for nodes in $unassigned[i]$. Table 6.1 summarizes these notations. This construction is illustrated in Fig. 6.1. In the 1st iteration, when none of nodes in subtree τ_7 rooted at V_7 have been assigned, $unassigned[7]$ contains all four nodes in τ_7, $\{V_1, V_2, V_6, V_7\}$. The last variable $r_c[i]$ is the maximum common rate that can be assigned to the nodes in $unassigned[i]$. $r_c[i]$ is computed by dividing V_i's remaining rate capacity $CAP[i]$ by $|unassigned[i]|$ which is the number of nodes in $unassigned[i]$, i.e. $r_c[i] = \frac{CAP[i]}{|unassigned[i]|}$. For example, in the 1st iteration, $r_c[7] = \frac{CAPACITY_7}{|unassigned[7]|} = \frac{16}{4} = 4$.

MAX-UTILITY, shown in Algorithm 7, is specified as follows. Line 2 initializes $ASSIGNED\_SET$ to $\emptyset$, $CAP[i]$ to $CAPACITY_i$, and $unassigned[i]$ to τ_i. In lines 3-5, for any node V_i I check whether it has sufficient capacity to sustain the minimum required rate r^{min} for all the nodes in its subtree τ_i. If there exists a node with insufficient capacity, then there is no feasible rate assignment that can satisfy the min-rate constraint and capacity constraint (at V_i) at the same time. At this point MAX-UTILITY terminates immediately with empty $\tilde{R}[]$. Line 6 starts the rate assignment loop which will terminate when all N

Algorithm 8 Algorithm MAX-UTILITY

```
- Input: {CAPACITY_i} and {τ_i}; Output: R̃[]
- Initialization: ASSIGNED_SET = ∅,
  ∀V_i, CAP[i] = CAPACITY_i, unassigned[i] = τ_i
for each node V_i in the network do
   If CAP[i] < r^min · |unassigned[i]|, then return ∅
end for
while |ASSIGNED_SET| < N do
   for any node V_i ∉ ASSIGNED_SET do
      r_c[i] = CAP[i]/|unassigned[i]|
   end for
   Find out V_u which has the least r_c[i] among any V_i ∉ ASSIGNED_SET
   for any node V_i ∈ unassigned[u] do
      Set R̃[i] = r_c[u], and add V_i to ASSIGNED_SET
   end for
   for any node V_i ∉ ASSIGNED_SET do
      if V_u ∈ unassigned[i], i.e. V_i is V_u's ancestor then
         CAP[i] = CAP[i] − r_c[u] · |unassigned[u]|
      end if
      if V_u ∈ unassigned[i] then
         for any node V_j ∈ unassigned[u] do
            Remove V_j from unassigned[i].
         end for
      end if
   end for
end while
return R̃[].
```

nodes are assigned. In lines 7-9, for any node $V_i \notin ASSIGNED\_SET$, I compute the maximum common rate $r_c[i]$ that can be assigned to the nodes in $unassigned[i]$ (line 8). In line 10, I find the node V_u which has the least $r_c[i]$ among any $V_i \notin ASSIGNED\_SET$. Then for any nodes in $unassigned[u]$, MAX-UTILITY assigns $r_c[u]$ to them and adds them to $ASSIGNED\_SET$ (line 11-13).

Table 6.1: List of notations

$CAP[i]$	The remaining rate capacity of node V_i
$unassigned[i]$	The set of unassigned nodes in V_i's subtree τ_i
$\|unassigned[i]\|$	The size of $unassigned[i]$
$ASSIGNED\_SET$	The set of assigned nodes
$\tilde{R}[]$	The rate assignment derived by MAX-UTILITY
$r_c[i]$	The maximum common rate for nodes in $unassigned[i]$
V_u	The node with the least $r_c[i]$ among all the unassigned nodes

Next, lines 14-23 update $CAP[i]$, and $unassigned[i]$ for any nodes without an assigned rate. Specifically, since any node V_i where $V_u \in unassigned[i]$ are ancestors of V_u and need to forward packets received from the nodes in $unassigned[u]$, I need to subtract $r_c[u] \cdot |unassigned[u]|$ (i.e. the total traffic that V_i received from V_u) from their $CAP[i]$ values (line 15-17). For any other nodes that are not ancestors of V_u, their $CAP[i]$ values remain the same. Moreover, any nodes $V_j \in unassign[u]$ with newly assigned rates must be removed from the $unassigned[i]$ sets of any node V_i that is an ancestors of V_u, i.e. $V_u \in unassigned[i]$ (line 18-22). The rate assignment process from line 7 to 23 continues until all N nodes are assigned.

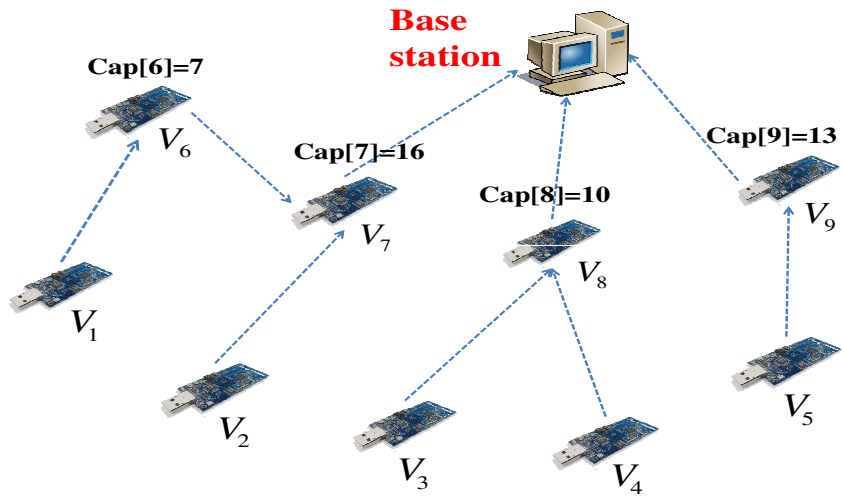


Figure 6.1: Example Data Collection Tree

As can be seen, in each iteration, MAX-UTILITY picks a V_u which has the least common rate $r_c[i]$ among any unassigned node V_i, and assigns $r_c[u]$ uniformly to any nodes in $unassigned[u]$, then produces a pruned tree by removing any newly assigned nodes. I refer to the node with the least common rate $r_c[i]$ among any nodes in a tree (or pruned tree) as the *critical node* of the tree. Note that in any iteration, the selected V_u is the critical node in the tree (pruned tree). The concept of critical node plays an important role in the optimality proof.

Steps of MAX-UTILITY are illustrated through an example. In Fig. 6.1, before the 1st iteration starts, $r_c[6] = \frac{7}{2} = 3.5$, $r_c[7] = \frac{16}{4} = 4$, $r_c[8] = \frac{10}{3}$, $r_c[9] = \frac{13}{2} = 6.5$. Line 10 picks V_8 as V_u, and line 12 assigns $\tilde{R}[3] = \tilde{R}[4] = \tilde{R}[8] = \frac{10}{3}$, and adds V_3, V_4, V_8 to $ASSIGNED\_SET$. Since this assignment does not affect any other nodes, lines 14-23 are not executed. In the 2nd iteration, V_6 is picked as V_u, so MAX-UTILITY assigns $\tilde{R}[1] = \tilde{R}[6] = 3.5$, and updates $CAP[7] = 16 - 7 = 9$, and $unassigned[7] = \{V_2, V_7\}$. V_1, V_6 are added to $ASSIGNED\_SET$. In the 3rd iteration, V_7 is picked as V_u as $r_c[7] = 9/2 = 4.5 <$

$r_c[9] = 6.5$. MAX-UTILITY assigns $\tilde{R}[2] = \tilde{R}[7] = 4.5$, marks V_2, V_7 assigned and makes no update. In the 4th iteration, V_9 is picked as V_u, MAX-UTILITY assigns $\tilde{R}[5] = \tilde{R}[9] = 6.5$, and adds V_5, V_9 to $ASSIGNED\_SET$. Since all the nodes are now in $ASSIGNED\_SET$, the algorithm terminates and returns $\tilde{R}[] = \{3.5, 4.5, 10/3, 10/3, 6.5, 3.5, 4.5, 10/3, 6.5\}$.

6.3.2 Optimality of algorithm MAX-UTILITY

In Eq. (6.1), the network utility U^{tot} is defined as the aggregate utility accrued jointly by all the nodes, $\sum_{i=1}^{N} U(r_i)$. Since $U(r_i)$ is concave, U^{tot} can be maximized by assigning packet rates to nodes as evenly as possible, while satisfying the min-rate, and capacity constraints at any nodes. This observation is supported by the following proposition, which can be proved based on the properties of concave functions:

Proposition 1. *Given n non-negative real numbers $\{r_1, \ldots, r_n\}$ where $\sum_{i=1}^{n} r_i \leq W$ (W is any non-negative constant), $\sum_{i=1}^{n} U(r_i)$ (U is a concave, non-decreasing function of r) is maximized when $r_1 = r_2 = \ldots = r_n = W/n$.*

Now I prove that the rate assignment $\tilde{R}[]$ derived by Algorithm MAX-UTILITY maximizes U^{tot}, while satisfying constraints (6.10-6.11). For the sake of contradiction, I assume that $\tilde{R}[]$ is not optimal, and that there exists an optimal assignment $R^*[]$ which differs from $\tilde{R}[]$. The network utility resulting from $R^*[]$ is denoted by $U^{tot,*}$. I will show that $R^*[] \neq \tilde{R}[]$ contradicts the optimality of $R^*[]$, hence proving that $R^*[i] = \tilde{R}[i]$ must hold for every node V_i in the network. I first propose Lemma 3, which is proved in the Appendix.

Lemma 3. *Given an arbitrary tree-based network, the optimal rate assignment $R^*[]$ must assign equal rates to all nodes in $unassigned[u]$, where V_u is the critical node of the tree.*

Lemma 3 implies that $R^*[]$ must assign a common rate denoted as $R^*[u]$ to all the nodes in $unassigned[u]$. I now present Lemma 4, which shows that $R^*[u]$ must be equal to $r_c[u]$, i.e. the maximum common rate that can be assigned to nodes in $unassigned[u]$.

Lemma 4. *Given an arbitrary tree-based network where V_u is the critical node in the tree, the optimal rate assignment $R^*[]$ must assign a common rate $r_c[u]$ to any nodes in $unassigned[u]$, regardless of the rates assigned to other nodes in the tree.*

The proof of Lemma 4 can be found in the Appendix. Since $\forall V_i \in unassigned[u], \tilde{R}[i] = r_c[u]$ holds (line 12), Lemma 4 implies that $\forall V_i \in unassigned[u]$, $R^*[i] = \tilde{R}[i]$ must hold. In other words $\tilde{R}[]$ derived by MAX-UTILITY is optimal for any nodes in $unassigned[u]$, where V_u is the critical node. Since assigning $R^*[u] = r_c[u]$ to any node $V_i \in unassigned[u]$ is optimal regardless of what rates are assigned to the rest of nodes, I can focus on finding the optimal rate assignment for a *partial tree* formed by disregarding any nodes in $unassigned[u]$ (i.e. removing nodes in $unassigned[u]$ from $unassigned[i]$ of any ancestors of V_u), and setting $CAP[i] = CAP[i] - r_c[u] \cdot |unassigned[u]|$ for any node V_i that is an ancestor of V_u. Note that the partial tree generated in this way contains exactly the set of nodes remaining unassigned at the beginning of the 2nd iteration of MAX-UTILITY. Also each node in the partial tree must have the same $CAP[i]$ and $unassigned[i]$ as they have in the 2nd iteration of MAX-UTILITY. Since Lemma 4 applies to an arbitrary tree, it holds for this partial tree as well. In other words, $R^*[]$ must assign a common rate $r_c[u']$ to any nodes $V_i \in unassigned[u']$, where $V_{u'}$ is the critical node of the partial tree.

Since the structure and capacity of the partial tree is the same as the updated tree in the 2nd iteration, the critical node $V_{u'}$ must be exactly the V_u picked by MAX-UTILITY in the 2nd iteration. Thus MAX-UTILITY must also assign $r_c[u']$ to any $V_i \in unassigned[u']$. Therefore, $\forall V_i \in unassigned[u']$, $\tilde{R}[i] = r_c[u']$ must hold. Since $\forall V_i \in unassigned[u']$, $R^*[i] = r_c[u']$ holds as implied by Lemma 4, I have $\forall V_i \in unassigned[u']$, $\tilde{R}[i] = R^*[i]$ must hold. That is, $\tilde{R}[]$ is optimal for any nodes in $unassigned[u']$. Again, I may disregard any nodes in $unassigned[u']$ and focus on the new partial tree formed in this way. The new partial tree has the same structure and capacities as the updated tree in the 3rd iteration of MAX-UTILITY. As iteratively pruning the tree (partial tree) and applying

Lemma 4 to the generated partial trees until all the nodes are pruned except the root BS, I have proved that the $\tilde{R}[]$ derived by MAX-UTILITY is completely identical to $R^*[]$, and therefore $\tilde{R}$ is optimal for the entire original tree.

Complexity analysis:

I show that for N nodes, the centralized MAX-UTILITY runs in $O(N^3)$ time. I first focus on one iteration of the while-loop from lines 6-24. The for-loop from lines 7 to 9 has complexity $O(N)$, since there are $O(N)$ unassigned nodes in each iteration. The selection of V_u in line 10 has complexity $O(N)$ as this is equivalent to finding the minimum element among $O(N)$ $r_c[i]$ values. The rate assignment to nodes in $unassigned[u]$ in lines 11-13 has complexity $O(N)$ as well, since the size of $unassigned[u]$ is $O(N)$. The for-loop from lines 14 to 23 has complexity of $O(N^2)$. In each loop, I need to update $CAP[i]$ and $unassigned[i]$ for $O(N)$ unassigned nodes V_i. Updating $CAP[i]$ for each V_i takes $O(1)$ time (line 16), while updating each $unassigned[i]$ take $O(N)$ time as it requires removal of $O(N)$ nodes (lines 19-21). Thus, the entire for-loop completes in $O(N^2)$ time as there are $O(N)$ nodes $V_i \notin ASSIGNED\_SET$. Therefore, each iteration of MAX-UTILITY completes in $O(N^2)$ time.

Next the number of iterations of MAX-UTILITY is given by $O(N)$, as the set $ASSIGNED\_SET$ is initially empty and each iteration adds at least one node to $ASSIGNED\_SET$. Therefore, the total complexity of MAX-UTILITY is $O(N^3)$. For practical purposes I expect that the number of iterations will be small. This is because the nodes that are close to the root have more descendent nodes, hence much larger $unassigned[i]$ than the nodes in the lower levels of the tree. Therefore the high level nodes usually have small $r_c[i]$ values and are more likely to be picked as V_u. Since once these nodes are picked as V_u, all nodes in their $unassigned[i]$ sets are assigned and added to $ASSIGNED\_SET$ at once in one iteration, the set $ASSIGNED\_SET$ will grow rapidly and the expected number of iterations of MAX-UTILITY should be small. I will justify this assertion through simulation analysis.

6.3.3 The distributed version

I now present the distributed version of MAX-UTILITY, referred as MAX-UTILITY-D. I decompose and distribute the computations in Algorithm 7 to every node in the network. The purpose of MAX-UTILITY-D is to support systems that do not possess or make use of a single resource-rich central control point. MAX-UTILITY-D only requires a single coordinator node such as a routing tree root. The tree root can be any node in the network. The root does not perform computations but only disseminates the minimum common rate $r_c[u]$ to all other nodes.

MAX-UTILITY-D is specified as follows. Initially, each node individually computes its rate capacity $CAPACITY_i$ using Eq. (6.9). Then MAX-UTILITY-D starts an initialization stage during which all the nodes send an empty control packet to the root. If a node V_i receives a control packet from another node V_j, then V_i knows V_j resides in its subtree τ_i. By the end of the initialization stage, each node knows its subtree τ_i, and hence the initial $unassigned[i]$. Using $CAPACITY_i$ and $|unassigned[i]|$, V_i computes the common rate $r_c[i]$ that can be assigned to nodes in $unassigned[i]$.

Each iteration of MAX-UTILITY-D consists of two stages. The first stage determines the minimum common rate $r_c[u]$ and the critical node V_u in the tree by requiring all the nodes forward their $r_c[i]$ values to the root. In the second stage, the root disseminates $r_c[u]$ across the network, and all nodes in $unassign[u]$ receive and use $r_c[u]$ as their packet rate. In the first stage, each leaf node in the tree V_i sends its computed $r_c[i]$ towards the root. V_i's parent V_j receives $r_c[i]$, compares $r_c[i]$ to $r_c[j]$ of itself, and forwards the larger of the two upwards. V_j also temporarily stores any received $r_c[i]$. This process proceeds over the entire tree in bottom-up fashion, and finally the root discovers the minimum common rate $r_c[u]$ in the tree. Then in the second stage the root disseminates $r_c[u]$ across the tree. When a node receives $r_c[u]$ from the root, it compares $r_c[u]$ to its local r_c to identify whether it is the critical node. If it is not the critical node, it then compares $r_c[u]$ to each of the previously stored r_c values of its descendent nodes to find out who is the critical node. In this way

the critical node V_u identifies itself, and commands all unassigned nodes in $unassigned[u]$ to use $r_c[u]$ as their rates for the coming epoch. This rate assignment command can be piggybacked in the packet for disseminating $r_c[u]$.

After the dissemination of $r_c[u]$ completes, all the nodes has learned the identity of the critical node V_u, either by comparing $r_c[u]$ to its own or stored r_c values, or from the rate assignment command issued by V_u. Finally, I need to update $CAP[i]$ and $unassigned[i]$ of the nodes remaining unassigned. If a node V_i is an ancestor of V_u, it has to update $CAP[i] = CAP[i] - r_c[u] \cdot |unassigned[u]|$ and then remove all nodes in $unassigned[u]$ from $unassigned[i]$. At the end of the iteration, each node discards the temporarily stored information and continues to the next iteration. In the next iteration, only the nodes remaining unassigned participate. The rate assignment iteration continues until all the nodes are assigned.

Complexity analysis:

In MAX-UTILITY-D, the tree root only disseminates $r_c[u]$. Each non-root node requires $O(N)$ comparisons in each iteration. This is because each non-root node compares its own r_c to $O(N)$ r_c values received from its descendent nodes (the 1st stage), and compares $r_c[u]$ disseminated by the root to its own r_c and $O(N)$ temporarily stored r_c. Further, updating $CAP[i]$ and $unassigned[i]$ takes $O(N)$ time for each node as described in the complexity analysis of centralized MAX-UTILITY.

Now I analyze the message complexity of the algorithm. The initialization stage requires one round of network-wide data collection. In each iteration, the determination of $r_c[u]$ requires all the nodes to report their $r_c[i]$ values, thus incurring another round of data collection. The root then announces $r_c[u]$ to all the nodes, incurring one round of network-wide data dissemination. In each data collection and dissemination round, each node sends exactly one packet. Given $O(N)$ iterations, MAX-UTILITY-D needs $O(N)$ rounds of network-wide data collection and dissemination. Finally, as mentioned in the analysis of the centralized MAX-UTILITY, the expected number of iterations is much lower than N in practice.

6.4 Performance Evaluation

Though I have formally proved the optimality of MAX-UTILITY, I have conducted a series of simulation experiments to evaluate its performance gain and overhead. I compare my algorithm against an alternative heuristic, called *Random Rate Augmentation* (RRA). The specification of RRA is given in the Appendix. By design, the rate assignment derived by RRA is also feasible in term of satisfying constraint (6.10-6.11). Although RRA is not optimal, it achieves reasonably high utility values as it maximizes total packet rate (flow) over the tree. Other algorithms, such as the one proposed in [21] are not directly comparable to ours, since their application and network model are different.

I compare the two rate allocation algorithms under various experiment settings that consider different energy budgeting schemes, utility functions, and network sizes. Without loss of generality, I consider solar-powered sensor networks in my simulation. The solar power harvesting profile is obtained from the Hamburg University of Technology [72]. I set the length of horizon to one day (24 hours) which is further divided into 96 epochs each with length of 15 minutes. I assume two energy budgeting schemes, denoted as Strictly Energy Neutral (SEN) and Maximum Uniform Budget (MUB). The SEN scheme assigns the exact amount of harvested energy as the energy budget for any epoch. For the evening epochs with extremely low harvested energy, SEN assigns a minimum energy budget equaling $5J$ to maintain the minimum rate r^{min}. The MUB scheme computes the maximum uniform energy budget across all 96 epochs while satisfying the min-rate and capacity constraints, and assigns this budget to every epochs. The MUB scheme uses the algorithm proposed in [51]. Note that the energy budget varies across epochs if SEN scheme is used, while the budget remains constant if MUB scheme is used. I am interested in evaluating how different budgeting schemes affect the network utility. This helps system designers in choosing the right energy budgeting scheme for their applications. I consider three different concave utility functions: $log(100r + 1)$ (denoted as LOG), $\sqrt{100r}$ (SQR) and $log[\sqrt{1000r} + 1]$ (denoted as composite utility or CPST). I use the coefficients 100 and 1000 with r because

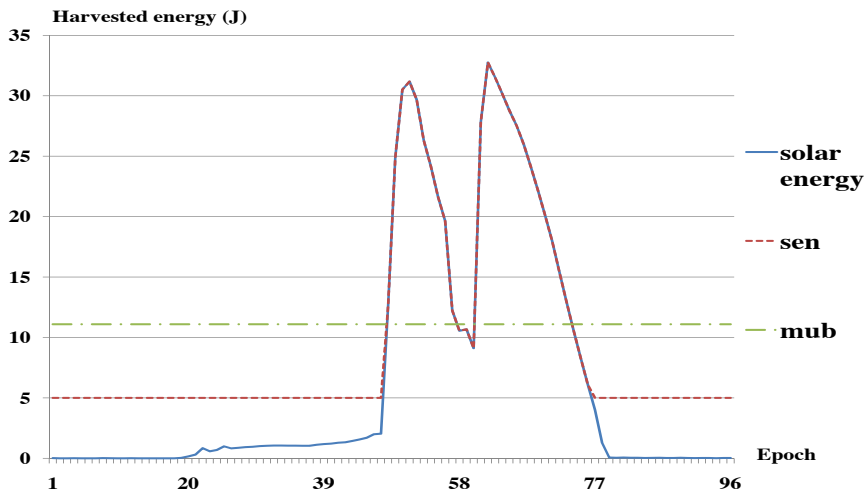


Figure 6.2: Solar energy harvested throughout a day.

r is commonly small. Finally, I repeat my simulation in six networks of different size, containing 25, 36, 64, 100, 169, 225 nodes respectively. I examine the accrued utilities, and the algorithm overhead as the network grows.

The sensor nodes are organized into a collection tree using the CTP protocol. The energy consumption for transmitting a packet is randomly selected from the range $(0.05 - 0.1)J$. This range is obtained based on the measurements in [73]. The energy storage device has capacity of 1000J. The initial energy level in the horizon is set to 500J. The minimum rate r^{min} is set to 0.01, i.e. 1 packet per 100 seconds.

6.4.1 Simulation results

I evaluated algorithm MAX-UTILITY and RRA in TOSSIM/TinyOS. I present my evaluation results along four dimensions: accrued network utility U^{tot}; rate (r_i) distribution among nodes; energy level variation across epochs of two selected nodes; algorithm running time and control overhead. I calculate the energy level Γ_k of a node at the end of the kth

epoch using the formulas below:

$$\Gamma_1 = Min.\{\Gamma^{init} + P_1^h \cdot S - E_1^c, \Gamma^{max}\}$$

$$\Gamma_k = Min.\{\Gamma_{k-1} + P_k^h \cdot S - E_k^c, \Gamma^{max}\}$$

Γ^{init} is the initial energy level. Γ^{max} is the energy capacity. Recall that P_k^h is the harvested power, S is the epoch length, and E_k^c is the energy consumed in epoch k (Eq. (6.2)). Γ_{k-1} is the ending energy level in the previous epoch $k-1$, which is also the starting energy level in epoch k. $P_k^h \cdot S$ gives the energy harvested in epoch k.

In Fig. 6.2, I show the solar energy trace, along with the energy budget assignments derived by SEN and MUB schemes. As seen from the figure, the amount of solar energy reaches its peak in the afternoon due to high sunlight intensity. The line representing the energy budget assignment derived by SEN overlaps the line representing the solar energy in noon and afternoon. This is because SEN assigns the amount of harvested energy as the energy budget for any epochs, except for the evening epochs in which the harvested energy is less than 5J. Finally, MUB assigns the highest uniform budget (around 11J) for any epochs.

6.4.2 Network utility

In Fig. 6.3 and 6.4, I compare the utility accrued by MAX-UTILITY and RRA algorithm. I show the results for the 100-node network. In each figure I plot the utility values in all 96 epochs, for all three utility functions. In each figure, I plot six utility lines, one for each combination of the three utility functions and two rate allocation algorithms.

Fig. 6.3 assumes SEN energy budgeting scheme. Since SEN scheme assigns the harvested energy as the energy budget, I observe that the variations in accrued utility indicate similar pattern as the harvested energy (Fig. 6.2). Specifically the utility reaches its peak in the afternoon since high solar energy availability leads to high rate capacity $CAPACITY_i$ at any nodes. For any utility functions, MAX-UTILITY (OPT) achieves significantly higher utility

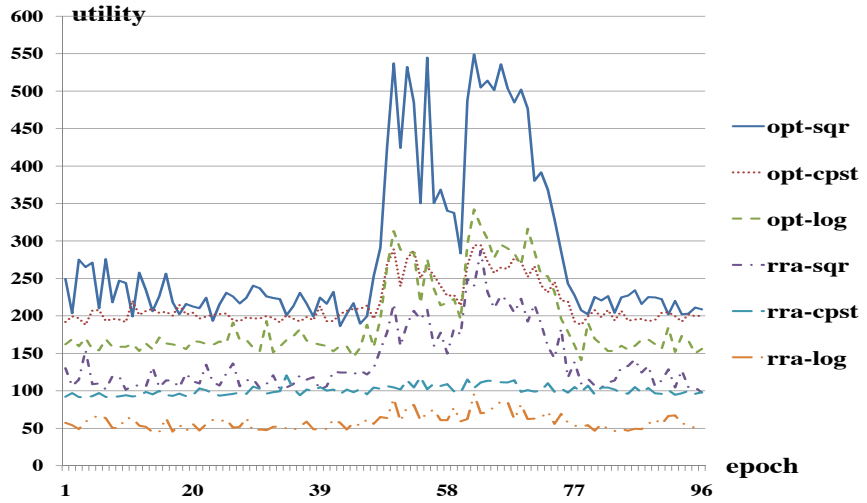


Figure 6.3: Utility accrued by max-utility (opt) and rra, with different utility functions, SEN budgeting.

than RRA. Specifically for SQR, the utility value increases above 500 in the afternoon if MAX-UTILITY is used. However, if RRA is used, the utility accrued never increases above 300. For the rest of the day, MAX-UTILITY achieves utility figures that are mostly above 200, while RRA achieves utility figures that are around 100.

In Fig. 6.4, I repeat the simulation for MUB energy budgeting. Because the MUB scheme is used, the energy budget B is uniform across different epochs, thus yielding constant rate capacities at nodes and much more stable utility performance in all six lines. Again, MAX-UTILITY achieves much higher utility than RRA. For SQR, the utilities accrued by MAX-UTILITY remain around 350 throughout the day, while the one for RRA remains around 150. As seen from Fig. 6.3 and 6.4, I observe that the MUB scheme maintains stable and high utility at any time and is hence suitable for the applications which desire utility stabilization. On the other hand, SEN energy allocation achieves very high utility when harvesting ability is high and is suitable for the applications where maximum utility is sought.

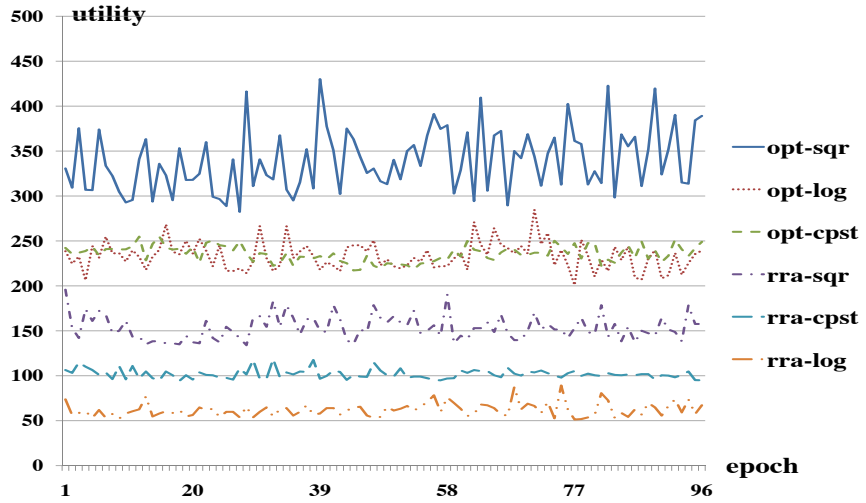


Figure 6.4: Utility accrued by max-utility (opt) and rra, with different utility functions, MUB budgeting.

6.4.3 Rate assignment and energy storage level

In this section, I show the resulting rate assignment and energy storage levels of the two algorithms. First, I plot in Fig. 6.5 the distribution of rates of all the nodes in the 100 nodes network at midnight, while using the SEN scheme and the CPST utility function. From the figure, I observe that MAX-UTILITY tends to distribute rates more evenly across nodes than RRA. The rate assignment by MAX-UTILITY has standard deviation of 0.1268, while for RRA the deviation equals 0.1936. In Fig. 6.6(a-b) I plot the energy levels of a heavily-loaded node and a lightly-loaded node in 96 epochs. The heavily-loaded node is high in the tree, thus has higher workload and energy demand and lower energy level. As seen from the figure, the energy neutral condition is maintained for both nodes at all times.

6.4.4 The impact of network size

Now I study the impact of network size on performance and overhead of MAX-UTILITY. In Fig. 6.7, I increase the size of the network from 25 nodes to 225 nodes, and plot the

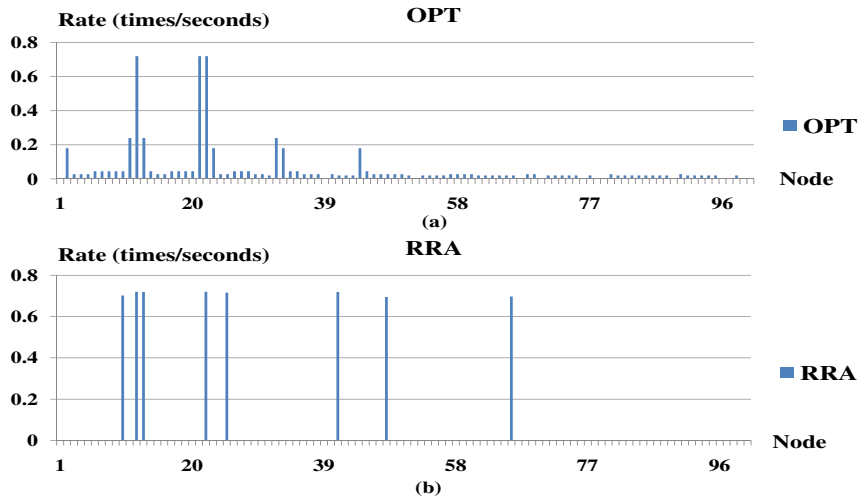


Figure 6.5: Distribution of rate (times/seconds) of all the nodes at midnight by (a) MAX-UTILITY; (b) RRA.

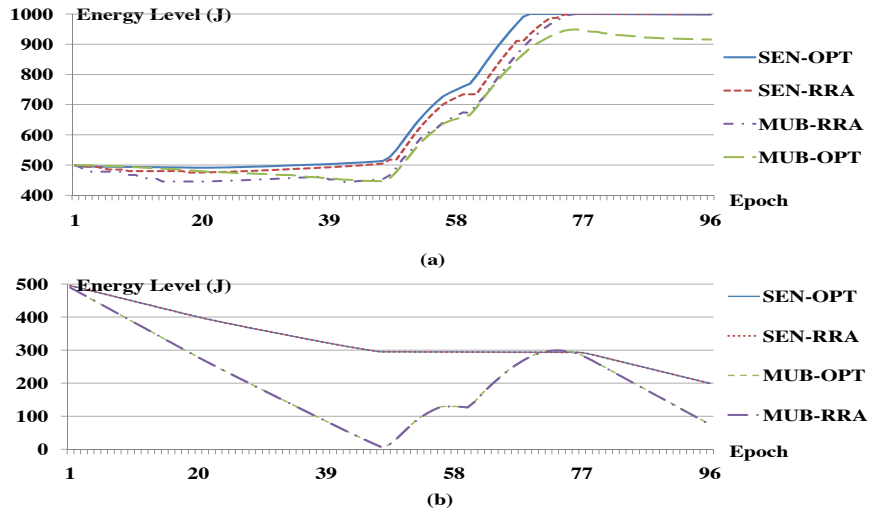


Figure 6.6: Variation of Energy levels over 96 epochs of (a) a lightly-loaded node; (b) a heavily-loaded node.

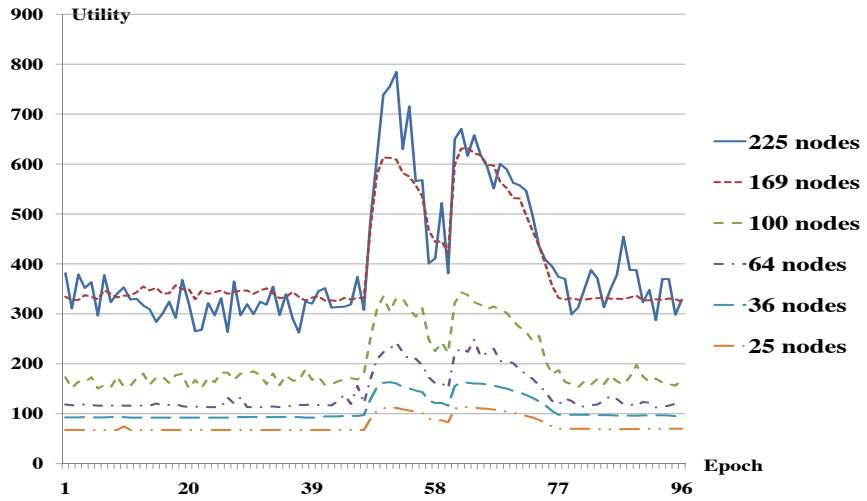


Figure 6.7: Total utility accrued by MAX-UTILITY in networks of different size

utility accrued by MAX-UTILITY. This simulation assumes SEN energy budgeting and LOG utility. I observe from the plot that the utility grows steadily as the network grows from 25 nodes to 169 nodes. As computed, the average utility is 78.10 for 25 nodes network, 109.34 for 36-nodes network, 142.84 for 64-nodes network, 203.79 for 100-nodes network, 396.54 for 169-nodes network, respectively. However, as the network size increases to 225, the utility increases to only 403.48. In other words the utility growth almost stops. This is because that, although there are potentially more utility contributors in the 225-nodes network, the achievable utility is constrained by the nodes that are close to the tree root, since the rate capacity of these nodes does not change with the network size. System designers must be aware of this phenomenon while choosing the network size and routing patterns.

I measure the actual running time of MAX-UTILITY in terms of the number of iterations shown in Table 6.2. Again, I assume 6 network sizes. As seen from the table, the number of iterations does not increase with the network size. On the contrary, I observe that the 25-node network requires the largest number of iterations, 9.97, while the 64-nodes network

requires the smallest number of iterations, 6.99. Based on the tree structure I observe that the number of iterations depends heavily on the number of children the tree root has. As mentioned in 6.3.2, the children of the root have large group of descendent nodes, hence are more likely to be picked as the critical node V_u. Once they become V_u, all their descendent nodes are added to the set $ASSIGNED\_SET$ in one iteration. This leads to the rapid growth of $ASSIGNED\_SET$.

I count the average number of control packets sent by a node in each invocation of the distributed MAX-UTILITY to be 27.32 in 100-nodes network. If the node operates at rate $r_i = 1$ packet per second then 900 data packets will be sent in an epoch. The packet overhead is then $27.32/900 \approx 3\%$. This indicates that the MAX-UTILITY-D is an entirely feasible alternative to MAX-UTILITY.

Table 6.2: The number of iterations of MAX-UTILITY as a function of network size

Num. of nodes	25	36	64	100	169	225
Num. of iterations	9.97	7.99	6.99	7.99	8.99	8.99

Finally I derived that our algorithm MAX-UTILITY outperforms the rival algorithm RRA with close to 100% confidence. The energy consumption for transmitting one packet is selected randomly (following uniform distribution) from the range $(0.05 - 0.1)Joule$. The simulation is repeated 50 times. I collected the accrued utility value by both algorithms at the end of an epoch over all 50 simulation runs. I then take the *difference* of the utility values of MAX-UTILITY and RRA for each simulation run, and computed the confidence intervals of these *utility differences* with different confidence levels. Since I repeated the experiment for 50 times, I used z-value for computing the confidence intervals [74]. I then found that zero is never included in the computed confidence interval even with close to 100% confidence level. This implies MAX-UTILITY outperforms RRA with close to 100% confidence [74].

The reason which MAX-UTILITY outperforms RRA with high confidence is that MAX-UTILITY is formally proved to be optimal in term of utility maximization, and that the calculation of energy consumption, rate capacity and utility value is based on Eq. 6.2, Eq. 6.9 and Eq. 6.1 respectively (TOSSIM does not provide ways for computing these values). Therefore the accrued utility by the rival algorithm RRA cannot be higher than that is accrued by MAX-UTILITY with any input values of per-packet energy consumption.

Chapter 7: Future Work

This chapter describes my future research work. A major part of future work is to implement on real sensor motes the energy management algorithms proposed in chapter 5 and 6 and maximal utility rate allocation algorithm proposed in chapter 6. This allows evaluating real-world performance of the algorithms. The detailed implementation of the algorithms can be found in the Appendix.

Another part of my future work is to consider probabilistic workload model for DVS-DMS based energy management. Probabilistic workload can be represented in two ways. First, an entire communication or computation task might not exist possibly because the sensed data is not of interest to the application or the sensed value is beyond the meaningful range. The probability which a task is to be executed can be given as Pr_i. Second, the workload of a communication or computation task, i.e. M and C is probabilistic where the worst case workload can still be estimated. The probability which the workload of a task is M or C can be given as $Pr(M)$, $Pr(C)$. The energy management algorithm must consider the workload unpredictability while achieving the objective.

Chapter 8: Conclusion

My dissertation proposes energy and performance management solutions for energy harvesting Wireless Sensor Networks. I first present an architecture for energy harvesting WSN systems which contains an epoch-based energy harvesting model, a DVS-DMS capable device and energy consumption model, a network topology model and data collection paradigm, a performance-sensitive WSN application model, and utility model for data collection WSN systems.

I then propose joint DVS-DMS energy management approaches for loosely-coupled and tightly-coupled WSN systems. The approaches maximize the energy storage level of sensor nodes by adjusting radio modulation levels and CPU frequencies while satisfying all the application performance requirements. Through this objective we ensure highly resilient performance under both normal and emergency situations. I formulate this objective as optimization problems, and solved them optimally with centralized and distributed algorithms. Through simulation we show our algorithms achieve significantly higher performance than a baseline approach under both normal and emergency situations.

Another important portion of my dissertation addresses a utility maximization problem for energy harvesting sensor networks. Utility is defined as concave and non-decreasing function of nodes sensing rate. I proposed MAX-UTILITY, a rate allocation algorithm to maximize the total utility achieved jointly by all the nodes, while ensuring energy neutrality for any nodes. I formally proved the optimality of the algorithm, and indicated that the algorithm can derive optimal rate assignment in $O(N^3)$ time where N is the network size. Finally, extensive simulation results demonstrate its superior performance.

Chapter 9: Appendix

Proof of Theorem 1

Let $\gamma_{i,j}^G$ and $\gamma_{i,j}^A$ denote the ending energy levels in frame (i, j), obtained by iteratively maximizng energy level increment of each frame, and that obtained by an arbitrary scheme A. Similarly, we denote $\Delta\gamma_{i,j}^G$ and $\Delta\gamma_{i,j}^A$ as the energy level increments in frame (i, j), obtained by our iterative scheme and scheme A. The initial energy level in epoch i is denoted as $\Gamma_i^{init} = \Gamma_{i-1}$. We will prove the theorem by induction over the frame sequence number, j.

Base case: If j=1, we have $\gamma_{i,1}^G = \Gamma_i^{init} + \Delta\gamma_{i,1}^G$, $\gamma_{i,1}^A = \Gamma_i^{init} + \Delta\gamma_{i,1}^A$. Since $\Delta\gamma_{i,1}^G \geq \Delta\gamma_{i,1}^A$ is valid by definition, $\gamma_{i,1}^G \geq \gamma_{i,1}^A$ is justified.

Now, suppose the statement holds for $j = 1, 2 \ldots n-1$ frames in epoch i. Based on our induction assumption, we have $\gamma_{i,n-1}^G \geq \gamma_{i,n-1}^A$. We claim that Theorem 1 also holds in frame (i, n), i.e., $\gamma_{i,n}^G \geq \gamma_{i,n}^A$. We distinguish two cases:

- $\gamma_{i,n}^G = \Gamma^{max}$: the energy level reaches Γ^{max} at the end of frame (i, n). Since the energy level achieved by any scheme A cannot exceed Γ^{max}, our claim holds.

- $\gamma_{i,n}^G < \Gamma^{max}$: Note that if $\gamma_{i,n}^G < \Gamma^{max}$, the optimal energy increment obtained by G after considering the constant harvested power and worst-case workload in frame (i, n) is not constrained by the maximum capacity constraint (otherwise $\gamma_{i,n}^G$ would be equal to Γ^{max}). This enables us to deduce that $\Delta\gamma_{i,n}^G \geq \Delta\gamma_{i,n}^A$, since regardless of the initial energy level at frame (i, n), G by definition accumulates energy which is at least equal to that yielded by any other scheme A during frame (i, n), as long as the constraint $\gamma_{i,n}^G < \Gamma^{max}$ is not violated. Also recall that by induction assumption, we

have $\gamma^G_{i,n-1} \geq \gamma^A_{i,n-1}$. Therefore, we have $\gamma^G_{i,n} = \gamma^G_{i,n-1} + \Delta\gamma^G_{i,n} \geq \gamma^A_{i,n-1} + \Delta\gamma^A_{i,n} = \gamma^A_{i,n}$. Thus, our claim holds as well.

Hence, we proved Theorem 1.

Proof of Lemma 1:

We denote the optimal solution of problem *HASS-N* as S^N, and its corresponding minimum energy level as Γ^N_{min}. S^N is optimal in the sense that it maximizes Γ_{min}, without considering constraint (5.9). We denote the optimal solution of problem HASS as S^*, and its corresponding minimum energy level as Γ^*_{min}. We show that, if S^N satisfies the positivity constraint which also means $\Gamma^N_{min} > 0$, then we have $S^* = S^N$; otherwise, S^* does not exist. We consider the following three cases:

- If in S^N, $0 < \Gamma_i \leq \Gamma^{max}$, $\forall\, V_i$, obviously, we have $S^* = S^N$.
- In case that S^N leads to $\Gamma_i > \Gamma^{max}$, $\exists V_i$, we claim that the compute and communicate speeds contained in S^N can be still used. This is because as soon as the maximum capacity of the energy storage is reached, the harvesting circuitry can be automatically turned off, keeping its energy level at Γ^{max}. In another words, in this case, we still have $S^* = S^N$.
- If S^N leads to $\Gamma_i \leq 0$, $\exists i$, this implies $\Gamma^N_{min} \leq 0$. Assume S^* exists, then we have $\forall V_i$, $\Gamma_i > 0$ in S^*, since S^* must satisfy the positivity constraint by definition. This indicates $\Gamma^*_{min} > 0 \geq \Gamma^N_{min}$ which contradicts the fact that S^N maximizes Γ_{min} (recall that the feasible region of *HASS* is contained in that of *HASS-N*). Therefore, if S^N violates the positivity constraint, S^* cannot exist.

Proof of Lemma 2:

Let (f_1, d_1) and (f_2, d_2) denote the fastest speed configurations at V_i found when the algorithm FS is invoked by using Γ_1 and Γ_2 as input respectively, where $\Gamma_1 \geq \Gamma_2$. We use

$l_{i,1}^{min}$ and $l_{i,2}^{min}$ to denote the (least) per-node latencies at V_i obtained by using (f_1, d_1) and (f_2, d_2), respectively. We will show $l_{i,1}^{min} \geq l_{i,2}^{min}$.

When the FS problem is solved with Γ_1 as input, (f_1, d_1) yields $l_{i,1}^{min}$, while satisfying $\Gamma_i(f_1, d_1) \geq \Gamma_1$. When it is solved with Γ_2 as input, the function *find_fastest* could at least find $(f_2, d_2) = (f_1, d_1)$ which yields $l_{i,2}^{min} = l_{i,1}^{min}$, while satisfying $\Gamma_i(f_2, d_2) \geq \Gamma_2$ (because $\Gamma_i(f_2, d_2) = \Gamma_i(f_1, d_1) \geq \Gamma_1 \geq \Gamma_2$). In many cases, (f_2, d_2) will yield an even smaller $l_{i,2}^{min}$.

Proof of Theorem 2:

Denote Γ^{low} and Γ^{high} in the Yth round as $\Gamma^{low,Y}$ and $\Gamma^{high,Y}$, respectively. According to the property of binary search, in the Yth round, the maximum Γ_{min} which is our search target is confined to the range $[\Gamma^{low,Y}, \Gamma^{high,Y}]$. As a result, the maximum Γ_{min} can be larger than the Γ_{min} found in the Yth round by at most $\Gamma^{high,Y} - \Gamma^{low,Y}$. In the Yth round, $\Gamma^{high,Y} - \Gamma^{low,Y} = (Max(EL) - Min(EL))/2^{Y-1}$.

Proof of Lemma 3:

For the purpose of contradiction, we assume that, in $R^*[]$, nodes in $unassigned[u]$ are assigned unequal rates $\{\mu_1, \ldots, \mu_{|unassigned[u]|}\}$. However, as shown by Proposition 1 if this assumption holds we can always further increase $U^{tot,*}$ by equalizing $\{\mu_1, \ldots, \mu_{|unassigned[u]|}\}$ to a common rate $R^*[u] = \frac{\mu_1 + \ldots + \mu_{|unassigned[u]|}}{|unassigned[u]|}$. This contradicts the optimality of $R^*[]$. Therefore, *equal rate assignment to nodes in* $unassigned[u]$ *is a necessary condition for the optimality of* $R^*[]$.

We still need to show such rate equalization does not violate the min-rate and capacity constraints (Eq. (6.10-6.11)). This is justified as follow. First, the min-rate constraint is not violated, because the new common rate $R^*[u]$ must be larger than the minimum of $\{\mu_1, \ldots, \mu_{|unassigned[u]|}\}$, which is in turn larger than r^{min} (due to the feasibility of $R^*[]$). The rate capacity constraint is also satisfied. This is because V_u is the critical node with the least common rate r_c among any nodes in the tree. Thus, $r_c[u]$ cannot be larger than $r_c[i]$ of any node V_i in $unassigned[u]$. Further, $R^*[u] \leq r_c[u]$ must hold, since if $R^*[u] > r_c[u]$ we

have $R^*[u] \cdot |unassigned[u]| > r_c[u] \cdot |unassigned[u]|$, which gives $\mu_1 + \ldots + \mu_{|unassigned[u]|} > CAP[u]$. This implies that $R^*[]$ violates the capacity constraint which contradicts the feasibility of $R^*[]$. Since $R^*[u] \leq r_c[u]$, and $r_c[u] \leq r_c[i]$, $\forall V_i \in unassigned[u]$ we know the common rate $R^*[u]$ cannot be larger than the maximally allowed common rate $r_c[i]$ of any node V_i in $unassigned[u]$, hence cannot violate the capacity constraint at these nodes. Finally, the capacity violation cannot happen to any nodes outside $unassigned[u]$ since the total packet flow at V_u does not change after the rate equalization. Therefore, we have proved that in $R^*[]$, any nodes in $unassigned[u]$ must use a common rate.

Proof of Lemma 4:

In Lemma 3, we have shown that in $R^*[]$ all nodes in $unassigned[u]$ must be assigned a common rate $R^*[u]$, where V_u is the critical node in the tree. Now we show that this common rate $R^*[u]$ must be equal to $r_c[u]$ in order for $R^*[]$ to be optimal. Recall that $r_c[u] = \frac{CAP[u]}{|unassigned[u]|}$. Notice that nodes in $unassigned[u]$ cannot be assigned a common rate $R^*[u]$ that is higher than $r_c[u]$ as this will exceed V_u's capacity, so $R^*[u] > r_c[u]$ cannot hold. Therefore we assume $R^*[u] < r_c[u]$. We show this assumption cannot hold either, and thereby $R^*[u] = r_c[u]$ must hold.

If $R^*[u] < r_c[u]$ holds, we assert that the rate capacity of V_u is not fully utilized. This is because $R^*[u] < r_c[u]$ means

$$R^*[u] \cdot |unassigned[u]| < r_c[u] \cdot |unassigned[u]| = CAP[u]$$

Since the left-hand side of the above inequality gives the total packet rate at V_u, if $R^*[u]$ is assigned to every nodes in $unassigned[u]$, this inequality implies V_u's capacity $CAP[i]$ is under-utilized.

Given that V_u's capacity is under-utilized, we make another assertion that there exists a node V_k, an ancestor of V_u, that must have its rate capacity fully utilized. This is because if all the ancestors of V_u are also under-utilized, then we can further increase $U^{tot,*}$ by

increasing $R^*[u]$ by an amount of $\epsilon/|unassigned[u]|$, where $\epsilon > 0$ is the strictly positive, minimum un-utilized capacity among any ancestors of V_u. This contradicts the optimality of $R^*[]$. If there are multiple such V_k nodes, we consider the one that is in the lowest level of the tree.

Given the existence of V_k, we assert that in $R^*[]$ there must exist a node $V_j \in \tau_k$ and $V_j \notin unassigned[u]$ which has rate $R^*[j] > R^*[u]$. (Recall that τ_k is the subtree rooted at V_k.) This is because if no node in τ_k has rate higher than $R^*[u]$, then the total packet rate at V_k must be strictly smaller than $R^*[u] \cdot |\tau_k|$ ($|\tau_k|$ is the number of nodes in τ_k), and in turn no larger than $r_c[k] \cdot |\tau_k| = CAP[k]$, i.e. V_k's capacity. $R^*[u] \cdot |\tau_k| \leq r_c[k] \cdot |\tau_k|$ holds because $R^*[u] < r_c[u]$ is assumed, and $r_c[u] \leq r_c[k]$ since V_u has the least r_c among any nodes in the tree. This implies the capacity of V_k is under-utilized which contradicts our previous assertion that V_k is fully utilized. Thus we justify the existence of such V_j.

Next, given $R^*[j] > R^*[u]$, we know that the marginal utility of the concave utility function U at point $R^*[j]$ is lower than that at $R^*[u]$. Therefore we can further increase $U^{tot,*}$ by decreasing $R^*[j]$ by a strictly positive amount $\epsilon^{'}$, and increasing $R^*[u]$ for each node in $unassigned[u]$ by the amount of $\frac{\epsilon^{'}}{|unassigned[u]|}$, as long as $R^*[j] > R^*[u]$ holds. In other words, we can always increase $U^{tot,*}$ by reducing the difference between $R^*[j]$ and $R^*[u]$. This rate adjustment cannot violate the min-rate constraint as $R^*[u] > r^{min}$. The capacity constraint is also not violated, as the decrease of $R^*[j]$ does not increase the total packet rate at any nodes. The increase of $R^*[u]$ by $\frac{\epsilon^{'}}{|unassigned[u]|}$ will not exceed the capacities of any V_u's ancestors inside τ_k, as long as $\epsilon^{'}$ is no larger than the minimum unused capacity of any ancestors of V_u in τ_k. This is because V_k is the lowest ancestor of V_u whose capacity is fully utilized, hence any ancestors of V_u inside τ_k (i.e. below V_k) must have positive unused capacity. Further, the total packet rate at any V_u's ancestors outside τ_k does not change after the rate adjustment, so their capacity constraints cannot be violated either.

Therefore, the feasible increase of $U^{tot,*}$ contradicts the optimality of $R^*[]$, hence justifying that $R^*[u] < r_c[u]$ cannot hold, and $R^*[u] = r_c[u]$ must hold in order for $R^*[]$ to be optimal. Therefore we have proved Lemma 4.

Specification of the RRA algorithm

The RRA algorithm initially allocates r^{min} to every node, and updates the remaining capacity of nodes accordingly. If this violates the capacity constraints at some nodes, then we know there exists no feasible assignments for the given network. Otherwise, we start from that assignment and progressively improve U^{tot} by augmenting the rates of nodes with unused capacity. Specifically, based on the initial assignment, RRA randomly selects a node V_i with positive remaining capacity, and determines the maximum possible rate increment which V_i and its ancestors can afford. Then RRA adds up this increment to V_i's current rate, and zeros out its remaining capacity. RRA updates the remaining capacity of other nodes accordingly. This is the end of one round of augmentation. We then continue augmenting other nodes until no node has remaining capacity. The resulting rates of nodes is the final utility achieved by RRA.

Implementation of Centralized and Distributed MAX-UTILITY

9.1 Specification of Maximal-Utility Rate Allocation Protocol

Chapter 6 proposes *MAX-UTILITY*, a rate allocation algorithm for maximizing application utility in energy harvesting WSN systems. Both centralized and distributed versions of the algorithm are proposed, namely *MAX-UTILITY-C* and *MAX-UTILITY-D*. For my future work I would like to implement *MAX-UTILITY* on real sensor motes.

9.1.1 The Centralized Protocol

The centralized *MAX-UTILITY* runs on the base station. First the base station has to collect from every node the information necessary for calculation of the optimal rate assignment. This includes the parent node ID, and the rate capacity $CAPACITY_i$. The parent ID is used for discovering the structure of the data collection tree, so as to derive $unassigned[i]$ for every node V_i. $CAPACITY_i$ is calculated individually by each node. These two pieces of information are used to calculate the maximum common rate $r_c[i]$.

First the base station sends a *collect_info* message to all the nodes for collecting information from them. The nodes response with a *collect_info reply* containing the requested information. To ensure collection of all the necessary information, a data collection protocol that guaranteeing reliable end to end data delivery must be used. A reliable collection protocol must support message acknowledgement and message retransmission upon transmission failure. Reliable delivery can be enforced either between each pair of communicating nodes or in an end-to-end fashion.

The base station starts calculation of algorithm MAX-UTILITY after it hears back from every node. The specification of the algorithm can be found in Chapter 7. *MAX-UTILITY* outputs the max-utility rate assignment. The base station has to disseminate the derived rate assignment as message *rate_assignment* to all the nodes. The rate assignment is essentially a set of $(node\_ID, rate)$ mappings. Note that regular sensor nodes are not involved in the execution of the algorithm.

Below I formally define the two message types: *collect_info* for data collection, and *rate_assignment* for disseminating rate assignment. Both messages contain a *type* field expressed as 4-bytes integer which indicates the message type. Message *collect_info* contain two additional fields, *parent_id* and *capacity* which are the information requested by base station. All three fields are expressed as 4-bytes integer. For *collect_info* request, these two fields are left empty and padded with meaningless value. A *rate_assignment* message also contains two additional fields (both are in 4-bytes integer), the ID of the node to be assigned and the rate to be assigned. Therefore both *collect_info* and *rate_assignment* have

three fields with a payload size of 12 bytes.

Since a sensor network may comprise hundreds of nodes, and every node needs to be assigned a rate, the entire rate assignment could be a large chunk of data. On another hand, the standard payload size of WSN message (≈ 30 bytes by default) is quite limited. The rate assignment has to be divided into multiple parts (that can fit into a standard message) and disseminated separately. The number of parts (messages) is proportional to the number of nodes in the network. For example, if the network size is 100, the total size of network-wide rate assignment is 1200 bytes which has to be disseminated in at least ≈ 40 separate *rate_assignment* messages. For a network of size 1000 nodes, the rate assignment size increases to 12000 bytes which must be packed into ≈ 400 separate messages. Therefore an efficient way for rate assignment dissemination must be used.

Next I present an efficient way for disseminating rate assignment to nodes. First note that since the rate assigned to a given node is only needed by that node, it is unnecessary to disseminate the rate to other nodes. Second, as mentioned in the specification of *MAX-UTILITY*, any nodes in a same node group $unassigned[u]$ identified by the critical node V_u are assigned a common rate i.e. $r_c[u]$. Because of these two properties, the base station needs to unicast the rate $r_c[u]$ to only the critical node V_u, and from there the rate is broadcast to the nodes in the node group $unassigned[u]$. Recall that any nodes in the node group $unassigned[u]$ are descendent nodes of the critical node V_u, the rate value is relayed node by node in downward direction following the subtree rooted at V_u. Note that this avoids unnecessary dissemination of $r_c[u]$ to nodes not in $unassigned[u]$. However, since only the base station knows the member nodes in node group $unassigned[u]$ while node V_u has no clue at all, the problem is then to stop further relaying of $r_c[u]$ when the message reaches the boundary of $unassigned[u]$.

The above problem can be solved using the following two observations. First as per its definition, node groups in a data collection tree are separated by their critical nodes. Furthermore, the only way to reach the nodes in a given node group is through its critical node. Each node group has only two neighboring node groups, the one where its critical

node connects to (sitting above it); and the one connects to it (sitting below it). Therefore the downwards relaying of a *rate_assignment* message within a given node group can stop when it reaches the critical node of a neighboring node group (i.e. the boundary of two node groups).

9.1.2 The Distributed Protocol

Now I present a distributed protocol denoted by *MAX-UTILITY-D*. It consists of one round of tree structure discovery stage where every node discover the structure of the subtree rooted at itself (stage 1), followed by multiple rate assignment iterations (stage 2). The base station does not have to collect any information from sensor nodes in advance. It only plays a role of coordinator.

MAX-UTILITY-D defines four types of message exchanged among nodes as below:

- *structure_discovery*: This message is sent by the base station and sensor nodes to perform tree structure discovery. If the message is sent by base station, then it is a command for nodes to start structure discovery. Any nodes receive the command response with a *structure_discovery reply*. The original sender of a message can be learned from the message header. The payload of this message only contains one field, a *type* field (4-bytes integer) that is set to *structure_discovery* (or *reply*).

- *start_assignment*: This message is sent by the base station as a command to start a new rate assignment iteration. This message only contains a *type* field set to *start_iteration*.

- *rate*: This message is sent by a regular node to report the minimum maximum common rates r_c among itself and all its descendent nodes. This message contains three fields, a *type* field set to *rate*, the value of r_c (in $(4, 8)$-bytes integer), and the ID (4-bytes integer) of the node who has the minimum r_c value. Note that r_c might be a floating number, however most WSN node platforms do not support this data type, the nodes have to multiply the floating number by a number that is multiple of 10 to make it

integer.

- *rate_assignment*: This message is sent by the base station to disseminate the value of $r_c[u]$. It contains a *type* field set to *rate_assignment*, the value of $r_c[u]$ (in $(4, 8)$-bytes integer), and *critical_node* giving the ID of critical node V_u (4-bytes integer).

Every node needs to maintain a set $unassigned[i]$ and its remaining rate capacity. I specify the tree structure discovery stage step by step as below:

- The base station announces the start of this stage by sending a *structure_discovery* message over the tree.
- Every sensor node responses to the received *structure_discovery* message by sending a *structure_discovery* reply towards the base station.
- Any node receiving a *structure_discovery* reply adds the original sender (not the relay sender) of the reply message to its set $unassigned[i]$. Note that a node receives *structure_discover* reply messages only from its descendant nodes, thus by the end of stage 1 the set $unassigned[i]$ is constructed.
- Finally every node computes its local rate capacity $CAPACITY_i$, and maximum common rate $r_c[i] = \frac{CAPACITY_i}{|unassigned[i]|}$.

Next I give the steps in one rate assignment iteration:

- The base station announces the end of stage 1, and the start of the first rate assignment iteration, by disseminating a *start_assignment* message.
- After received a *start_assignment* message, the non-leaf nodes simply pass the message down to its children, while the leaf nodes response to the message by sending its maximum common rate $r_c[i]$ included in a *rate* message up towards the base station.
- A non-leaf node receives *rate* messages from its children, compares its own $r_c[i]$ with the ones received, then forwards the minimum one along with the ID of node who has $r_c[i]$ up to its parent.

- When base station receives *rate* messages from all its children, it knows which node has the minimum r_c value (i.e. the critical node) among any nodes in the network (i.e. V_u), and the value of $r_c[u]$ (critical rate). Then it disseminates a *rate_assignment* message to nodes which contains these two pieces of information.

- A node identifies itself as the critical node V_u, or it learns who is the critical node after receiving the *rate_assignment* message. The critical node V_u and any nodes in $unassigned[u]$ use $r_c[u]$ as their rate for the coming epoch. One problem is how can the nodes in $unassigned[u]$ know V_u is their critical node.

  This problem is solved by making V_u to modify the *critical_node* field of to zero (non-existing node id) before passing down the *rate_assignment* message. The nodes in $unassigned[u]$ could learn V_u is their critical node if the *critical_node* field equals zero.

- Update of rate capacity and $unassigned[i]$: Any node V_i that is an ancestor of V_u must update their rate capacity and $unassigned[i]$ according to line 14-23 in algorithm spec. This is done by deducting the total assigned rate to nodes in $unassigned[u]$, and removing any nodes in $unassigned[u]$ from the set $unassigned[i]$.

- After an interval of time after base station disseminates *rate_assignment* message, it assumes the end of the current iteration, then starts a new iteration by disseminating a new *start_iteration* message. Only the nodes remaining unassigned participate in the new iteration. Base station knows all the nodes in the network have been assigned if no *rate* message is received.

Bibliography

Bibliography

[1] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Wireless sensor networks for structural health monitoring," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*. New York, USA: ACM, 2006, pp. 427–428.

[2] J. Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri, "A wireless sensor network for structural health monitoring: performance and experience," in *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, Washington, DC, USA, 2005.

[3] W. L. Hwang, F. Su, and K. Chakrabarty, "Automated design of pin-constrained digital microfluidic arrays for lab-on-a-chip applications," in *Design Automation Conference*, San Francisco, California, 2006.

[4] M. Lin, Y. Wu, and I. Wassell, "Wireless sensor network: Water distribution monitoring system," in *Radio and Wireless Symposium, 2008 IEEE*, Orlando, Florida, 2008.

[5] I. F. Akyildiz, D. Pompili, and T. Melodia, "Challenges for efficient communication in underwater acoustic sensor networks," *ACM SIGBED Rev.*, vol. 1, no. 2, pp. 3–8, 2004.

[6] B. Peleato and M. Stojanovic, "A mac protocol for ad-hoc underwater acoustic sensor networks," in *Proceedings of the 1st ACM international workshop on Underwater networks*, ser. WUWNet '06, 2006.

[7] J. L. Hill and D. E. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, 2002.

[8] C. Park, J. Liu, and P. H. Chou, "Eco: an ultra-compact low-power wireless sensor node for real-time motion monitoring," in *Proceedings of the 4th IEEE international symposium on Information processing in sensor networks*, 2005, p. 54.

[9] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.*, ser. Infocom 02', New York NY.

[10] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-mac: a hybrid mac for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 511–524, 2008.

[11] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY: ACM, 2004.

[12] J.-H. Chang and L. Tassiulas, "Maximum lifetime routing in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 4, pp. 609–619, 2004.

[13] O. Younis and S. Fahmy, "Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Transactions on Mobile Computing*, vol. 3, pp. 366–379, 2004.

[14] G. S. A. Kumar, G. Manimaran, and Z. Wang, "End-to-end energy management in networked real-time embedded systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 11, pp. 1498–1510, 2008.

[15] Y. Yu, V. Prassana, and B. Krishnamachari, "Energy minimization for real-time data gathering in wireless sensor networks," *Wireless Communications, IEEE Transactions on*, vol. 5, no. 11, pp. 3087 –3096, november 2006.

[16] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. on Embedded Computing Sys.*, vol. 6, no. 4, p. 32, 2007.

[17] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proceedings of the 4th IEEE international symposium on Information processing in sensor networks*, 2005.

[18] F. Simjee and P. Chou, "Everlast: Long-life, supercapacitor-operated wireless sensor node," in *Proceedings of the 2006 IEEE International Symposium on Low Power Electronics and Design*, 2006.

[19] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Comput. Netw.*, vol. 51, no. 4, pp. 921–960, 2007.

[20] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, New York, NY, 2001.

[21] R. Liu, P. Sinha, and C. E. Koksal, "Joint energy management and resource allocation in rechargeable sensor networks," in *the 29th IEEE Conference on Information Cmmunications*, San Diego, California, 2010.

[22] L. Su, Y. Gao, Y. Yang, and G. Cao, "Towards optimal rate allocation for data aggregation in wireless sensor networks," in *the 12th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. (Mobihoc'11), Paris, France, 2011.

[23] X. Wang and K. Kar, "Cross-layer rate control for end-to-end proportional fairness in wireless networks with random access," in *the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. (Mobihoc'05), Urbana-Champaign, Illinois, 2005.

[24] T. Bokareva, W. Hu, S. Kanhere, B. Ristic, T. Bessell, M. Rutten, and S. Jha, "Wireless sensor networks for battlefield surveillance," in *Proc. of the Land Warfare Conference*, 2006.

[25] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, New York, NY, 2002.

[26] R. G. Harish Ramamurthy, B. S. Prabhu and A. M. Madni, "Wireless industrial monitoring and control using a smart sensor platform," in *IEEE Sensors Journal*, vol. 7, no. 5, 2007, pp. 611–618.

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

[28] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4th ACM international conference on Embedded networked sensor systems*, New York, NY, 2006.

[29] K. A. Arisha, "Energy-aware tdma-based mac for sensor networks," in *IEEE Workshop on Integrated Management of Power Aware Communications, Computing and Networking*, New York, NY, 2002.

[30] X. Yang and N. H. Vaidya, "A wakeup scheme for sensor networks: Achieving balance between energy saving and end-to-end delay," *Real-Time and Embedded Technology and Applications Symposium, IEEE*, vol. 0, pp. 19–29, 2004.

[31] J. Pan, Y. T. Hou, L. Cai, Y. Shi, and S. X. Shen, "Topology control for wireless sensor networks," in *Proceedings of the 9th ACM annual international conference on Mobile computing and networking*. New York, NY: ACM, 2003.

[32] J. A. Zhenzhen Ye, A. A. Abouzeid, "Optimal policies for distributed data aggregation in wireless sensor networks." 26th IEEE International Conference on Computer Communications, 2007.

[33] C. Moser, L. Thiele, L. Benini, and D. Brunelli, "Real-time scheduling with regenerative energy," in *ECRTS '06: Proceedings of the 18th Euromicro Conference on Real-Time Systems*, Washington, DC, 2006.

[34] S. Liu, Q. Qiu, and Q. Wu, "Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting," in *Proceedings of the ACM conference on Design, automation and test in Europe*, 2008.

[35] N. H. Zamora, J.-C. Kao, and R. Marculescu, "Distributed power-management techniques for wireless network video systems," in *Proceedings of the conference on Design, automation and test in Europe*, Nice, France, 2007.

[36] P. Shah, T. Shaikh, K. Ghan, and S. Shilaskar, "Power management using zigbee wireless sensor network," in *Emerging Trends in Engineering and Technology. ICETET. First International Conference on*, Nagpur, India, 2008.

[37] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "Vigilnet: An integrated sensor network system for energy-efficient surveillance," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 1–38, 2006.

[38] M. H. T. A. Z. Skolicki, M. Wadda, "Reduction of physical threats to water distribution systems," *ASCE Journal of Water Resources Planning and Management*, no. 4, pp. 211–217, 2006.

[39] M. H. K. D. J. Z. Skolicki, T. Arciszewski, "Co-evolution of terrorist and security scenarios for water distribution systems," *Advances in Engineering Software*, vol. 39, no. 10, pp. 801–811, 2008.

[40] I. Stoianov, L. Nachman, S. Madden, and T. Tokmouline, "Pipeneta wireless sensor network for pipeline monitoring," in *Proceedings of the 6th IEEE international conference on Information processing in sensor networks*, 2007.

[41] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Adaptive power management for environmentally powered systems," *IEEE Transactions on Computers*, vol. 59, no. 4, pp. 478 – 491, 2010.

[42] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, 2004.

[43] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, Washington, DC, 2006.

[44] C. Rusu, R. Melhem, and D. Mossé, "Multi-version scheduling in rechargeable energy-aware real-time systems," *J. Embedded Comput.*, vol. 1, no. 2, pp. 271–283, 2005.

[45] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, 1994.

[46] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science*, Washington, DC, 1995.

[47] C. Schurgers, O. Aberthorne, and M. Srivastava, "Modulation scaling for energy aware communication systems," ser. ISLPED 01', 2001.

[48] V. Devadas and H. Aydin, "Real-time dynamic power management through device forbidden regions," in *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, DC, 2008.

[49] C. tai Yeh, Z. Fan, and R. Gao, "Energy-aware data acquisition in wireless sensor networks," ser. IMTC 07', Warsaw, Poland, 2007.

[50] Y. Yu and V. K. Prasanna, "Energy-balanced multi-hop packet transmission in wireless sensor networks," in *Globecom*, San Francisco, CA, 2003.

[51] C. Moser, J.-J. Chen, and L. Thiele, "Reward maximization for embedded systems with renewable energies," in *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.

[52] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Robust and low complexity rate control for solar powered sensors," in *Proceedings of the conference on Design, automation and test in Europe*, 2008.

[53] K.-W. Fan, Z. Zheng, and P. Sinha, "Steady and fair rate allocation for rechargeable sensors in perpetual sensor networks," in *the 6th ACM conference on Embedded network sensor systems*, ser. (SenSys'08), Raleigh, North Carolina, 2008.

[54] L. Lin, N. B. Shroff, and R. Srikant, "Asymptotically optimal energy-aware routing for multihop wireless networks with renewable energy sources," *IEEE/ACM Trans. Netw.*, vol. 15, no. 5, pp. 1021–1034, 2007.

[55] D. K. Noh, L. Wang, Y. Yang, H. K. Le, and T. Abdelzaher, "Minimum variance energy allocation for a solar-powered sensor system," in *Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems*, 2009.

[56] C. Vigorito, D. Ganesan, and A. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2007.

[57] D. K. Noh and K. Kang, "A practical flow control scheme considering optimal energy allocation in solar-powered wsns," in *the 18th IEEE International Conference on Computer Communications and Networks*, ser. (ICCCN'09), Washington, DC, 2009.

[58] S. H. Low, "A duality model of tcp and queue management algorithms," *IEEE/ACM Trans. on Networking*, vol. 11, pp. 525–536, 2002.

[59] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun*, vol. 24, pp. 1439–1451, 2006.

[60] C. Rusu, R. Melhem, and D. Mossé, "Maximizing the system value while satisfying time and energy constraints," in *the 23rd IEEE Real-Time Systems Symposium*, ser. (RTSS'02), Washington, DC, 2002.

[61] C. Moser, J.-J. Chen, and L. Thiele, "Power management in energy harvesting embedded systems with discrete service levels," in *the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. (ISLPED'09), San Fancisco, California, 2009.

[62] D. Li and P. H. Chou, "Maximizing efficiency of solar-powered systems by load matching," in *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 2004, pp. 162–167.

[63] R. Aydin, Hakan Melhem, D. Mosseé, and P. Mejía-Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 50, no. 2, pp. 111–130, 2001.

[64] C. M. S. Mokhtar S. Bazaraa, Hanif D. Sherali, *Nonlinear Programming: Theory and Algorithms*, 3rd ed. Wiley, 2006, p. 872.

[65] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st ACM international conference on Embedded networked sensor systems*, ser. SenSys '03, 2003.

[66] Memsic, "Imote2 datasheet," *www.memsic.com*.

[67] Marvell, "Xscale pxa27x datesheet," *www.marvell.com*.

[68] Chipcon, "Chipcon cc2420 datasheet," *focus.ti.com/docs/prod/folders/print/cc2420.html*.

[69] taosinc, "Taos tsl2561 datasheet," *www.alldatasheet.com/datasheet-pdf/pdf/203054/TAOS/TSL2561.html*.

[70] EPANET 2.0, "Water supply and water resources," *US Environmental Protection Agency*, 2010.

[71] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *the 6th ACM conference on Embedded network sensor systems*, ser. Sensys'05, Berkeley, California, 2009.

[72] C. Renner, J. Jessen, and V. Turau, "Lifetime prediction for supercapacitor-powered wireless sensor nodes," in *the 8th GI/ITG KuVS Fachgesprch "Drahtlose Sensornetze"*, Hamburg, Germany, 2009.

[73] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *the 6th ACM conference on Embedded network sensor systems*, ser. Sensys'04, Baltimore, MD, 2004.

[74] R. Jain, "The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling," in *Wiley*, 1991, pp. 204–209.

Curriculum Vitae

Bo Zhang receives his BS. degree from Huazhong University of Science and Technology, Wuhan, China; and MS. degree from University of Cincinnati, Cincinnati, OH, both in computer science. His research interests include wireless sensor networks, low-power embedded systems. He is the recipient of Best Paper Award of 2011 ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM'11).