UNIVERSITY OF CALIFORNIA, SAN DIEGO

Energy and Task Management in Energy Harvesting Wireless Sensor Networks for Structural Health Monitoring

A Thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Jamie Bradley Steck

2009

DEDICATION

To my amazing grandmother, Jean “Nana” Baron, with love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first like to thank my parents and my husband for their continuous encouragement and unconditional love. You have given me a glimpse of how much God loves me. I would also like to thank all my professors, company officers, and senior enlisted at the U.S. Naval Academy, specifically CDR Osborn USN, COL Athens USMCR, and MAJ Bishop, USMC, for encouraging me to attend graduate school and for showing me how to be a leader of character.

Additionally, I would like to thank my research advisor, Tajana Simunic Rosing, for her interest in this work and continual guidance. I would like to thank all the current members of SEE Lab: Edoardo, Diana, Gaurav, Ayse, Giacomo, Priti, Raid, Shervin, Denis, and Todor. Their company and sense of humor enabled me to push through the tough times with a dose of laughter and cynicism. Finally, I cannot thank Joaquin enough for his mentorship when I first began on this project, as I know I would not have come this far without it.

I would also like to thank Los Alamos National Laboratory for their support and motivation for this work. The research presented within this thesis was funded by Los Alamos National Laboratory through colloboration with UCSD, for which I am thoroughly appreciative. The structural engineers from UCSD and LANL, both students and faculty, provided the motivation and resources to accomplish this work. The input and explanations from Eric Flynn and David Mascarenas were invaluable, and I am extremely grateful.

Chapter 4, in part, has been submitted for publication of the material as it may appear in Networked Sensing Systems, 2009, Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this paper. Chapter 4, in part, is currently being prepared for submission for publication of the material. Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this material.

Chapter 5, in part, has been submitted for publication of the material as it may appear in Networked Sensing Systems, 2009, Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this paper. Chapter 5, in part, is currently being prepared for submission for publication of the material. Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this material.

ABSTRACT OF THE THESIS

Energy and Task Management in Energy Harvesting Wireless Sensor Networks
for Structural Health Monitoring

by

Jamie Bradley Steck

Master of Science in Computer Science

University of California, San Diego, 2009

Professor Tajana Simunic Rosing, Chair

Energy harvesting sensor nodes reduce the need for post-deployment physical human interaction by using environmental power and wireless communication; however, they must adapt performance to accommodate the energy availability. This thesis presents three application-independent algorithms that adapt performance based on energy availability for steady and external trigger state conditions. Steady state operation describes the periodic execution of a set of tasks on the system. For steady state operation, a method is presented that adapts the execution rate to achieve high performance while maintaining sufficient energy. External trigger state operation occurs when an external device makes a request to the system. For external trigger state operation, algorithms are used to determine the execution time, energy consumption and performance of the request. These methods are applied to SHiMmer, a wireless, energy-harvesting structural health monitoring platform. Unlike other sensor systems that periodically monitor a structure and route information to a base station, SHiMmer is designed to acquire data using active sensing and process it locally before communicating with an external device. Results from this application demonstrate the controller's ability to adapt at runtime and maintain sufficient energy. Steady state results show that the execution rate changes with weather conditions. On average, the execution rate on a sunny day increases by 62% compared to the rate on cloudy days. External trigger state results show that processing significantly affects the efficiency of a structural health monitoring system; specifically, complex processing requires 17 times less execution time and 2.5 times less energy than transmitting raw data.

Chapter 1

Introduction

During rush-hour traffic on August 1, 2007, the I-35W bridge in Minneapolis, Minnesota collapsed, resulting in 13 deaths. After intense review and analysis, the US National Transportation Safety Board found the cause to be inadequate design of the gusset plates that failed that day, causing the unexpected catastrophe. Among other recommendations found in the accident report [24], the board advised the Federal Highway Administration to require the use of nondestructive evaluation technologies to be implemented in addition to current visual inspection methods.

Due to the need for nondestructive evaluation and in response to advances in sensor technology, sensor networks are being employed to enhance the visual inspection required for all US government-owned bridges. These networks use sensors and data processing algorithms to measure material or geometric qualities of a bridge and indicate a possible failure or need for repair that visual inspections may miss.

Along with bridge inspection, many applications are in need of smart wireless sensor networks for remote monitoring. A significant challenge for these types of networks is energy consumption. Energy harvesting is a promising technique to significantly enhance the lifetime of sensor networks by gathering energy from the environment. However, harvesting energy from the environment requires strict energy and task management in order to maintain energy neutrality - the amount of energy consumed must equal the amount of energy harvested.

This thesis presents the design of a controller for an energy-harvesting, wireless sensing system that adapts performance to maintain energy neutrality. The design is applied to a structural health monitoring application, showing how energy neutrality can be maintained by adapting performance. This work has been motivated by three promising research areas: wireless sensor networks, energy harvesting, and structural health monitoring, each of which will be described in more detail in the following sections.

1.1 Wireless Sensor Networks

As a result of the increase in computing power and decrease in chip cost and size, the emergence of small, inexpensive devices has enabled the development of wireless sensor networks. A wireless sensor network (WSN) is composed of a collection of small, wireless nodes with sensing technology, and the scope of use for WSNs is continuously expanding, from environmental and structural monitoring to robotics. For example, WSNs are used for habitat monitoring on Great Duck Island in Maine [11] and for underwater coral reef monitoring surrounding the island of Moorea [1].

One of the most attractive features of WSNs is their ability to operate without human intervention. Through wireless communication, nodes can route data through the network to a base station, enabling them to be placed in obscure, inaccessible areas without access to a constant power supply. The sensor nodes used at Great Duck Island, for example, operate on batteries for approximately 8 months during breeding season without need for any physical human intervention. While batteries can serve as a temporary power supply, they must eventually be replaced, limiting the lifetime of the network. Energy harvesting is a promising solution to this limitation and is described in the following section.

1.2 Energy Harvesting

Energy harvesting involves the use of environmental energy sources, such as the sun, to sustain devices. Because an energy harvesting system obtains power from the environment, it can operate without human intervention for long periods of time. Successful use of energy harvesting in a remote area can be seen from the Mars Exploration Rovers [17]. The rovers carry two rechargeable lithium batteries, powered using solar arrays that provide, on average, 900 watt-hours per Martian day. As impressive as the numbers are, even the Mars Rovers have limitations. Due to the location of sunlight, the rovers are limited to exploring only certain regions of the planet.

A significant challenge in an energy harvesting system is the management and conservation of energy. In order for the system to operate for long periods of time without human intervention, the system must maintain energy neutrality, meaning that it can only consume as much energy as it can harvest. Typically, the goal of a sensing system is to complete a sequence of tasks within a designated period of time; however, to maintain energy neutrality, the system may need to either wait to complete all tasks until additional energy can be obtained or reduce the accuracy, or utility, of the tasks. Therefore, when given a sequence of tasks, the system must find a way to complete the sequence while respecting the system's unique constraints.

1.3 Structural Health Monitoring

One application that can benefit from energy-harvesting WSNs is structural health monitoring. Monitoring certain features of a structure over time and evaluating these features to determine the health of a structure is referred to as Structural Health Monitoring (SHM). The structure can be anything from a bridge to a building to an aircraft, and the feature can be any material or geometric property of the structure that signifies a defect or flaw in the system, such as temperature, strain, or cracks. Non-destructive testing, specifically, involves evaluating a structure by measuring the output in result to a known input without damaging the structure or inhibiting its operation.

Currently, most deployed SHM systems are wired, and thus take a significant amount of time and money to install. Evolving these networks to use wireless communication and design can help to reduce cost, time of installation, and maintenance requirements. For these reasons, SHM is a practical application for an energy-harvesting wireless sensor network. In contrast to current SHM sensor networks, this thesis discusses a wireless, energy-harvesting sensing system that uses local processing to provide long-lasting structural health monitoring.

1.4 Thesis Contribution

To provide energy and task management for energy harvesting wireless sensing systems, a system software controller is presented that adapts performance based on energy availability. Three application-independent algorithms are presented to balance performance and execution time subject to an energy constraint and then applied to external trigger state and steady state conditions. Steady state operation defines the periodic execution of a set of tasks on the system, adapting the execution rate to the energy profile. External trigger state operation, on the other hand, occurs when an external device makes a request to the system, specifying either a time limit or a desired utility.

The energy and task management methods are then applied to a structural health monitoring application. Structural health monitoring (SHM) consists of monitoring a structure over time and evaluating the health of the structure to determine if damage exists. SHiMmer [16] is a wireless platform that combines active sensing and localized processing with energy harvesting to provide long-lived structural health monitoring, using piezoelectric transducers (PZTs) to evaluate a portion of a structure to determine if damage exists. Unlike other sensor systems that periodically monitor a structure and route information to a base station, SHiMmer is designed to acquire data and process it locally before communicating with an external device, such as a remote controlled helicopter.

Results from this SHM application demonstrate the controller's ability to adapt at runtime and maintain energy neutrality. Steady state results show that the execution rate adapts to varying weather conditions. On average, the execution rate on a sunny day increases by 62% compared to the rate on cloudy days. External trigger state results show that processing significantly affects the efficiency of a structural health monitoring system; specifically, complex processing requires 17 times less execution time and 2.5 times less energy than transmitting raw data.

The remainder of this thesis describes the design and the contributions of the work in more detail. Chapter 2 summarizes relevant research relating to energy harvesting, energy management, and task assignment. Chapter 3 describes structural health monitoring techniques, technology, and related systems. Chapter 4 defines the energy and task management problem for a wireless sensing system and presents solutions for steady state and external trigger state operation. Chapter 5 then provides results obtained from evaluation of both modes of operation. Finally, chapter 6 concludes the thesis and discusses future work.

Chapter 2

Related Work

While abundant research has been conducted in the area of energy-constrained wireless sensor networks, the literature for energy-harvesting networks is scarce. This chapter provides an overview of related work in energy harvesting and wireless sensor networks. The first section discusses the work specific to energy harvesting: the challenges of energy harvesting systems [22], the differences between supercapacitors and rechargeable batteries [6], and the concept of energy neutrality [8] [9]. The following section addresses task scheduling in WSNs, from adjusting task execution rates [13] [15] to choosing among multiple task versions [25] [26] to mapping tasks among multiple systems [32]. Finally, two different methods for network data collection are outlined: routing [33] and data muling [29].

2.1 Energy Harvesting

Energy harvesting embedded systems present unique challenges and require creative solutions. Raghunathan and Chou [22] present an overview of design considerations for energy harvesting systems at both the system level and the software level, including energy storage, energy harvesting, power management policies, and energy-aware routing. The goal of an energy harvesting system is to maintain energy neutrality, a term used to describe the seemingly perpetual operation of a system - the amount of energy consumed should not exceed the amount of energy harvested. Thus, the power management policy of a single sensor node must adapt both the performance and energy consumption of the system to maintain energy neutrality.

A significant challenge and design decision for energy harvesting systems is the buffer used for energy storage. In the past, batteries and rechargeable batteries were used; however, batteries have a limited lifetime due to a limited number of recharge cycles. An alternative to a battery is a supercapacitor. Supercapacitors have a significantly longer lifetime than batteries but also typically have a reduced capacity. Thus, the needs of the system should dictate which storage buffer is used. In response to this challenge, Jiang et al. [6] propose a storage hierarchy, consisting of a supercapacitor and a rechargeable battery. The supercapacitor is the primary storage unit, while the battery is only used when the energy stored in the supercapacitor is insufficient. This hierarchy attempts to provide the benefits of both storage buffers, capitalizing on the longevity of the supercapacitor and the increased capacity of the battery. An energy management policy determines which buffer provides power to the system based on the energy buffer levels, the rate of energy harvesting, and the system state. Additionally, the algorithms estimate the highest allowed duty cycle for a given energy level to maintain energy neutrality.

The concept of energy neutrality is discussed in detail in [8], [9], and [5]. To guarantee energy neutral operation, Kansal *et al.* [9] present a harvesting theory that characterizes an energy source and energy consumer and determines the necessary storage capacity needed to ensure a desired performance level. Furthermore, in [8], Kansal *et al.* prove that a system can achieve energy neutrality using these characterizations with the assumption that task execution is modified using duty cycling alone. In [5], Hsu *et al.* present an algorithm for estimating the amount of future harvested energy and use this prediction to alter the node's duty cycle accordingly. While these works provide a means to adapt the duty cycle to account for changes in harvested energy, for many applications, duty cycling alone is not enough. Systems containing a variety of interdependent tasks require a more detailed task model. Additionally, this work assumes that the system power consumption is constant, which is an assumption not made in this thesis.

2.2 Task Scheduling in Energy Harvesting WSNs

The most relevant related work for this thesis is the scheduling and execution of tasks. The methods described in [6, 9, 8] adapt the duty cycle to account for the changes in harvested energy. Duty cycling is a energy-saving technique achieved by shutting down the device for a designated period of time[1]. A duty cycle of 1% indicates that 1% of the time the device will be active, and 99% of the time it will be shut down. In a simple system, duty cycling may suffice; however, many applications require more detailed task scheduling due to the variety of tasks and their inter-dependencies. Additionally, duty cycling prohibits the system from adapting to the stochastic nature of sunlight. For the applications targeted in this research, more sophisticated task scheduling is needed.

Moser *et al.* [14] present the Lazy Scheduling Algorithm (LSA) for an energy-driven scenario. The LSA gathers environmental energy and schedules tasks when either 1) the energy storage buffer is full, and thus, unused energy will be wasted, or 2) a task will miss its deadline if not scheduled. The goal of the LSA is to find the optimal start time for a task by considering task requirements, such as deadline and arrival time, and the node energy capacity. The LSA does not acknowledge dependencies between tasks and in many systems, tasks must occur in a designated order, as data dependencies exist.

Moser *et al.*, in [13], however, consider task dependencies by presenting an energy-harvesting system design to maximize node utility by adjusting task execution rates. An estimator predicts the amount of energy harvested in the future, while a controller adapts the system parameters to maximize node utility using critical regions defined offline with multi-parametric linear programming. The work provided in [15] extends [13] by reducing the number of critical regions using approximation. Both [13] and [15] demonstrate their solution using only two tasks and restrict task variation to altering only the execution rate, not providing a method for application-specific accuracy

[1]Another effective energy-saving technique not applied in this thesis is Dynamic Voltage Scaling [27].

measurements.

Rusu *et al.* [25] [26] present the only work to my knowledge that considers the accuracy level when scheduling tasks. In [25], each task has an associated reward, a measurement of the value of the task to the system. The goal of the system is to maximize the system value while meeting the global deadline and respecting the energy constraint. The authors present a heuristic that schedules tasks based on reward until a constraint is violated. The heuristics presented do not require that every task must be scheduled and do not address the dependencies between tasks. In [26], however, the authors extend the problem by assuming that each task can have multiple versions that each produce a different reward and also requiring that all (mandatory) tasks be scheduled. A similar heuristic is provided to maximize the system value while adhering to the energy and deadline requirements of the system. The algorithm requires that there exist a discrete number of versions per task (using four versions per task in their simulations) and assumes that a minimum amount of energy is available at the beginning of each “discharge” period.

Finally, Tian *et al.* [32] propose a task scheduling and mapping method for energy-constrained WSNs called EcoMapS. Tasks and their dependencies are modeled using directed acyclic graphs (DAG), where a hyper-DAG accounts for the communication constraints - a virtual node is used to represent the communication channel, and virtual tasks are used to represent communication between two nodes. The EcoMapS solution is composed of two stages: the initialization phase and the quick recovery phase. Initialization consists of two parts. First, tasks are arranged into a list/queue such that dependencies are maintained and critical paths occur first. Second, tasks are dequeued and assigned to sensors with minimum execution start time, which is done iteratively with different numbers of sensors. The schedule with the minimum length that meets the energy budget is chosen. Quick recovery occurs if a sensor node fails, and all tasks assigned to the failed node are given to the node with the most idle time.

Regardless of the energy and task management used on systems, a WSN must have a method for getting data from nodes to a base station. There are two primary ways of accomplishing this: the first is through multi-hop routing to a base station, and the second is through the use of a technique called data muling.

2.3 Data Collection

The most common form of data collection is multi-hop routing. Voigt *et al.* [33] present two solar-aware routing protocols that are related to directed diffusion but where route selection prefers solar-powered nodes over battery-powered nodes. The first protocol modifies directed diffusion gradients to include information about energy supply (solar vs. battery) and balances solar-powered nodes with nodes on the shortest path. The second extends directed diffusion by adding extra header fields and a mechanism to prevent loops from occurring.

Voigt *et al.* [23] also propose a solar-aware extension to LEACH, the popular low-energy adaptive clustering hierarchy. In the modified LEACH, solar-powered nodes are preferred during cluster head selection, and cluster heads can be changed during a

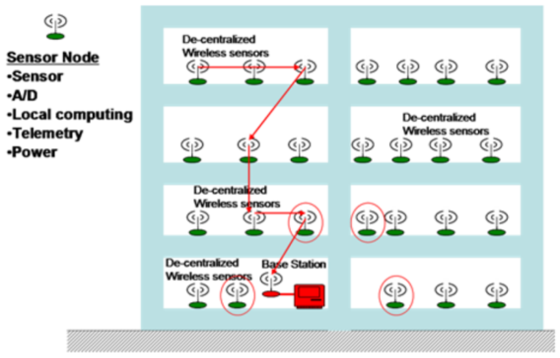

Sensor Node
•Sensor
•A/D
•Local computing
•Telemetry
•Power
De-centralized Wireless sensors
De-centralized Wireless sensors
De-centralized Wireless sensors
De-centralized Wireless sensors
Base Station

Figure 2.1: Data Collection: Routing [30]

round depending on solar changes. To extend the centralized LEACH protocol, each node sends its solar status to the base node in addition to its remaining energy and position. Base nodes then select cluster heads using a heuristic. To extend the distributed LEACH protocol, solar-powered nodes have a higher probability of becoming a cluster head and thus, will be a cluster head more often than typical nodes. The handover mechanism allows a current cluster head to relinquish its responsibilities to another node if the energy profile changes.

In contrast to routing, a mobile external agent can also be used to collect data from nodes in a sensor field. Sugihara and Gupta [29] provide an overview of the concept of data muling and the challenge of optimizing the trade off between energy consumption and data delivery latency. Eliminating multi-hop routing in a network reduces the energy consumption but requires intelligent and efficient data muling. They show that the path selection problem for data muling is NP-hard, but present a near optimal algorithm.

Additionally, Taylor *et al.* [30] use an unmanned vehicle to collect data from

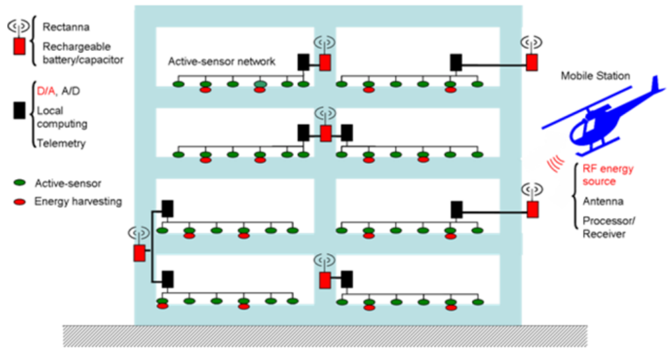


Figure 2.2: Data Collection: Data Muling [30]

sensor nodes on a bridge. The authors use figures 2.1 and 2.2 to contrast the differences between routing and data muling for the purpose of data collection in their application. They describe the implementation and practical experience from using a mobile agent to not only collect data but to also power the sensor nodes using radio-frequency transmission.

Data muling has a promising future but requires solutions unique from the traditional routing paradigm. The work in [29] and [30] each address a challenge of data muling, specifically, choosing the optimal path for data collection and remotely powering sensor nodes using radio transmission. While this thesis does not address these two challenges, it focuses on the interaction between the data mule and the sensor node characterized by external trigger state operation. Energy management is presented for both types of data collection: data muling using external trigger state operation and routing using steady state operation.

2.4 Summary

In summary, past research has focused on many aspects of energy-harvesting systems such as guaranteeing energy neutrality [8] [9], adjusting system duty cycle [13] [15], choosing among multiple task versions depending on energy constraints [25] [26], and collecting data in an energy efficient way [33] [13] [15].

While significant research has been explored in the area of energy management and task scheduling in energy-harvesting systems, only a few specifically address the trade off between performance and energy. In contrast to previous work, this thesis specifically addresses the trade off between performance and system constraints in energy-harvesting systems and provides a flexible and adaptable solution for maintaining energy neutrality.

Chapter 3

Structural Health Monitoring

The process of monitoring a structure for the purpose of damage identification is known as structural health monitoring (SHM). SHM requires knowledge of the undamaged state of the structure as a means of comparison, as well as continual comparison of periodic measurements. SHM can be separated into two basic categories: rapid event assessment and periodic lifetime monitoring. Rapid event assessment addresses the need to obtain data from a structure immediately following a significant event, such as an earthquake. Periodic lifetime monitoring seeks to identify damage that accumulates over a long period of time.

This chapter describes the fundamentals of structural health monitoring as they relate to the work in this thesis. First, section 3.1 provides an overview of the Statistical Pattern Recognition Paradigm used for SHM. Then, section 3.2 discusses a specific type of SHM - non-destructive evaluation using piezoelectric transducers (PZTs), followed by an example damage identification process in section 3.3. Section 3.4 describes two current SHM systems, while section 3.5 provides an overview of the wireless SHM platform that is used in this thesis.

3.1 Statistical Pattern Recognition Paradigm

The SHM process can be described using the four step, statistical pattern recognition paradigm as presented by Farrar and Worden [2]. The first step is operational evaluation, the process of observing and testing a system to determine the part(s) of the system that should be monitored and the best method for monitoring them. As with the other steps, operational evaluation is application specific as each system has its own characteristics that can be exploited for accurate damage identification. The second step is a combination of data acquisition, normalization, and cleansing. Data is first collected from the system using sensors, followed by normalization and cleansing, the processes of distinguishing between damage and environmental factors and choosing which data to disregard. The next step in the paradigm is feature selection and information condensation. A feature is a specific property of a system that can be used to identify damage. Features can be extracted from the data, which, inherently, reduces the amount of data. Finally, statistical model development for feature discrimination involves the evaluation of features to quantify the damage. Statistical modeling refers to three types of algorithms: group classification, regression analysis, and outlier detection. Each of these types of algorithms attempts to classify damage by one or more of the following properties:

1. Existence: Does damage exist?
2. Location: Where is the damage?
3. Type: What kind of damage is it?
4. Extent: How severe is the damage?
5. Prognosis: How much useful life remains?

The statistical pattern recognition paradigm is a general framework for structural health monitoring and can be applied to various types of sensors and methods of sensing. Typically, sensors are thought of as passive - essentially "listening" to something and recording the results. Sensors can also be active, however, by stimulating the environment and recording the response to the stimulus. Piezoelectric transducers, a type of a material that perform non-destructive evaluation by actively sensing a structure, will be described in more detail in the next section.

3.2 Piezoelectric Transducers

In contrast to passive sensing, damage monitoring can be accomplished using nondestructive evaluation, such as the use of smart materials. A promising method for SHM is the integration of smart materials, such as Lead-Zirconate-Titanate Piezoelectric Transducers (PZTs) shown in figures 3.1 and 3.2[1]. A PZT can be used to generate as well as sense signals and thus is capable of performing the role of both an actuator and a sensor. For example, a PZT can use high frequency vibrations to generate a lamb wave,

[1]Several pictures were taken at the Alamosa Canyon Bridge in New Mexico while LANL structural engineers were performing tests.

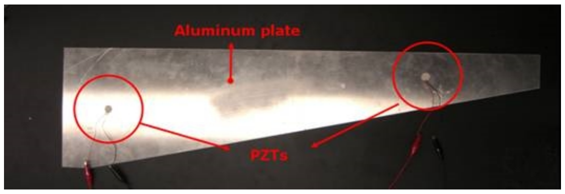


Figure 3.1: Two PZTs on an Aluminum Plate

Figure 3.2: PZTs on a Bridge

a type of elastic perturbation that can propagate in and reveal certain characteristics about a solid. The speed of a lamb wave is dependent on the frequency of the solid and can be generated by a smart material, such as a PZT. For the SHM application in this thesis, lamb waves are generated by a PZT at a frequency of 1 MHz frequency with a peak-to-peak maximum amplitude of 15 volts and sampled by another PZT at 10 MHz. Because the electrical impedance of a PZT is directly related to the structure's mechanical impedance, this impedance-based method can use lamb waves to monitor the structure's mechanical impedance in order to detect and locate damage in a structure.

SHM requires knowledge of the undamaged state of the structure as a means of comparison to determine if damage may exist. Operational evaluation using PZTs involves testing a structure by generating a lamb wave and sensing the resulting wave to evaluate the impedance of the structure. Gathering data from a healthy, undamaged state of the structure can be evaluated against data acquired during operation. The features of both the undamaged and operational state can then be compared to see if damage exists. The following section describes in detail an example damage identification process using active sensing with PZTs.

3.3 Damage Identification

Structural damage can be defined as any geometric or material change introduced into a system that negatively impacts its current or future performance [2]. Damage detection is the process of identifying this change and can either be global or local. Global damage detection is accomplished by placing a small number of sensors across a structure to identify global properties, such as monitoring the effect of temperature on a bridge's length. Local damage detection, on the other hand, involves the placement of many sensors in a small area, focusing on specific parts of a structure such as a bolt or plate. The work in this thesis is based on local damage identification [10].

The first step in damage identification [3] is the collection of data, referred to as actuation and acquisition. The process of actuation and acquisition uses PZTs to actuate a wave through the structure and then sense the wave and its reflections. A path is defined as the pairing of one PZT as the actuator with another PZT as the sensor. Figure 3.3 shows sixteen PZTs, resulting in up to 120 paths. Greater reliability can

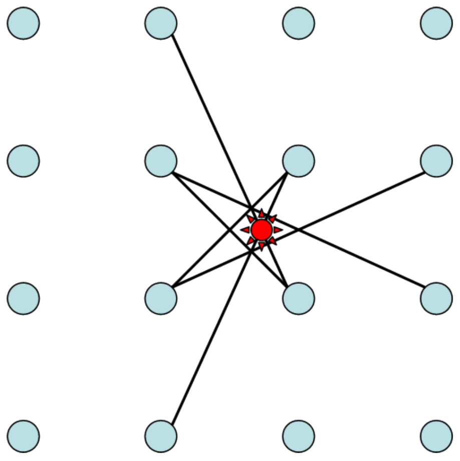


Figure 3.3: Damage Identification Using PZTs [16]

Figure 3.4: Two Structural Engineers Evaluating Data

be obtained by interrogating one path several times and then averaging the multiple iterations. After a path is evaluated, the data must be processed to determine if damage exists. Figure 3.4[1] shows two structural engineers evaluating data using sophisticated algorithms implemented in MatLab.

The first part of processing is to remove noise from the received signal by filtering. Filtering noise from the signal can be done using either a bandpass or a matching filter. The signal is converted to the frequency domain using the FFT, multiplied by the filter, and converted back to the time domain using the inverse FFT [28][2]. While FFT complexity has improved over time, a single FFT still requires approximately $4\text{N}\log_2\text{N}$ floating point operations [7]. Thus, if N is 10,000 points, one FFT would require 531,510 floating point operations[3].

After a signal is filtered, it is divided into blocks such that each block represents a different route from the actuator to the sensor. The first block of the signal indicates the direct route from one PZT to the other, while each subsequent block is a reflection from the area surrounding the two PZTs. To perform feature extraction, each block is compared to a baseline signal, where the baseline is the response of the system to a healthy, non-damaged structure (obtained *a priori* to deployment.) The root mean square deviation between the two signals is called the feature and represents the difference between the healthy signal and the current signal [19].

Using these extracted features from many paths, damage can be identified by

[2]Filtering can also be done in the time domain using a moving window averaging procedure.

[3]Fixed point arithmetic could also be used to increase efficiency, depending on the architecture of the DSP.

correlating different paths. Within the sensing area of the structure, points of interests are identified that may be susceptible to damage. For each point of interest, several steps are taken. Before deployment, the distance between the point of interest and each PZT path is calculated, indicating the location in each path corresponding to the point of interest. Using this information during deployment, the system can find a weighted average of the previously computed features to determine the feature of each path corresponding to the point of interest, referred to as that point's feature vector. Comparing the sum of the features from all paths to a threshold value can indicate if damage exists at the point of interest.

3.4 Current Systems

There have been several wireless sensor networks designed specifically for the purpose of SHM. The range of SHM platforms varies depending on the sensing method. PZTs, for example, can be used in an active or passive manner. PZTs are used for active sensing when actuation and sensing occur(as described in section 3.2), but can also be used for passive sensing through impedance monitoring [30]. Accelerometers also provide a method for SHM sensing by measuring structural vibration [34] [18]. Two current SHM systems include the WID 3.0 network [30] and Wisden [34].

Taylor *et al.* [30] present a mobile-agent based SHM wireless sensor network. The WID 3.0 is a sensor node that passively monitors the impedance of a several piezo-electric transducers. The WID 3.0 is combined with a data acquisition system to assist in data acquisition and storage. The WID 3.0 can be remotely powered using a rectenna (rectifying antenna) that converts transmitted microwave energy into DC power. In testing, a bridge is outfitted with many of these sensor nodes. A remote-controlled robot, shown in figure 3.5, travels along the bridge, stops at each node, powers the node, and then receives acquired data. The data is then transmitted to a base station to be processed using SHM analyzing software. Additionally, small local area networks (LANs) are designed such that one sensor node acts as a coordinator while other sensor nodes act as end devices. The robot then only communicates with the coordinator to retrieve data. However, in this LAN scenario, all nodes are powered using batteries without the addition of energy harvesting.

Xu *et al.* [34] describe the design and deployment of an SHM network paradigm called Wisden. Wisden nodes, specifically Mica-2 motes, use accelerometers to collect structural data and transmit it to a base station. The nodes self-configure to form a tree topology such that the root of the tree is the base station. Important design characteristics of Wisden include: reliable data transfer, topology self-configuration, hop-by-hop reliability, end-to-end reliability, data (time) synchronization, and data compression. The primary disadvantage of this design is that it is only a data acquisition system and does not perform any local processing. Local processing, while consuming significant power, reduces the amount of data to be transmitted and also provides the means to interrogate a structure based on past data.

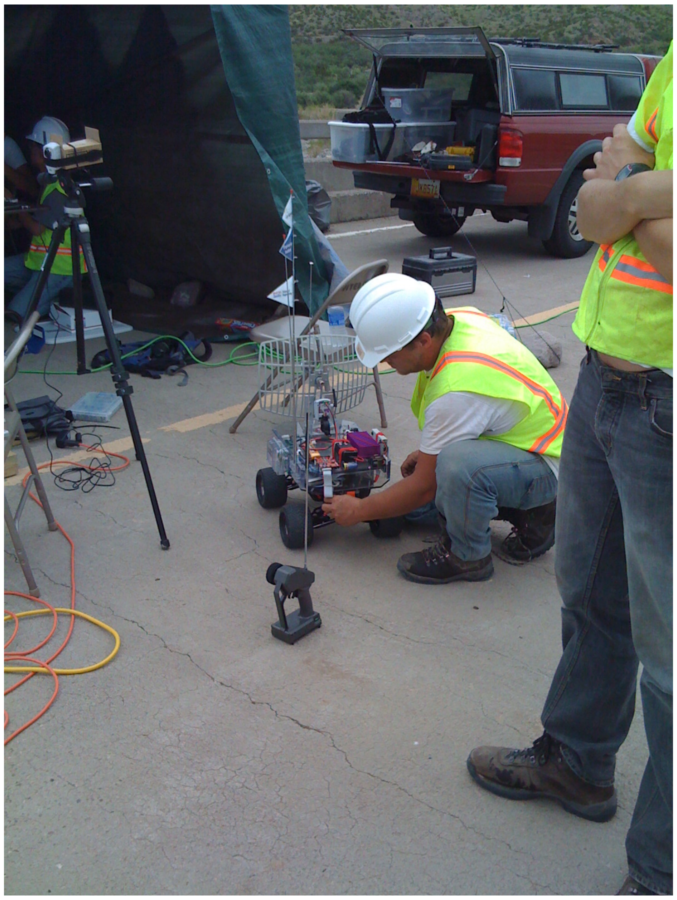


Figure 3.5: Remote Controlled Robot Used for Collecting SHM Data

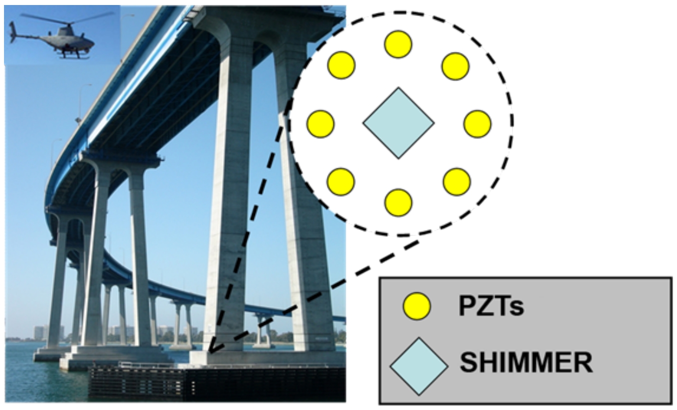


Figure 3.6: SHiMmer Design [16]

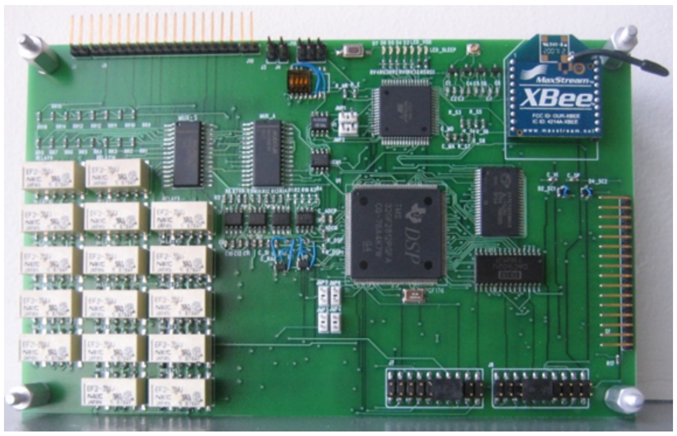


Figure 3.7: SHiMmer Board: Topview

3.5 SHiMmer

Unlike the previously described systems, SHiMmer[4] [16], a wireless SHM platform, is designed to actively sense a structure using PZTs and communicate with an external device, such as a remote controlled helicopter shown in figure 3.6. SHiMmer needs to both actively sense the structure and also process the readings using the sophis-

[4]The SHiMmer platform discussed in this thesis has no connection to the SHIMMER health monitoring sensors [20].

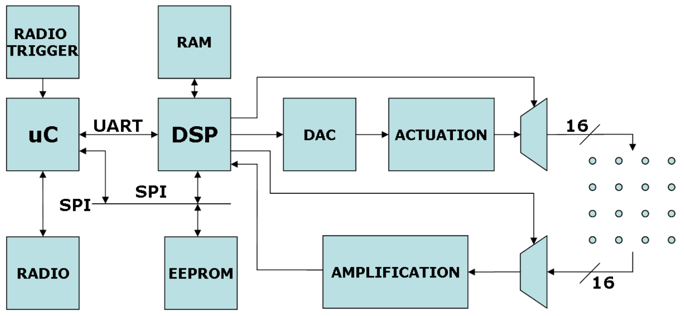


Figure 3.8: SHiMmer Board: Block Diagram [16]

ticated SHM damage-detection algorithms described in section 3.3. SHiMmer is part of a joint project between the Los Alamos National Laboratory (LANL) Engineering Institute and the University of California, San Diego. The goal of this project is to develop a wireless sensor network to be deployed over civil infrastructure and mechanical systems for SHM purposes.

The SHiMmer platform can be categorized into three parts: the digital components, the analog components, and the energy harvesting components. The parts relating to energy harvesting will be further described in chapter 5, as SHiMmer can operate with any power source. The digital and analog components form the hardware of the platform. As seen in figures 3.7 and 3.8, SHiMmer is composed of several key hardware components described in table 3.1. The Atmel 128L microcontroller controls the entire operation of the platform, communicating with the DSP and the radio. To enhance functionality and assist in control, the microcontroller runs a real time operating system (freeRTOS [4]) that provides OS primitives such as mutual exclusion, time synchronization, and task abstraction. The DSP, on the other hand, executes the application-specific

Table 3.1: Description of SHiMmer Hardware Components

Component	**Purpose**	**Characteristics**
Atmel 128L Microcontroller	Platform Control	High-performance, Low-power, 8-bit, RISC architecture
TI TMS320C2811 Digital Signal Processor	Actuation, Sensing, and Data Processing	32 bit, 135 MIPs
Maxstreams XBee OEM RF module	Wireless Communication	IEEE 802.15.4 Zigbee standard protocol, low power, low data rate
Cypress 8Mbits SRAM	Data Storage	8Mbit, Low power, high speed
Sandisk 4Gbits Serial Flash Memory	DSP Code Storage	8 GB, low power, small form factor
DAC902 Digital to Analog Converter	Conversion of Actuation Signal	12-Bit, 165MSPS
LT1218 Current to Voltage Converter	Conversion of Actuation Signal	low input offset voltage
LT1210 and THS4082 Amplifiers	Sensing Signal Amplification	1.1A, 35MHz and 175-MHz low-power, high-speed
UART and SPI	Platform Communication	UART Baud rate = 9600, SPI frequency = 1-30 MHz
Radio Triggering Circuit	External Triggering	2.4GHz, requires 2.0 V

tasks, such as data processing and wave generation. Specifically, SHiMmer is equipped with a multiplexer and demultiplexer that select a couple of the 16 PZTs. Using a DAC, waves can be generated with a frequency up to 1 MHz and a peak-to-peak amplitude of 15 volts. During acquisition, waves can be sampled at a frequency of 10 mega-samples per second.

SHiMmer communicates with external devices using Maxstreams XBee OEM RF Module [12]. The XBee module uses the IEEE 802.15.4 Zigbee standard protocol and operates in the ISM 2.4 GHz frequency band. The radio can transmit to distances up to 100 meters, assuming line of sight. The diagrams of both the XBee and XBee PRO modules are show in Figure 3.9. On The SHiMmer platform, however, only a subset of the available pins are connected, specifically: PIN 1 (supply voltage), PIN 2 (UART data out), PIN 3 (UART data in), PIN 9 (sleep control line), and PIN 10 (ground). The XBee module interfaces with a host device through a serial port and, thus, can communicate with the microcontroller using SHiMmer UARTs interface. Figure 3.10 shows the relationship between SHiMmer and the XBee module. Data is sent from the SHiMmer microcontroller over the DI pin and back through the DO pin. Because the Clear-to-Send and Request-to-Send pins are not connected on the SHiMmer platform, the DI06 and DI07 configurations were disabled, and, thus, hardware flow control cannot be used. Instead, software flow control was implemented in order to avoid buffer overflow.

XBee can operate in two modes: transparent and API. In transparent operation, the XBee module acts as a serial line replacement, transmitting only the UART data received from the microcontroller and passing the exact received data back to the UART;

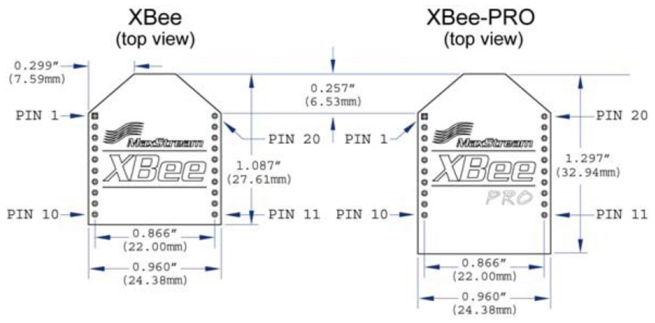


Figure 3.9: Maxstream's OEM RF Modules: XBee and XBee PRO

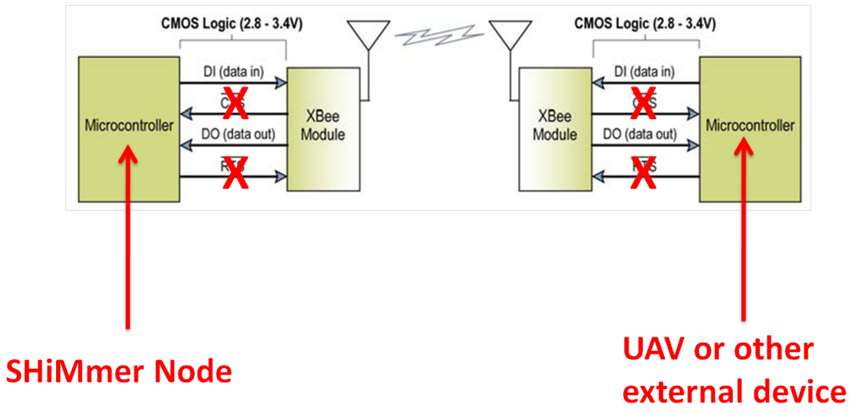


Figure 3.10: XBee System Flow Diagram

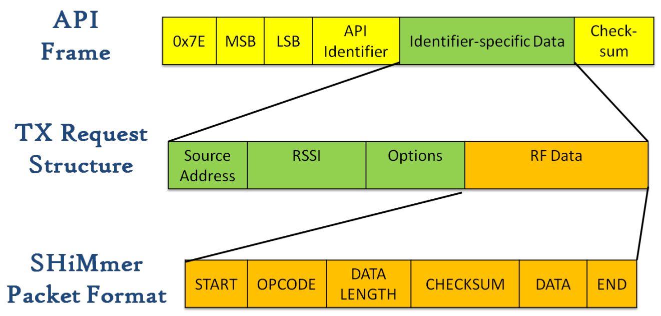
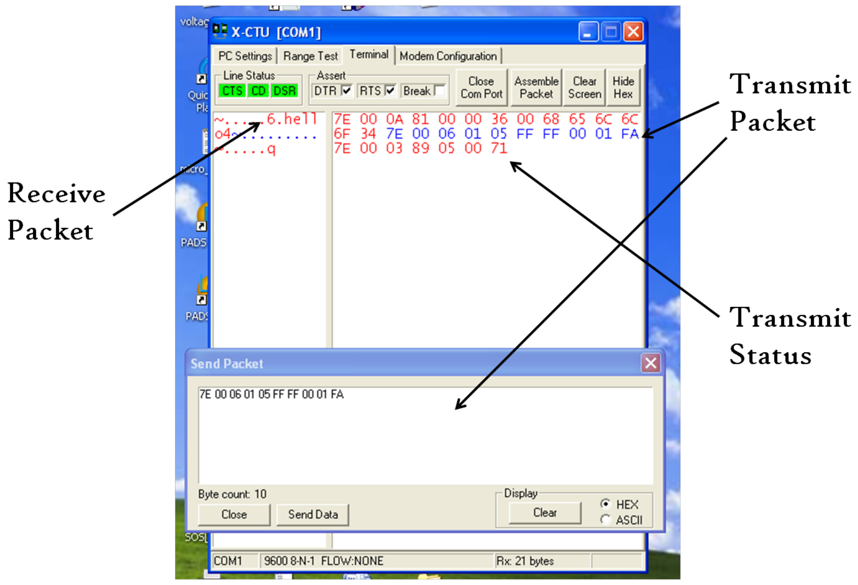


Figure 3.11: XBee API: Transmit Frame Structure

no additional information is added. Due to the fact that hardware flow control cannot be used, it is necessary to use software techniques when interacting with the XBee module, such as ensuring that size of the data sent to the module is smaller than the size of the DI buffer. In addition, there is no way in transparent operation to check if a packet has been sent successfully or whether another attempt should be made. In API mode, however, the XBee module packages data in a structured interface and provides the desired functionality for SHiMmer. Data sent over the DI pin must be contained in the specified frame, defining what type of event is to be performed (transmit, AT command, etc.)

Using the API operations enables SHiMmer to receive the status of a transmitted packet, which is essential to determining if data is sent successfully. Whenever data is to be transmitted, the Transmit Request frame format is used, as shown in Figure 3.11. Finally, the RF data contains the precisely defined SHiMmer packet format, which is interpreted at the receiving end [16]. In the same manner, when data is received from the XBee module, the interrupt service routine processes the received packet as



DTR ✓ RTS ✓ Break ☐

```
~.....6.hell   7E 00 0A 81 00 00 36 00 68 65 6C 6C
o4~.........   6F 34 7E 00 06 01 05 FF FF 00 01 FA
~.....q        7E 00 03 89 05 00 71
```

```
7E 00 06 01 05 FF FF 00 01 FA
```

Byte count: 10

Display: [✓ HEX] ASCII

COM1 | 9600 8-N-1 FLOW:NONE | Rx: 21 bytes

Figure 3.12: XCTU: Interaction with SHiMmer

either a transmit status packet or a received packet, which can be determined by the API identifier. Figure 3.12 shows the XCTU console running on a desktop that is used to test the SHiMmer platform. XCTU receives a packet from SHiMmer, sends a packet back to SHiMmer, and subsequently, receives a transmit status packet that indicates that the packet was sent successfully.

3.6 Summary

In this chapter, the fundamentals of structural health monitoring were described as they relate to the work in this thesis. The Statistical Pattern Recognition Paradigm is a four-step paradigm for SHM that involves operational evaluation, data acquisition, feature extraction, and statistical modeling. Piezoelectric Transducers (PZTs) are a type of smart material that can be used for SHM active, non-destructive evaluation and applied to the paradigm using a process of damage identification. Two relevant SHM systems are described - one uses PZTs for passive sensing and the other uses accelerometers. Unlike these systems, however, SHiMmer uses PZTs to actively sense a structure, process the results to determine if damage exists, and transmit to a mobile device. The following chapter presents energy and task management algorithms that maintain energy neutrality while executing the necessary tasks, such as actuation and processing.

Chapter 4

System Energy and Task Management

In contrast to battery powered systems, the goal of an energy harvesting sensing system is to maximize performance while maintaining energy neutrality. Because energy harvested from the environment depends on the environment (such as solar power depending on the sun), performance must be adapted based on variations in the environmental energy source. For example, on a sunny day, a system should sense a large amount of data and perform extensive processing on the data, while on a cloudy day, less data can be sensed and processing should be reduced. Additionally, the system must adapt performance according to external requests and requirements of the network. For instance, if a system is notified of an emergency situation, it should temporarily sacrifice energy neutrality to fulfill the requirements needed by the emergency.

The system can operate in one of two modes: steady state and external trigger state. Steady state operation is typical of sensor nodes; data is gathered, processed, and transmitted according to a predefined schedule over a long period of time. The goal of steady state operation is to periodically execute a set of tasks while maintaining a desired level of energy and either send this data to a base station or store the results for a later time. In contrast, when external trigger state operation occurs, an external device is present and issues a request to the system that imposes a constraint. The goal of the system in this state is to meet the constraint and inform the external device of the expected performance of the request. The external device can be any device that communicates with the system, such as a base station, a mobile agent, or another sensor node.

This chapter proposes three application-independent algorithms for energy and task management in externally-triggered, energy harvesting sensing systems. Specific to the state of operation, a software controller uses these algorithms to determine system performance, execution time, and energy consumption in order to maintain energy neutrality and satisfy performance constraints. This chapter is structured as follows. Section 4.1 describes the system model, specifically the task, energy, and prediction models. Section 4.2 details the three application-independent algorithms. Section 4.3 describes steady state operation, while section 4.4 describes external triggering state operation. The chapter concludes in section 4.5.

4.1 System Description

The purpose of the system is to execute a set of tasks while maintaining energy neutrality. The system model, shown in Figure 4.1, illustrates the components that com-

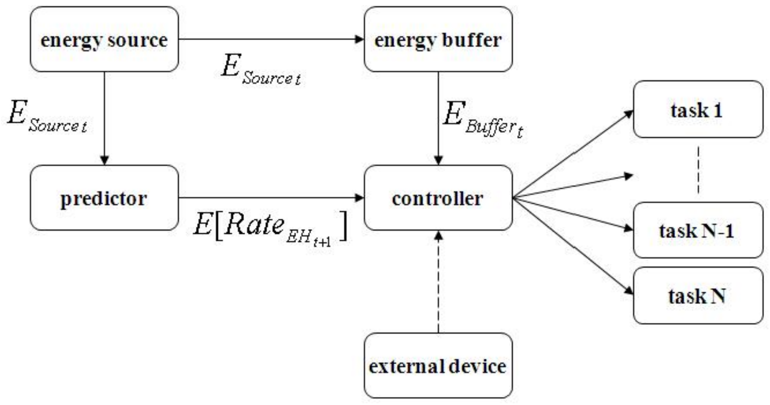


Figure 4.1: System Model

prise the sensor node. The system is assumed to be powered by an energy harvesting device, denoted as the energy source, and to store the energy in either a rechargeable battery or a super capacitor, labeled as the energy buffer. The two primary software components are the predictor and the controller. The predictor, further described in section 4.1.3, uses past energy harvesting information to estimate the future rate of energy harvesting. The controller then uses this predicted information, along with the current amount of energy in the buffer, E_{buffer}, and the system requirements to determine the performance, energy consumption, and execution time of a set of tasks. The following subsections describe in detail the three models used in the system: the task model, the energy model, and the prediction model.

4.1.1 Task Model

The relationship among the tasks in a system is modeled as a Directed Acyclic Graph (DAG). In the DAG G = (**T**,**E**), each vertex represents a task $\tau \in \mathbf{T}$, which is an

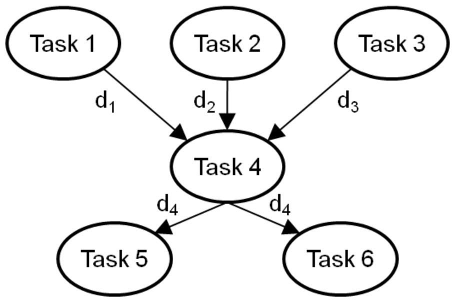


Figure 4.2: Example Task Graph

action or combination of actions that the system performs. Each edge $e_{ij} \in \mathbf{E}$ represents a dependency between tasks such that τ_j cannot begin execution until τ_i is complete. It is assumed that the Worst Case Execution Time (WCET) for each task is known ahead of time.

DAGs have several useful properties for the model. First, in a DAG, there exists at least one vertex with no incoming edges. A task with no incoming edges is referred to as an entry task. Conversely, a DAG contains at least one vertex with no outgoing edges. This vertex is referred to as an exit task. Tasks are also identified by their connecting edges. Suppose that there is an edge from task τ_i to task τ_j. Task τ_i is then called the predecessor of task τ_j, while task τ_j is the successor of task τ_i. A task can have multiple predecessors and successors, which indicates the dependencies that exist among tasks in the system. Figure 4.2 illustrates a task DAG. Tasks τ_1-τ_3 are entry tasks, as they have no incoming edges. Upon execution of their predecessors, task τ_4 can begin execution. After task τ_4 completes, the exit tasks, τ_5 and τ_6 can execute.

In the system, each task τ is defined by its relationship with the other tasks in the system, its priority, and its application specific execution characteristics, each of which will be described in further detail in this section. A task τ_i can be represented as a seven-tuple $\{p_{\tau_i}, U_{\tau_i}, t_{\tau_i}, P_{\tau_i}, s_{\tau_i}, f_{\tau_i}, d_{\tau_i}\}$ where p_{τ_i} is the priority, U_{τ_i} is the utility, t_{τ_i} is the WCET, P_{τ_i} is the power, s_{τ_i} is the start time, f_{τ_i} is the finishing time, and d_{τ_i} is the amount of data produced.

Upon the completion of any task τ_i, an amount of data, d_{τ_i}, is produced. The produced data is a function of the utility, U_τ, although the exact relationship is defined by the application. This data is then passed on to successors in the DAG. For example, as shown in figure 4.2, task τ_4 sends data d_{τ_4} to both of its successors, tasks τ_5 and τ_6, upon completion.

In this model, each task τ is executed with a level of utility, U_τ, such that $0 \leq U_\tau \leq 1$. In this work, the terms utility and accuracy will be used interchangeably, as in most applications, they have similar implications. A utility of 1 represents the highest level of utility a task can perform, while a utility of 0.1 indicates little to no accuracy, and a utility of 0 indicates that the task should not be executed at all.

The relative importance or priority of each task τ in the system is denoted by the task's priority, p_τ, as shown in the task graph in Figure 4.2. The priority is used to determine the utility of each instance of a task in relation to other tasks in the system such that equation 4.1 holds. The task with the highest priority will be executed with the highest utility and so forth.

$$\mathrm{U}_{\tau_\mathrm{i}} = \frac{\mathrm{p}_{\tau_\mathrm{i}}}{\sum_{\mathrm{j}=1}^{\mathrm{N_{tasks}}} \mathrm{p}_{\tau_\mathrm{j}}} * \sum_{\mathrm{j}=1}^{\mathrm{N_{tasks}}} \mathrm{U}_{\tau_\mathrm{j}} \tag{4.1}$$

For example, consider a system designed to provide a video interface for a teleconference; the two tasks in this system might be: task τ_1, display local video, and τ_2, display remote video. In this system, it is desirable to display the remote video at a higher resolution than the local video, as it is of more importance to see the other person than oneself. Thus, the priority for τ_1, display local video, should be smaller than that of τ_2, display remote video. If $\mathrm{p}_{\tau_1} = 1$ and $\mathrm{p}_{\tau_2} = 2$, the utility of task τ_2 should be twice that of task τ_1.

Because the relative importance of each task may change depending on the system's current conditions, the task's priority is expected to change in a system over time. The priorities of tasks are assumed to only change before or after an iteration of the task graph.

The relationship between the utility of a task and its specific execution characteristics must be defined by the application. The expected execution times for each level of utility for each task can either be experimentally measured or calculated using a functional relationship such that t_τ equals the expected execution time of τ. For the general solution, this application specific relationship is referred to as $f(\mathrm{U}_\tau)$, shown in equation 4.2. The energy consumption e_τ of a task τ can then be determined using the execution time t_τ and the task power P_τ. To simplify the model, the power P_τ is assumed to be constant for all utility values of a task; however, this may not be the case for all applications, in which case the power can be determined in the same manner as the execution time using an application specific function.

$$\mathrm{t}_{\tau} = \mathrm{f}\left(\mathrm{U}_{\tau}\right) \tag{4.2}$$

$$\mathrm{e}_{\tau} = \mathrm{t}_{\tau} * \mathrm{P}_{\tau} \tag{4.3}$$

It is expected that as the utility U_{τ} of a task increases, the energy and time of task τ will also increase; however, the actual implementation of the above function depends entirely on the application. One possibility for the relationship between the utility and time of a task is a function. The utility of a temperature sensor reading, for example, may increase logarithmically with its execution time, as saturation occurs. Another possible relationship could be the use of a lookup table representing only several discrete utility values. For instance, video resolution may only have two levels of utility: low (U = .5) or high (U = 1). Chapter 5 defines these relationships for SHiMmer, a Structural Health Monitoring system.

4.1.2 Energy Model

The energy model for this work has two parts: the energy harvesting source and the energy storage buffer. A requirement for the system is that the status and level of both of these components must be closely monitored as they are the lifeline of the device. Decisions made by the controller must adhere to the energy storage constraints and can exploit the knowledge of the energy source to improve performance.

Energy Harvesting

The energy harvested from the energy source at time t, $\mathrm{E}_{\mathrm{harvest}_t}$, is defined as the amount of energy harvested between time t-1 and time t such that the time between time t-1 and time t is the period T. The total energy harvested at time t is a result of the energy at the source, $\mathrm{E}_{\mathrm{source}_t}$, the efficiency of conversion, $\eta_{\mathrm{conversion}}$, and the energy lost due to leakage in the period T, $\mathrm{E}_{\mathrm{leakage}}$. The instantaneous rate of energy harvesting, $\mathrm{Rate}_{\mathrm{EH}} = \frac{\mathrm{dE}}{\mathrm{dT}}$, can then be defined as the energy harvested at time t divided by the time period T.

$$\mathrm{E}_{\mathrm{harvest}_t} = \eta_{\mathrm{conversion}} * \mathrm{E}_{\mathrm{source}_t} - \mathrm{E}_{\mathrm{leakage}_t} \tag{4.4}$$

Energy Storage

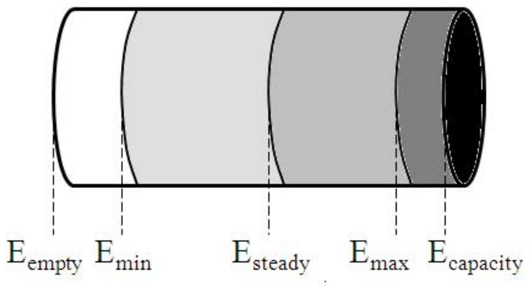


Figure 4.3: Energy Buffer Constraints

Energy thresholds, shown in Figure 4.3, are set based on the needs of the application and the characteristics of the buffer and source. The energy storage buffer has capacity, E_{capacity}, and a maximum allowed amount, E_{max}, that may or may not be equal to E_{capacity}. For example, Recas *et al.* [21] show that the recharge rate of a supercapacitor is significantly lower when the buffer is nearly full and thus, it is more beneficial to use the extra energy to execute tasks than to completely fill the supercapacitor. During steady state operation, $E_{\text{buffer}} \geq E_{\text{steady}}$, and during an external request, $E_{\text{buffer}} \geq E_{\text{min}}$. The available energy at any time t is denoted as $E_{\text{available}}$. The energy in the buffer at time t, $E_{\text{buffer}_\text{t}}$, is the sum of the buffer energy at time t-1, $E_{\text{buffer}_\text{t-1}}$ and the energy harvested at time t, $E_{\text{harvest}_\text{t}}$ less the energy consumed, $E_{\text{consumed}_\text{t-1}}$.

$$E_{\text{buffer}_\text{t}} = E_{\text{buffer}_\text{t-1}} + E_{\text{harvest}_\text{t}} - E_{\text{consumed}_\text{t-1}} \tag{4.5}$$

4.1.3 Prediction Model

At any point in time, there may not be enough energy available to execute all the desired tasks immediately. The controller can thus estimate the additional time needed to harvest energy, t_{wait}, shown in equation 4.6, using the expected rate of energy harvesting, as provided by the predictor, described in detail below. E_{required} is the sum of the energy required for each task in the system at the desired utility level.

$$t_{\text{wait}} = \frac{E_{\text{required}_\text{t}} - E_{\text{available}_\text{t}}}{E[\text{Rate}_{\text{EH}_\text{t+1}}]} \tag{4.6}$$

$$E_{\text{required}_\text{t}} = \sum_{\text{j}=1}^{\text{N}_{\text{tasks}}} e_{\tau_\text{j}} \tag{4.7}$$

To estimate the amount of time needed to harvest the additional energy, the system must predict how much energy can be harvested in the future. For this, a method based on work by Recas *et al.* is used [21]. The predictor uses past energy harvesting information from previous days to estimate the rate of energy harvesting in the next period, $E[\text{Rate}_{\text{EH}_{\text{t+1}}}]$. To do this, the predictor stores data gathered from past days in a matrix of size DxN, where D is the number of past stored days, and N is the number of energy values stored per day. At any time t, the predictor provides a predicted rate of energy harvesting for time t+1 that indicates the number of expected joules per second that will be harvested in the next time period.

The basic equation shown in figure 4.8 uses an Exponentially Weighted Moving-Average combined with a GAP factor to account for differences specific to solar energy harvesting. α is the weight given to the previous time period in the current day, while (1-α) is the weight given to data from the previous D days. The GAP factor accounts for the variations in weather conditions among days by comparing the average solar energy in a given time period. If GAP is greater than 1, then the energy harvesting rate of the current day is greater than the average; conversely, if GAP is less than 1, then the rate of the current day is smaller than the average. GAP can be used to estimate if the day is sunny, cloudy, or somewhere in between. To compute the GAP factor, a vector V is defined, where each element in V equals the current rate divided by the average rate from the past D days. The vector V is then multiplied by a weighting vector P, which weights the current time of day higher than previous times during the day, and is finally normalized by dividing by the sum of P. The results from [21] show that this predictor

Table 4.1: Variables Used in the ILPs

p_{τ_i}:	priority for task τ_i	t_{wait}:	Time delay for energy harvesting
U_{τ_i}:	Utility for task τ_i	T_{limit}:	Execution time limit for all tasks
s_{τ_i}:	Start time of task τ_i	t_{total}:	Total time needed to complete all tasks
t_{τ_i}:	WCET of task τ_i	$E[\mathrm{Rate}_{EH}]$:	Expected rate of energy harvesting (J/sec)
P_{τ_i}:	Power of task τ_i	N_{tasks}:	Total number of tasks
f_{τ_i}:	Finish time of task τ_i	$\mathbf{X}$:	Set of utilities specified by the external request
d_{τ_i}:	Data produced by task τ_i	$E_{available}$:	Energy available for task execution

can predict future energy within a 30 minute time frame with 10% accuracy.

$$E[\mathrm{Rate}_{\mathrm{EH}_{t+1}}] = \alpha \mathrm{Rate}_{\mathrm{EH}_t} + (1-\alpha)\mathrm{GAP}\frac{\sum_{i=1}^{D} \mathrm{Rate}_{\mathrm{EH}_{t+1}}}{D} \tag{4.8}$$

$$\mathrm{GAP} = \frac{\mathbf{V} * \mathbf{P}}{\sum \mathbf{P}} \tag{4.9}$$

4.2 Energy and Task Management Algorithms

The goal of the system controller is to maintain energy neutrality while satisfying performance constraints. There are three ways to approach this problem, each with a different constraint. First, the controller can determine the maximum achievable utility using only the current energy available in the system (energy constraint). Second, the controller can determine the maximum achievable utility given a time limit (time constraint). Finally, the controller can determine the expected execution time, harvesting time, and energy consumption given a desired utility (utility constraint). Each of these algorithms must meet the energy restrictions outlined in section 4.1.2 and also account for additional time needed to harvest energy if appropriate.

Table 4.2: ILP Formulation of the Energy Constraint Problem

maximize $\sum_{j=1}^{N_{tasks}} U_{\tau_j}$ subject to constraints:	
(a) $\sum_{j=1}^{N_{tasks}} e_{\tau_j} \leq E_{available}$	The energy budget is not exceeded.
(b) $U_{\tau_i} = \frac{p_{\tau_i}}{\sum_{j=1}^{N_{tasks}} p_{\tau_j}} * \sum_{j=1}^{N_{tasks}} U_{\tau_j}$	Utilities are proportional to priorities.
(c) $\forall \tau_j : s_{\tau_j} \geq \max f_{\tau_k} \forall e_{kj} \in \mathbf{E}$	The task dependencies are followed.

Table 4.3: ILP Formulation of the Time Constraint Problem

maximize $\sum_{j=1}^{N_{tasks}} U_{\tau_j}$ subject to constraints:	
(a) $\sum_{j=1}^{N_{tasks}} e_{\tau_j} \leq E_{available} + E[Rate_{EH}] * t_{wait}$	The energy budget is not exceeded.
(b) $U_{\tau_i} = \frac{p_{\tau_i}}{\sum_{j=1}^{N_{tasks}} p_{\tau_j}} * \sum_{j=1}^{N_{tasks}} U_{\tau_j}$	Utilities are proportional to priorities.
(c) $\forall \tau_j : s_{\tau_j} \geq \max f_{\tau_k} \forall e_{kj} \in \mathbf{E}$	The task dependencies are followed.
(d) $\sum_{j=1}^{N_{tasks}} t_{\tau_j} + t_{wait} \leq T_{limit}$	The time limit is met.

Table 4.4: ILP Formulation of the Utility Constraint Problem

minimize t_{total} subject to constraints:	
(a) $\sum_{j=1}^{N_{tasks}} e_{\tau_j} \leq E_{available} + E[Rate_{EH}] * t_{wait}$	The energy budget is not exceeded.
(b) $U_{\tau_i} = \frac{p_{\tau_i}}{\sum_{j=1}^{N_{tasks}} p_{\tau_j}} * \sum_{j=1}^{N_{tasks}} U_{\tau_j}$	Utilities are proportional to priorities.
(c) $\forall \tau_j : s_{\tau_j} \geq \max f_{\tau_k} \forall e_{kj} \in \mathbf{E}$	The task dependencies are followed.
(d) $t_{total} \geq \sum_{j=1}^{N_{tasks}} t_{\tau_j} + t_{wait}$	The total time must be no less than the task execution time and wait time.
(e) $U_{\tau_i} \geq U_{\tau_{specified}}$ foreach $U_{\tau_{specified}} \in \mathbf{X}$	Utilities are no less than external specifications.

To define these problems, integer linear programs (ILPs) are formulated, shown in Tables 4.2, 4.3, and 4.4 (Table 4.1 defines the variables used). Among other requirements, each ILP solution must respect the energy constraints (a), the priority constraints (b), and the task dependencies (c).

For the energy constraint problem, the goal is to maximize the system utility given the current energy available in the system. For the time constraint problem, the goal is to maximize the system utility within a given time limit (d). Optimal solutions to the energy and time constraint problems are too complex and time-consuming to be executed on an energy constrained platform; therefore, heuristics are used to solve the problems in sections 4.2.1 and 4.2.2. For the utility constraint problem, the goal is to minimize the total execution time (d) while satisfying the utility constraint(s) as specified by the external request (e). This problem can be solved by determining the expected execution time and energy consumption of the set of tasks with the provided utilities. The following sections describe each algorithm, beginning with the energy constraint algorithm.

4.2.1 Energy Constraint (EC)

To determine the maximum achievable utility within a given energy budget, a heuristic is presented that quickly determines the utility of tasks using a binary search method, shown in figure 4.4. The algorithm begins with the priorities (p) for all tasks and the energy available in the system. First, the search bounds are set such that the most

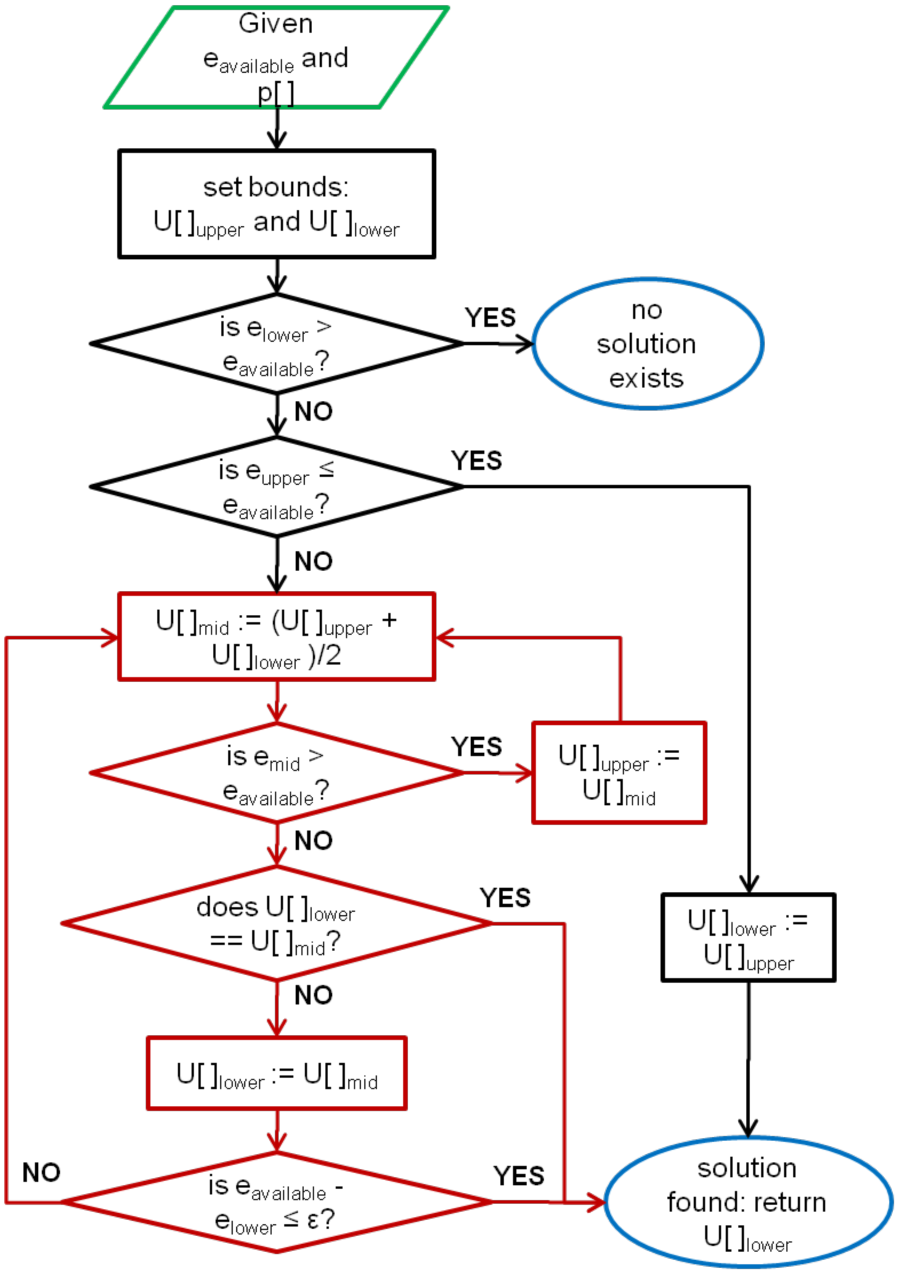

Given
$e_{available}$ and
p[ ]
set bounds:
$U[\,]_{upper}$ and $U[\,]_{lower}$
is $e_{lower} > e_{available}$?
YES
no solution exists
NO
is $e_{upper} \leq e_{available}$?
YES
NO
$U[\,]_{mid} := (U[\,]_{upper} + U[\,]_{lower})/2$
is $e_{mid} > e_{available}$?
YES
$U[\,]_{upper} := U[\,]_{mid}$
NO
does $U[\,]_{lower} == U[\,]_{mid}$?
YES
NO
$U[\,]_{lower} := U[\,]_{upper}$
$U[\,]_{lower} := U[\,]_{mid}$
NO
is $e_{available} - e_{lower} \leq \varepsilon$?
YES
solution found: return $U[\,]_{lower}$

Figure 4.4: Energy Constraint Algorithm

important task (highest p) has a utility of either U = 1 or U = 0, denoting the upper and lower bounds respectively. The upper, lower, and midpoint task utilities are evaluated to determine how much energy is needed, which is then compared to the energy available in the energy buffer. Because the possible utility values are numerous, and there may not exist a set of utilities that consume exactly the energy available, a threshold ϵ is set such that the resulting energy consumption must be within ϵ of the energy available in the system. This threshold limits the search time for the algorithm and consequently, the time and energy overhead of the system control as well.

4.2.2 Time Constraint (TC)

In addition to the energy constraint algorithm, a time constraint (TC) algorithm is presented. Within a given time limit, the controller uses the heuristic shown in figure 4.5 to determine a set of task utilities that satisfies the request. The heuristic consists of two basic steps: first, satisfy the time limit and second, satisfy the energy constraint.

Step 1: Satisfy the time constraint

The first step in solving this problem uses a binary search method similar to that used in the energy constraint algorithm in section 4.2.1. The time required for upper, lower, and midpoint task utilities is compared to the time limit to find a set of task utilities whose execution times satisfy the time constraint. As with the EC algorithm, a threshold ϵ is set such that the resulting execution time must be within ϵ of the time limit.

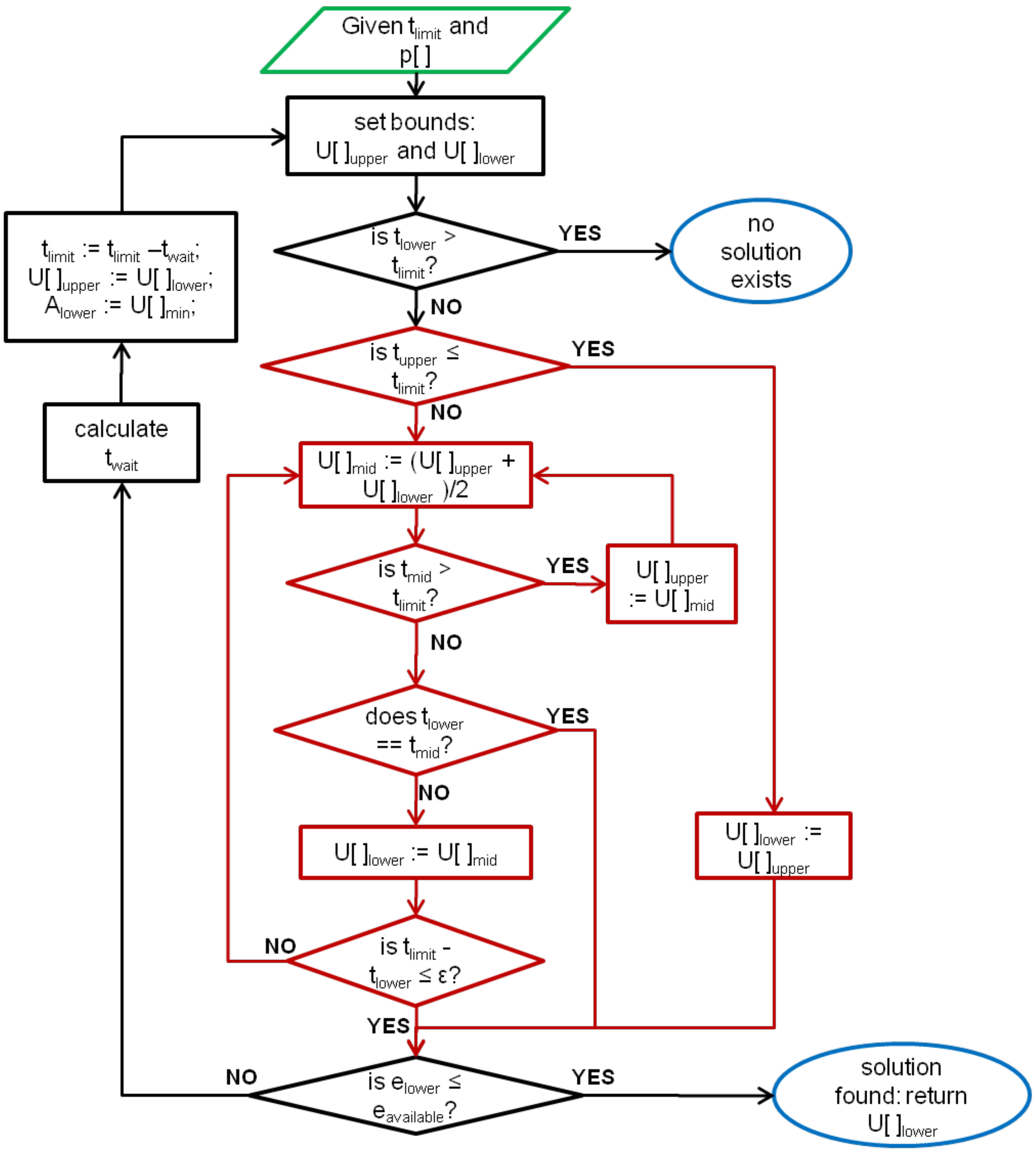

Given t_{limit} and p[ ]
set bounds: $U[\ ]_{upper}$ and $U[\ ]_{lower}$
is $t_{lower} > t_{limit}$?
YES
no solution exists
NO
is $t_{upper} \le t_{limit}$?
YES
NO
$U[\ ]_{mid} := (U[\ ]_{upper} + U[\ ]_{lower})/2$
is $t_{mid} > t_{limit}$?
YES
$U[\ ]_{upper} := U[\ ]_{mid}$
NO
does $t_{lower} == t_{mid}$?
YES
NO
$U[\ ]_{lower} := U[\ ]_{mid}$
$U[\ ]_{lower} := U[\ ]_{upper}$
NO
is $t_{limit} - t_{lower} \le \varepsilon$?
YES
NO
is $e_{lower} \le e_{available}$?
YES
solution found: return $U[\ ]_{lower}$
calculate t_{wait}
$t_{limit} := t_{limit} - t_{wait}$; $U[\ ]_{upper} := U[\ ]_{lower}$; $A_{lower} := U[\ ]_{min}$;

Figure 4.5: Time Constraint Algorithm

Step 2: Satisfy the energy constraint

After finding a set of task utilities that satisfies the time constraint, the algorithm compares the amount of required energy with the amount of available energy. If sufficient energy exists, a solution has been found; if not, then the amount of time to wait to harvest additional energy is determined.

One option to solve this problem would be to try every possible combination of wait time and execution time and choose the best solution; however, on an energy-constrained embedded platform, such a high level of computational complexity is unreasonable. Instead, the algorithm estimates the time to wait under the assumption that the ratio between the current energy and time limit will be proportional to the ratio for a new energy consumption and time limit. Based on this premise, the wait time is computed using equation 4.10, and the time limit is altered to account for this time by reducing the original time limit by the wait time. Then, step 1 of the algorithm is executed again to determine a new set of utilities that will satisfy the reduced time limit.

$$\mathrm{t_{wait}} = \frac{\mathrm{E_{needed}}}{E[\mathrm{Rate_{EH}}] + \frac{\mathrm{E_{required}}}{\mathrm{T_{limit}}}} \tag{4.10}$$

$$\mathrm{T_{limitNEW}} = \mathrm{T_{limitOLD}} - \mathrm{t_{wait}} \tag{4.11}$$

4.2.3 Utility Constraint (UC)

The goal of the final algorithm, the utility constraint (UC) algorithm, is to determine the total time and energy needed to execute all tasks at a specified level of utility. If the energy needed to execute all tasks at the desired utility exceeds the energy available, the total delay time needed to harvest additional energy, $\mathrm{t_{wait}}$, is estimated. If the energy harvesting rate is not positive, which is likely the case during the night, then no additional energy can be harvested, and no solution exists. If the energy harvesting rate is positive, the total time, $\mathrm{t_{total}}$, needed to execute the task set is the sum of the total execution time and the wait time.

$$\mathrm{t_{total}} = \sum_{\mathrm{j=1}}^{\mathrm{N_{tasks}}} \mathrm{t}_{\tau_\mathrm{j}} + \mathrm{t_{wait}} \tag{4.12}$$

The steps for solving this problem are shown in the flow chart in figure 4.6 (assumes a single CPU). Using the application-specific functions from equations 4.2 and 4.3, the controller determines $\mathrm{t_{wait}}$ and $\mathrm{t_{total}}$ based on the predicted energy-harvesting rate, $E[\mathrm{Rate_{EH}}]$(section 4.1.3).

4.3 Steady State Operation

Steady state operation describes the periodic execution of tasks on a system. In some systems, steady state operation is defined by the method of duty cycling. Duty cycling, however, does not provide detailed information on the specifics of a set of tasks and assumes constant power for all tasks execution. Thus, for the system targeted in this

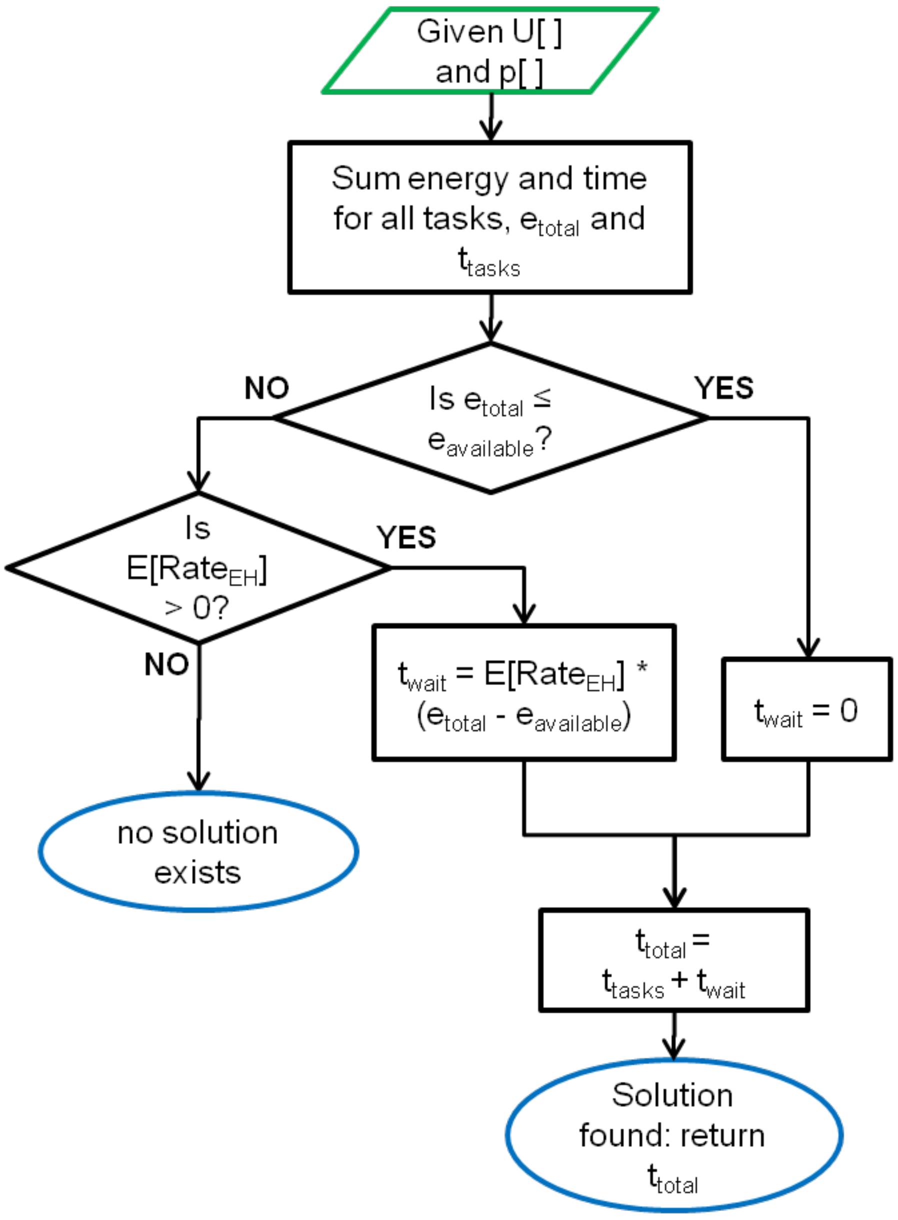

Given U[ ] and p[ ]
Sum energy and time for all tasks, e_{total} and t_{tasks}
Is $e_{total} \leq e_{available}$?
NO
YES
Is $E[Rate_{EH}] > 0$?
YES
NO
$t_{wait} = E[Rate_{EH}] * (e_{total} - e_{available})$
$t_{wait} = 0$
no solution exists
$t_{total} = t_{tasks} + t_{wait}$
Solution found: return t_{total}

Figure 4.6: Utility Constraint Algorithm

thesis, steady state operation defines the periodic execution of a task graph, the utility of which is determined using the energy constraint algorithm described in section 4.2.1.

As the energy harvesting rate assumed for this system is variable, the amount and rate of energy harvesting must determine the rate of task execution and the types of tasks to be executed. The basic intuition behind steady state operation control is the following: (1) tasks should be executed at a consistent rate when possible and (2) the utility of tasks should be adapted to fit the energy constraints of the system.

System time is divided into two periods: T_{positive} and T_{negative}. T_{positive} indicates the time period when energy is being harvested, while T_{negative} represents the time period when energy harvesting is negligible. For the system targeted in this work, a solar-powered system, T_{positive} is equivalent to the span of a day, while T_{negative} is the time during the night. T_{positive} may change over time (certainly for solar-powered systems), and is thus updated based on previous days' positive time periods.

The execution rate, r, indicates the number of times per period T_{positive} that a set of tasks is executed. Thus, the task graph currently selected will be executed every t_{frame} time units, where $t_{\text{frame}} = T_{\text{positive}}/r$,. Because time period lengths and energy harvesting rates change over time, the execution rate should be altered to maximize a desired parameter. Possible parameters to optimize include: total utility achieved in a single period, average utility for each execution, and number of maximum utility executions per period. For the following algorithm, the number of maximum utility executions per period is chosen to be maximized based on the needs of the application (further described in section 5.4), and thus the rate r is adapted based on this metric.

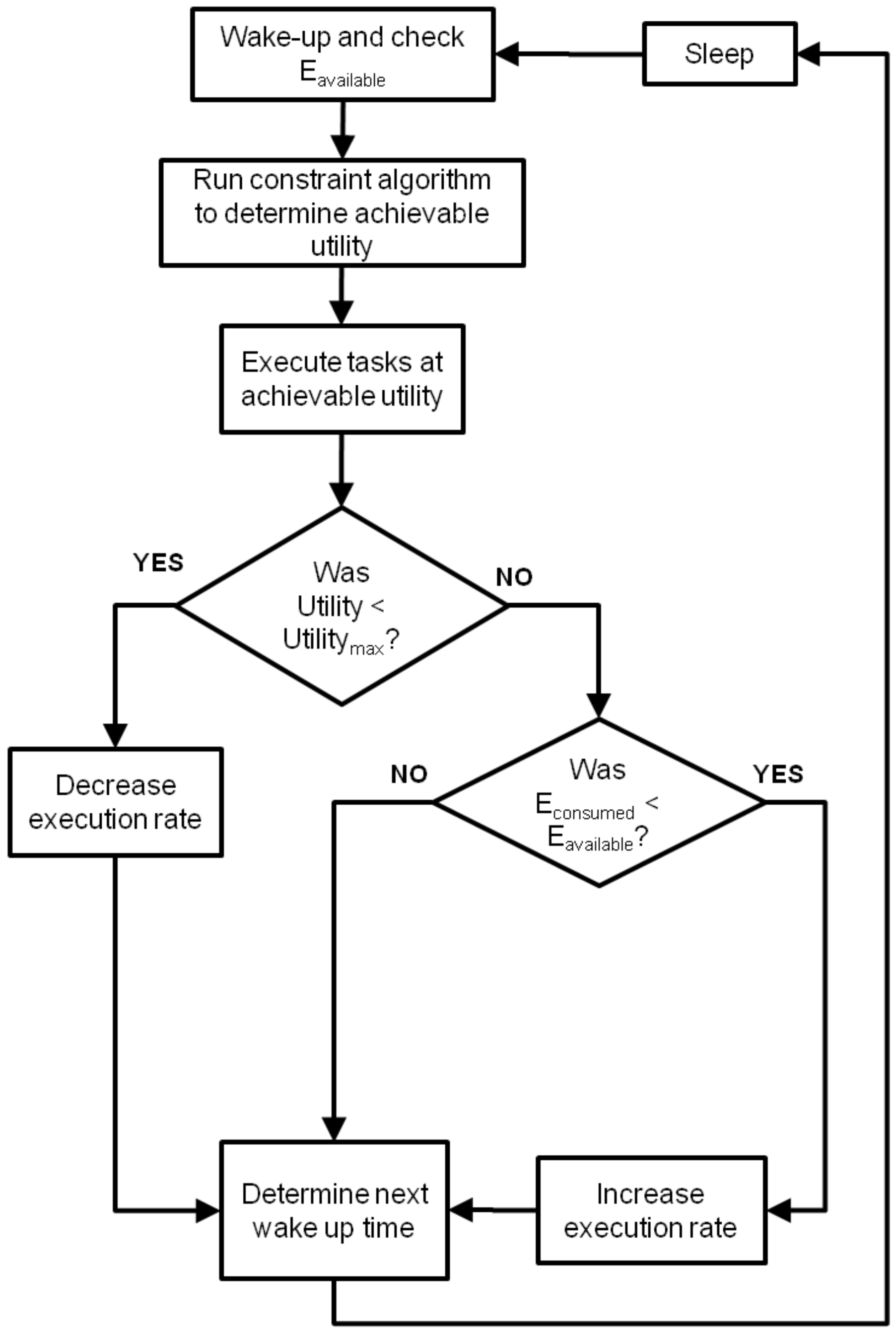

Wake-up and check $E_{available}$
Sleep
Run constraint algorithm to determine achievable utility
Execute tasks at achievable utility
YES
Was Utility < $Utility_{max}$?
NO
Decrease execution rate
NO
Was $E_{consumed}$ < $E_{available}$?
YES
Determine next wake up time
Increase execution rate

Figure 4.7: Steady State Control Flow

The flow chart for steady state operation is shown in figure 4.7. Every t_{frame} time units the energy constraint algorithm is executed to determine the utility at which to execute tasks.[1] As described in section 4.1.2, the energy available during steady state operation, $E_{\text{available}}$, equals the current energy in the buffer, E_{buffer}, minus the steady state energy threshold, E_{steady}. Thus, the energy constraint equals $E_{\text{available}}$. After execution, the achievable utility is compared to the maximum utility of the set of tasks. If the maximum achievable utility is less than the maximum utility, r is decreased. If the maximum achievable utility equals the maximum possible utility and leftover energy exists, then r is increased. The ideal situation occurs when the maximum achievable utility equals the maximum possible utility, and the energy in the buffer equals the steady state energy threshold E_{steady}; in this case, the execution rate is not changed.

4.4 External Trigger State Operation

In contrast to steady state operation, external trigger state operation occurs when an external device such as another sensor node or an unmanned vehicle initiates a request to the system. Steady state operation is interrupted, and the system controller responds to the request. Because the external device does not know the energy status of the system, the system must be able to adapt to the request according to its own energy availability. The external device can specify either the desired utility or the maximum time limit, so the controller uses either the utility constraint or the time constraint algorithm.

[1]The time constraint algorithm could also be used by setting the time limit equal to the length of a frame, t_{frame}.

Following a significant event, such as an earthquake, fire, or car accident, an external device may request results from the system indicating a required level of accuracy; in this situation, the controller runs the utility constraint algorithm (section 4.2.3). The trigger specifies a desired utility for a specific task, and the controller then uses the set of priorities $\{p_{\tau_1}, p_{\tau_2}, ..., p_{\tau_N}\}$ to calculate the remaining task utilities. In a state of emergency, the execution time of the tasks may not be vital, and thus is not considered in a utility constraint external request.

In contrast, data mules, such as that described by Sugihara and Gupta [29], must typically collect data from many nodes, and thus, it may be desirable to designate a time limit in which results must be reported. When the external device specifies a time limit, the controller runs the time constraint algorithm (section 4.2.2) to determine a set of task utilities that satisfies the request. After finding a set of task utilities that satisfies both the time limit and the energy constraint, the system can inform the external device of the maximum achievable utility for each task that can be executed in the given time limit, T_{limit}, which includes the actual time spent executing tasks and the time, t_{wait}, spent harvesting additional energy. Estimating these values ahead of time provides the external device with valuable information, enabling it to, for example, collect data from other nodes while it waits for a result.

4.5 Summary

This chapter presented application-independent methods for energy and task management in externally-triggered, energy-harvesting sensing systems. First, the system was described, which is composed of an energy source, an energy buffer, a predictor, a controller, a set of tasks, and an external device. Next, three algorithms for balancing system performance with time and energy constraints were presented. Finally, external trigger state and steady state operation were described, outlining a method for responding to external requests and for adapting execution rate based on the energy harvesting rate. The following chapter will explain in detail the experiments performed to test the algorithms in external trigger state and steady state operation.

Chapter 4, in part, has been submitted for publication of the material as it may appear in Networked Sensing Systems, 2009, Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this paper. Chapter 4, in part, is currently being prepared for submission for publication of the material. Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this material.

Chapter 5

Results

To test and apply the contributions presented in the previous chapter, the controller and predictor for external trigger state and steady state operation were implemented. Inputs to the simulation include the task graph, task execution characteristics, energy source data, and storage thresholds. The task graph and application-specific execution time functions are defined for a structural health monitoring (SHM) application using data obtained from hardware measurements and code estimations. The results of the simulations show that energy neutrality is maintained, while task utility is adapted according to system and environmental constraints.

The remainder of this chapter contains the application details and simulation results. First, in section 5.1, the SHM application is described, and the definitions for the application-specific functions are given. Section 5.2 describes the evaluation setup, followed by the results from each of the energy and task management algorithms in section 5.3. Finally, steady state and external trigger state operation are discussed in sections 5.4 and 5.5 respectively. The chapter concludes in section 5.6, outlining the significance of the results.

5.1 Application

Because SHiMmer, described in section 3.5, harvests energy from the environment and performs numerous tasks related to sensing, processing, and transmitting, this SHM platform is a perfect application for testing the energy and task management methods proposed in this thesis. Therefore, the methods described in chapter 4 were implemented for SHiMmer. In order to apply a specific application to the system controller, a task graph and task execution characteristics need to be defined. First, an example SHiMmer task graph is created, shown in Figure 5.1. For each task, $\mathrm{f}\left(\mathrm{U}_{\tau}\right)$ is defined in terms of the data flow, the worst case execution time (WCET), and the power consumption. Table 5.1 shows the execution time and data formulas defined for each task, and table 5.2 shows the data used.

As described in chapter 3, actuation and acquisition involves the use of piezoelectric transducers (PZTs) that serve as both actuators and sensors. A path is defined as the pairing of two PZTs. The SHiMmer platform connects to 16 PZTs and thus, can evaluate up to 120 paths, as shown in figure 3.3. In the task graph, each of the tasks act/acq_1 – act/acq_{120} represents one of the 120 possible paths that can be evaluated to perform actuation and acquisition (*act/acq*). The utility for an *act/acq* task can be increased by evaluating each path multiple times, up to $\mathrm{MAX}_{\mathrm{iterations}}$.

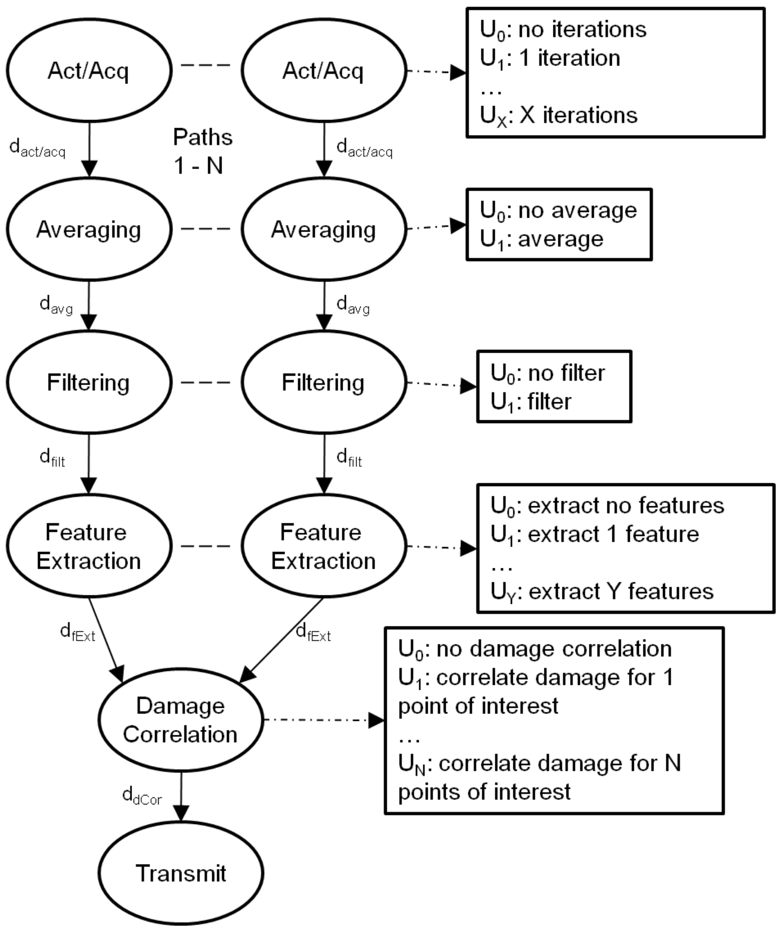


Figure 5.1: SHM Task Graph

Table 5.1: SHM Execution Definitions

Task	**Purpose**	**Execution Time: f** (U_τ)	**Data Produced:** d_τ	**Power:** P_τ
$act/acq_1 - act/acq_{120}$	Actuate & Acquire (*act/acq*)	$\mathrm{t} = (\mathrm{t_{act}} + \mathrm{t_{sense}}) * \mathrm{N_{iterations}}$ where $\mathrm{N_{iterations}} = \left\lceil \frac{e^{4.6*\mathrm{U}}-1}{\mathrm{MAX_{iterations}}} \right\rceil$	$\mathrm{d} = \mathrm{N_{iterations}} * \mathrm{d_{path}}$	3.5 W 780 mW
$avg_1 - avg_{120}$	Average (*avg*)	$\mathrm{t} = 0 \Vert \mathrm{t} = \mathrm{t_{avg}} * (\mathrm{N_{iterations}} - 1)$	$\mathrm{d} = \mathrm{d}_{act/acq} \Vert \mathrm{d} = \mathrm{d_{path}}$	680 mW
$filt_1 - filt_{120}$	Filter (*filt*)	$\mathrm{t} = 0 \Vert \mathrm{t} = \mathrm{t_{filt}} * \frac{\mathrm{d}_{avg}}{\mathrm{d_{path}}}$	$\mathrm{d} = \mathrm{d}_{avg}$	680 mW
$fExt_1 - fExt_{120}$	Extract Feature (*fExt*)	$\mathrm{t} = \lceil \mathrm{MAX_{blocks}} * \mathrm{U} \rceil$	$\mathrm{d} = \lceil \mathrm{MAX_{blocks}} * \mathrm{U} \rceil *$ $\frac{\mathrm{d}_{avg}}{\mathrm{d_{path}}} * \mathrm{d_{block}}$	680 mW
dCor	Correlate Damage (*dCor*)	$\mathrm{t} = \lceil \mathrm{MAX_{poi}} * \mathrm{U} \rceil *$ $(\mathrm{t_{poi}} * \mathrm{N_{paths}} + \mathrm{t_{add}} * (\mathrm{N_{paths}} - 1))$	$\mathrm{d} = \lceil \mathrm{MAX_{poi}} * \mathrm{U} \rceil * \mathrm{d_{poi}}$	680 mW
trans	Transmit (*trans*)	$\mathrm{t} = \mathrm{d}_{dCor} * \mathrm{t_{transmit/byte}} + \mathrm{t_{wakeup}}$ $\Vert \mathrm{t} = 0$	d = 0	168 mW

Table 5.2: SHM Execution Data

$\mathrm{MAX_{iterations}}$:	10	$\mathrm{t_{act}}$:	0.1 ms	$\mathrm{t_{block}}$:	0.5596 ms	$\mathrm{d_{path}}$:	20 KB
$\mathrm{MAX_{paths}}$:	120	$\mathrm{t_{acq}}$:	1 ms	$\mathrm{t_{poi}}$:	0.0107 ms	$\mathrm{d_{block}}$:	2 B
$\mathrm{MAX_{poi}}$:	90	$\mathrm{t_{avg}}$:	4.004 ms	$\mathrm{t_{transmit/byte}}$:	0.071 ms	$\mathrm{d_{poi}}$:	2 B
$\mathrm{MAX_{blocks}}$:	50	$\mathrm{t_{filt}}$:	1294.6 ms	$\mathrm{t_{wakeup}}$:	13.2 ms	$\mathrm{t_{add}}$:	0.0004 ms

After the data is acquired from the previous tasks, it is processed. Tasks *avg*, *filt*, *fExt*, and *dCor* represent the different types of processing that SHiMmer performs as described in section 3.3 and are executed on a TMS320C2811 Digital Signal Processor [31]. Averaging, represented by tasks $avg_1 - avg_{120}$, reduces the amount of data acquired by the PZTs by averaging multiple iterations of each path into a single data set. If data is averaged, then the utility is 0.25 ($\text{U} = 0.25$); if data is not averaged, then the utility is 0.0 ($\text{U} = 0.0$). Filtering ($filt_1 - filt_{120}$) removes noise from the signal using a matching filter. If data is filtered, then the utility is 0.75 ($\text{U} = 0.75$); if data is not filtered, then the utility is 0.0 ($\text{U} = 0.0$). Feature extraction, $fExt_1 - fExt_{120}$, divides the signal into N blocks, up to $\text{MAX}_{\text{blocks}}$, and compares each block to a baseline signal. The utility for feature extraction increases as the number of features evaluated increases. Finally, task *dCor* performs damage correlation, combining the features extracted from all paths to determine if damage exists at a specific point of interest on the structure for up to MAX_{poi}.

Task *trans*, the exit task in the graph, represents the transmitting of data. The utility of the data transmission affects whether or not the data is transmitted. If the utility is greater than zero, the data is transmitted; if the utility equals zero, the data is not transmitted. While this method of utility measure is simple due to the needs of the application, more complicated utility measurements could be used such as varying the signal strength, data rate, or amount of data transmitted.

5.2 Evaluation Setup

The three constraint algorithms were implemented using the SHM task graph and execution definitions in the previous section as task input and applied to external trigger state and steady state operation. The data collected from SHiMmer's solar panel and supercapacitor are used as input for the energy source and energy storage buffer. The energy harvesting circuit used for SHiMmer enables the node to harvest energy from a 100 cm^2 solar cell and store up to 780 J in a 250 F, 2.5 V supercapacitor. The relationship between the voltage of the solar panel and charging rate for the supercapacitor is defined using the results obtained from twenty days of measurement, further described by Recas *et al.* [21]. Figure 5.2 shows the actual solar panel voltage, the predicted voltage obtained by the prediction model, and the predicted voltage using only the EWMA. As expected, the voltage of the solar panel is high during the day and low during the night, and the predicted voltage using the prediction model closely tracks the actual voltage.

5.3 Energy and Task Management Algorithms

While testing each of the three energy and task management algorithms, the amount of energy available in the buffer, $E_{available}$, is varied from 100J-400J, depending on the requirements of the systems and the size of the energy buffer. Unless otherwise noted, the priorities for all tasks are 1.0. For each algorithm, the results are evaluated based on the amount of energy in the buffer, the rate of energy harvesting, the priorities,

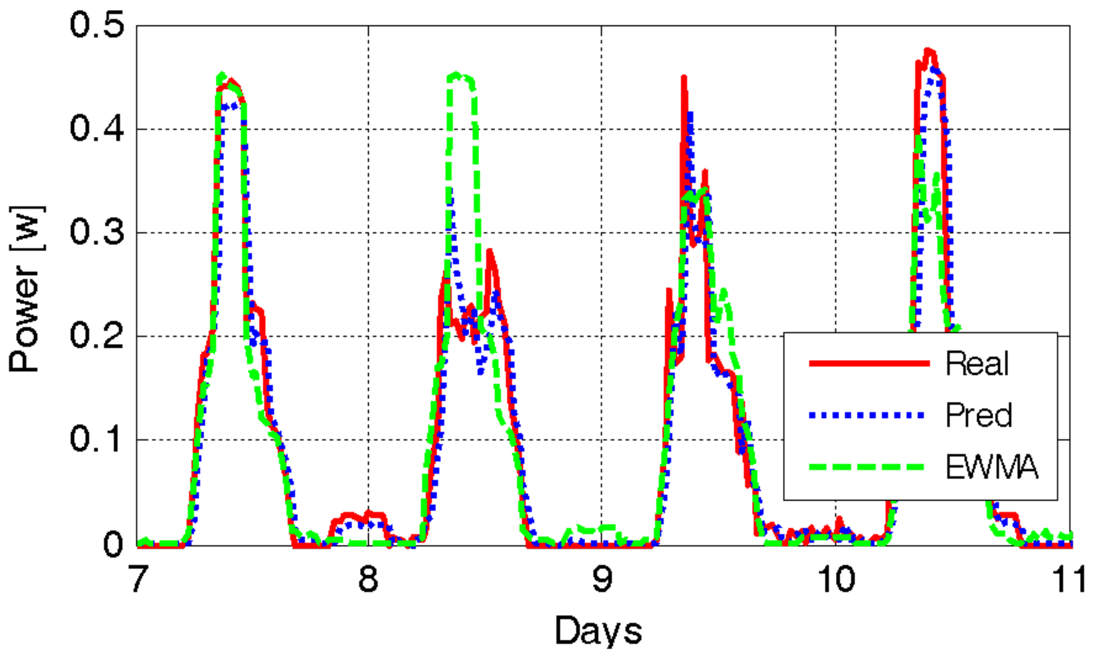


Figure 5.2: Solar Panel Data: Actual, Predicted, and EWMA

and the amount of processing performed.

5.3.1 Energy Constraint (EC) Algorithm

The first algorithm, the energy constraint (EC) algorithm, determines the highest achievable utility given the amount of energy available in the buffer. In the following subsections, the energy constraint algorithm is evaluated under different conditions, such as varying the amount of energy in the buffer, changing task priorities, and changing the amount of processing performed.

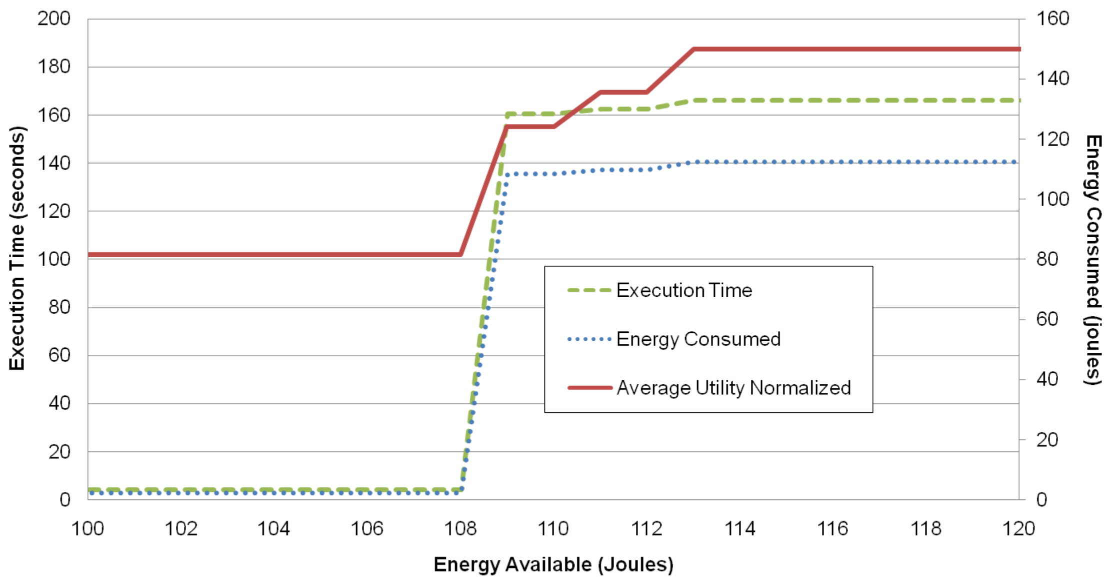


Figure 5.3: EC: Effect of Available Energy on Execution Time and Utility

Energy Harvesting

Because the purpose of the energy constraint algorithm is to obtain results given only the energy available, no time is spent harvesting additional energy. Therefore, this first algorithm is not dependent on the energy harvesting rate, only the energy stored in the buffer. Figure 5.3 shows the results of the energy constraint algorithm for different amounts of available energy, from 100 to 120 joules, as these amounts are the most interesting. As the energy available increases, the execution time and the average utility (normalized to a range from 1 to 200) also increase.

Priority

To test the impact of priority on achievable utility in this SHM application, three modes of priority combinations are defined, shown in table 5.3. For the first mode (mode

Table 5.3: Modes with Various Priority Combinations

	Paths 1-40	Paths 41-60	Paths 61-80	Paths 81-120
Mode 1:	p = 1.0	p = 1.0	p = 1.0	p = 1.0
Mode 2:	p = 1.0	p = 1.0	p = 0.6	p = 0.6
Mode 3:	p = 1.0	p = 0.6	p = 0.6	p = 0.1

1), all priorities for all paths are set to 1.0. In the second mode (mode 2), one half of the tasks for all paths (*act/acq*, *avg*, *filt*, and *fExt*) are given a priority of 1.0, while the other half have a priority of 0.6. Finally, in the third mode (mode 3), one third of the tasks for all paths have a priority of 1.0, on third have 0.6 and the last third have 0.1. In all modes, the *dCor* task has a priority of 1.0. Each of these modes were run varying the available energy from 0 to 140 joules. As seen in figure 5.4, obtaining full utility for all tasks in mode 1 requires 112 joules. If mode 2 or mode 3 is used, however, full utility for the more important tasks (those with p = 1.0) can be achieved with less energy, 47% less energy for mode 2 (at 59 joules) and 64% less for mode 3 (at 40 joules).

Processing

In contrast to typical WSNs, this SHM application should perform complex processing on the data, dramatically reducing the amount of data transmitted. To provide users options on the amount of processing performed, the energy constraint algorithm is tested using different levels of processing that consume different amounts of energy and take various lengths of time. The four levels tested are: *no processing*, *averaging only*, *averaging and filtering*, and *all processing*. These levels are specific to this application and are summarized in table 5.4.

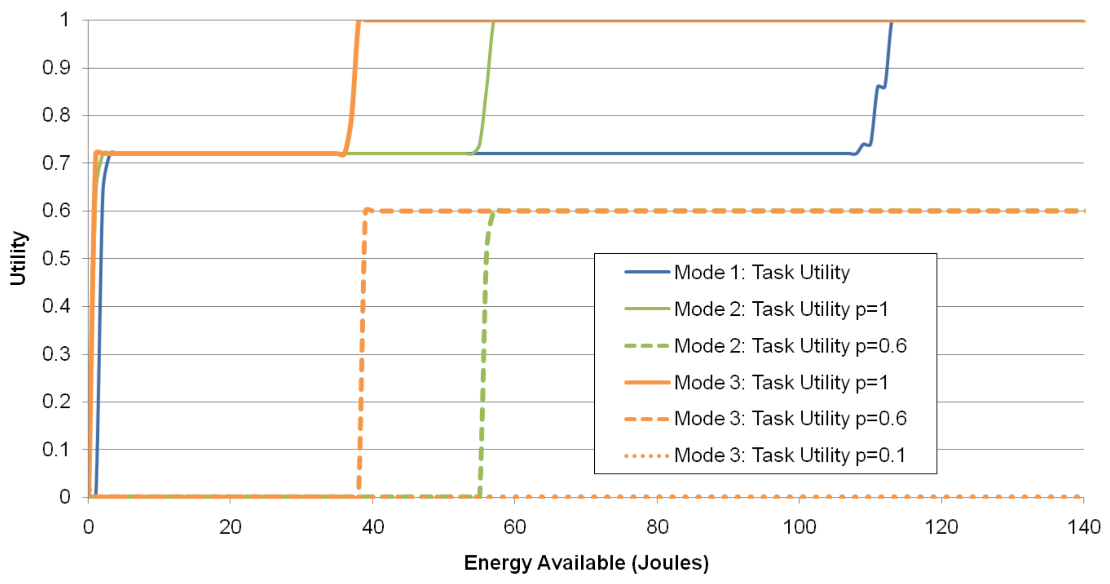


Figure 5.4: EC: Effect of Priority on Achievable Utility

Figures 5.5 and 5.6 show how execution time and average utility vary depending on both the energy available and the amount of processing performed. The average utility shown in figure 5.6 equals the average utility for all tasks with p $= 1.0$, thus the maximum average utility differs for the different levels of processing. Due the massive amounts of data produced by the *act/acq* task, transmitting raw data (*no processing*) can require up to 300 joules and 1700 seconds to achieve full utility. Simply averaging the multiple path iterations (*averaging only*) can reduce the energy and time significantly, as the amount of data per path is reduced from (at worst) 200 KB to 20 KB.

5.3.2 Time Constraint (TC) Algorithm

The second constraint algorithm, the time constraint algorithm, determines the highest achievable utility given a time limit. In the following subsections, this algorithm

Table 5.4: Levels of Processing

Level	Description	Priority Settings
no processing	no processing performed on data; raw data transmitted	$p_{act/acq,\ trans} = 1.0$, $p_{avg,\ filt,\ fExt,\ dCor} = 0.0$
averaging only	multiple iterations for each path averaged and transmitted	$p_{act/acq,\ avg,\ trans} = 1.0$, $p_{filt,\ fExt,\ dCor} = 0.0$
averaging and filtering	multiple iterations for each path averaged, filtered, and transmitted	$p_{act/acq,\ avg,\ filt,\ trans} = 1.0$, $p_{fExt,\ dCor} = 0.0$
all processing	all processing performed on data; results transmitted	$p_{act/acq,\ avg,\ filt,\ fExt,\ dCor,\ trans} = 1.0$

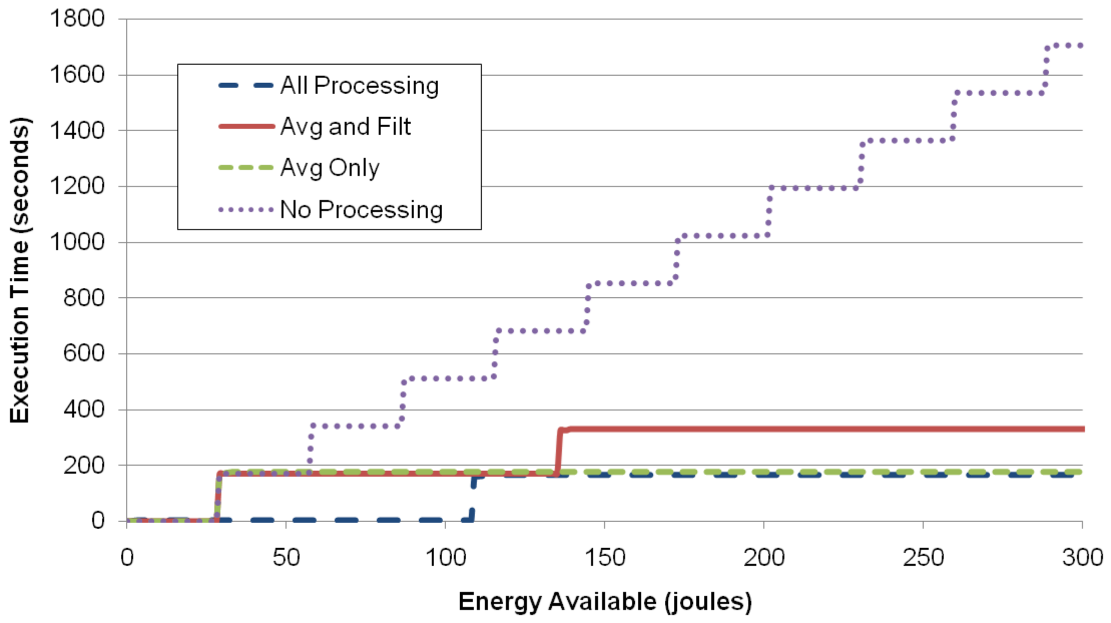


Figure 5.5: EC: Effect of Processing on Execution Time

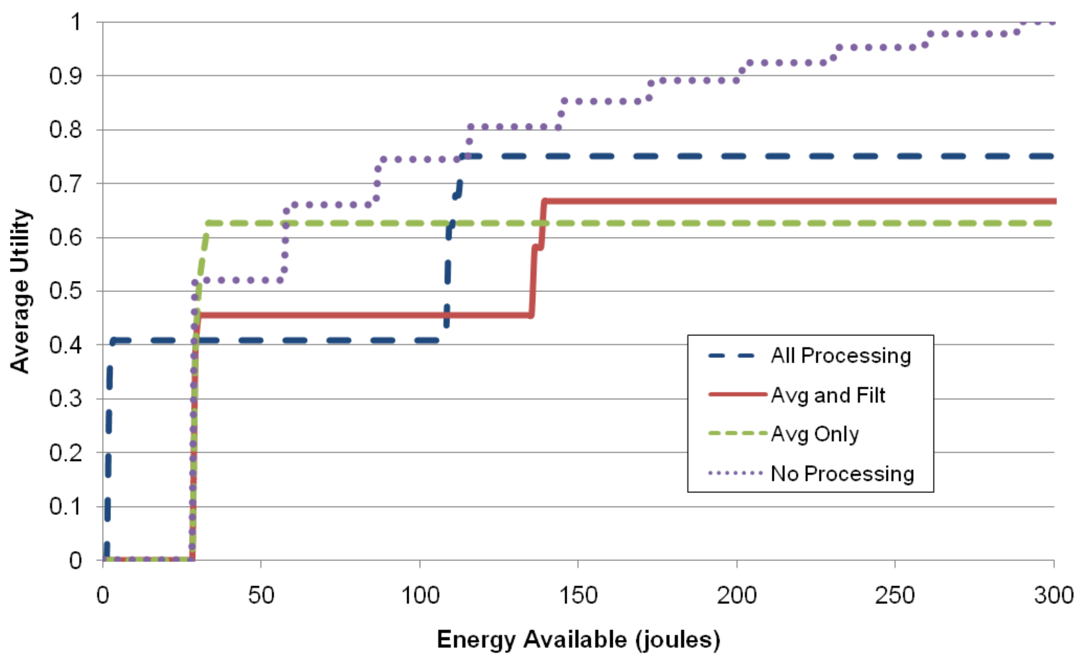


Figure 5.6: EC: Effect of Processing on Average Utility

is tested under different conditions, such as varying the rate of energy harvesting, changing the task priorities, and changing the amount of processing performed, but first, the algorithm is evaluated with different threshold values.

Because the running time of the time constraint algorithm depends on the granularity of the utility values, there may be too many values to search for on an energy-constrained platform. Thus, the heuristic presented in chapter 4 uses a threshold value ϵ to calculate the acceptable percentage of deviation from the specified time limit and constrain the amount of overhead. To determine an appropriate value for ϵ, the optimal values are compared with the results using several threshold values. The optimal values are obtained for various time limits using an exhaustive search with the utility constraint algorithm. Figure 5.7 shows the achievable utility for time limits with thresh-

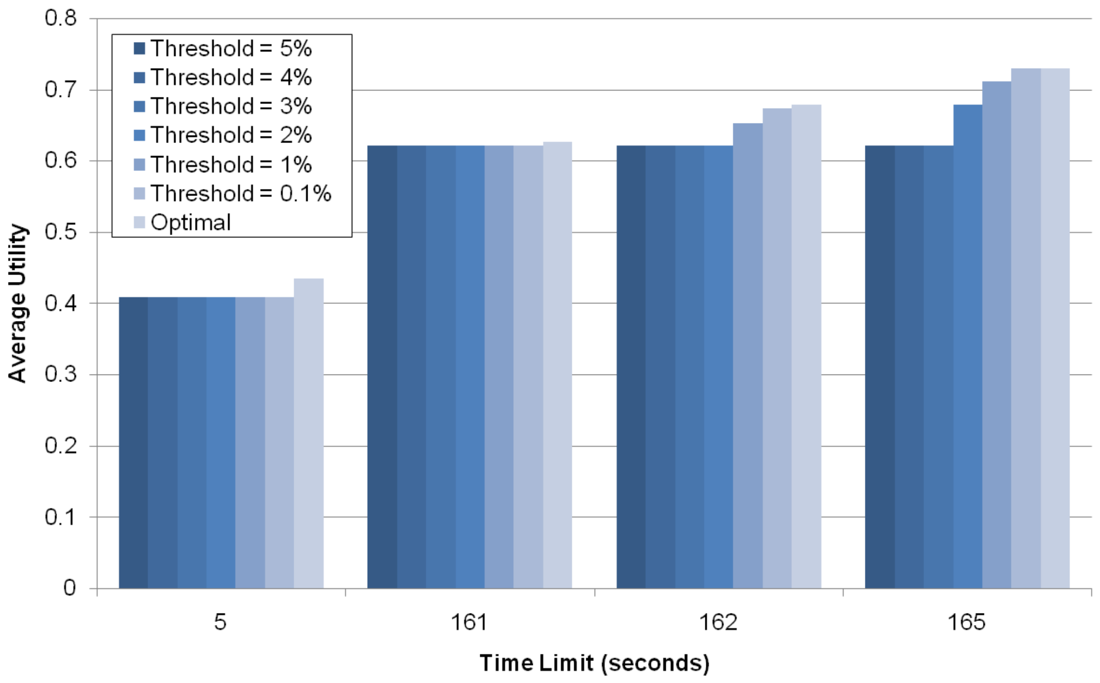


Figure 5.7: TC: Varying the Threshold Value

olds varying from 5% to 0.1%. For some time limits, a very low threshold will achieve a higher utility, but at the cost of higher processing overhead. For the tests presented in this section, a threshold of $\epsilon = 2\%$ is used to reduce the amount of overhead yet obtain acceptable results.

Energy Harvesting

The rate of energy harvesting significantly influences the obtainable utility when insufficient energy is stored in the buffer. To test the effect of the energy harvesting rate, the time constraint algorithm is tested at different times during the day. The time limit is varied from 1 minute to 35 minutes at three distinct times of a day: the peak of a sunny day ($\text{Rate}_{\text{EH}} = 0.1022$ mV/s), the peak of a cloudy day ($\text{Rate}_{\text{EH}} = 0.0340$ mV/s),

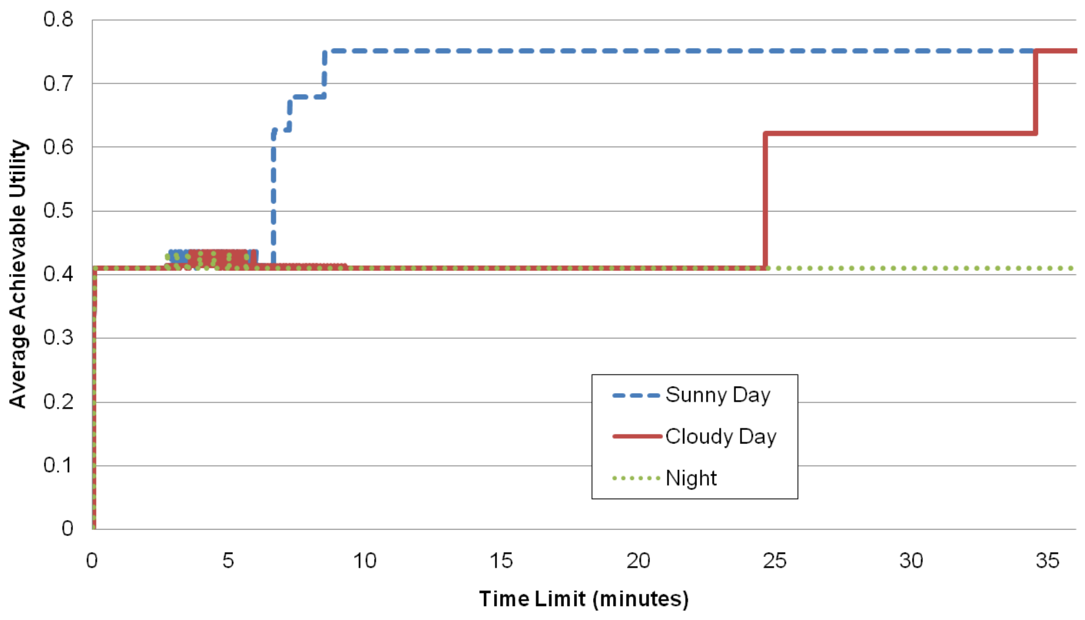


Figure 5.8: TC: Achievable Utility at Different Times of Day

and the middle of the night ($\text{Rate}_{\text{EH}} = 0$ mV/s). Figure 5.8 shows the impact of energy harvesting on achievable utility. During a sunny day, energy can be harvested at a high rate, and thus full utility can be obtained when the time constraint exceeds 8 minutes and 30 seconds. During a cloudy day, full utility can still be obtained but only if the time limit is almost 35 minutes. During the night, however, there is no energy to harvest and thus only a maximum utility of 0.41 can be obtained.

Priority

To test the impact of priority on achievable utility for the time constraint algorithm, the time constraint algorithm is run using the same priority modes described in table 5.3 with energy available equal to 200 J. The results for the three modes are shown

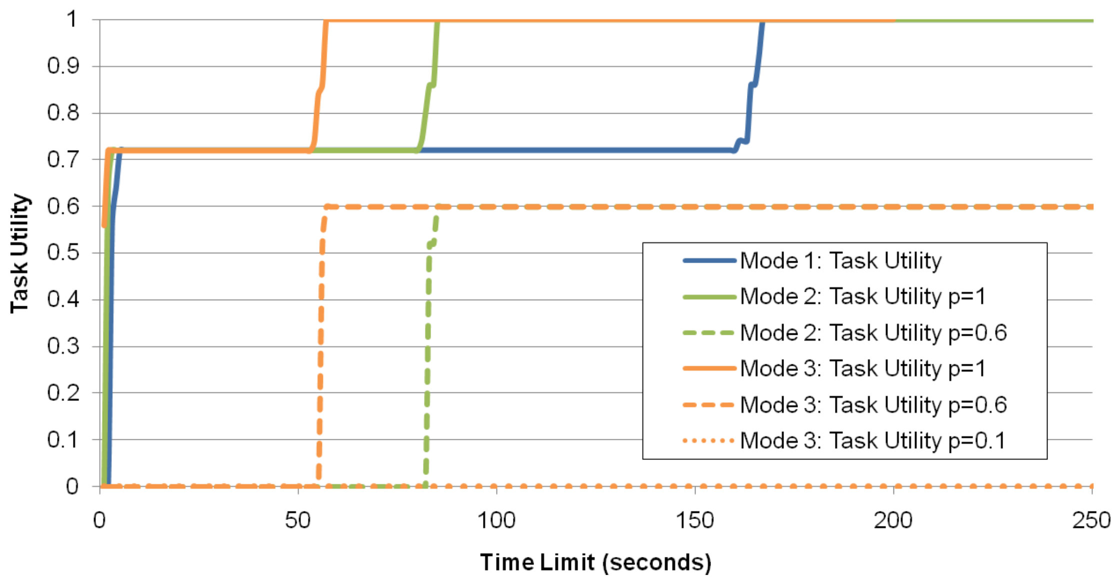


Figure 5.9: TC: Impact of a Task's Priority on Achievable Utility

in Figure 5.9 where the achievable task utility of the *fExt* task changes with time limit and mode. As the time limit in the request is increased, mode 1 requires over 150 seconds to obtain full utilty, while mode 3, for example, takes less than 60 seconds. The results show that for mode 3, the more important tasks (with p = 1.0) can achieve full utility, while the less important tasks settle for either U = 0.6 or U = 0.1. Distinguishing between the importance of types of tasks allows the more important tasks to achieve full utility at the expense of the other tasks.

Processing

To test the effect of processing with the time constraint algorithm, the same four processing levels described in table 5.4 are used. The time-constraint algorithm is tested using these four combinations, assuming the available energy to be 400 joules and the

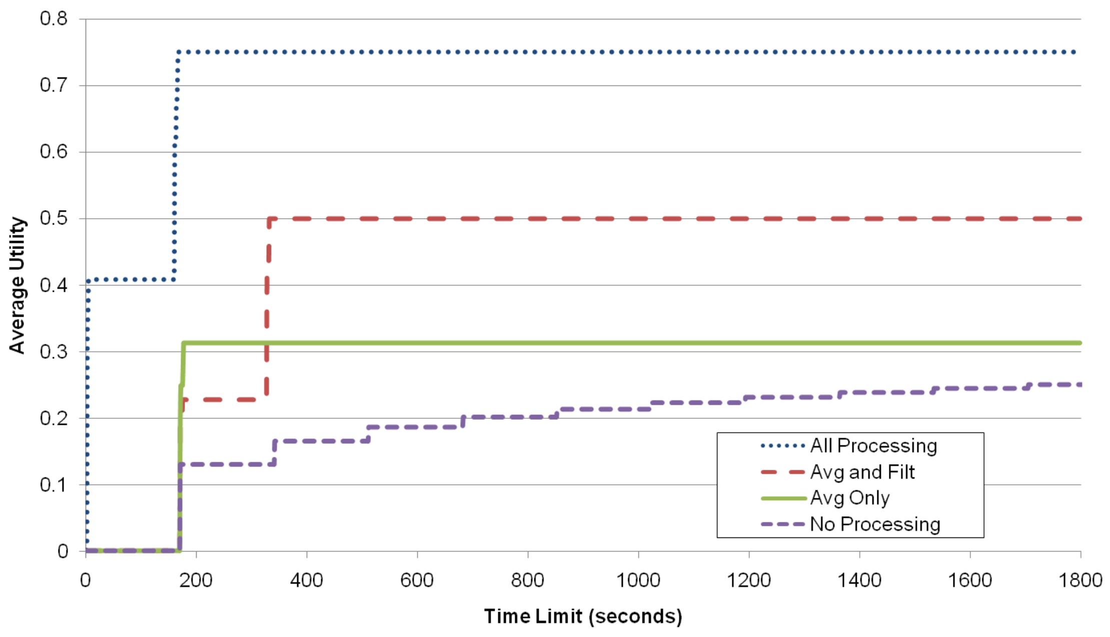


Figure 5.10: TC: Achievable Task Utility for Various Levels of Processing

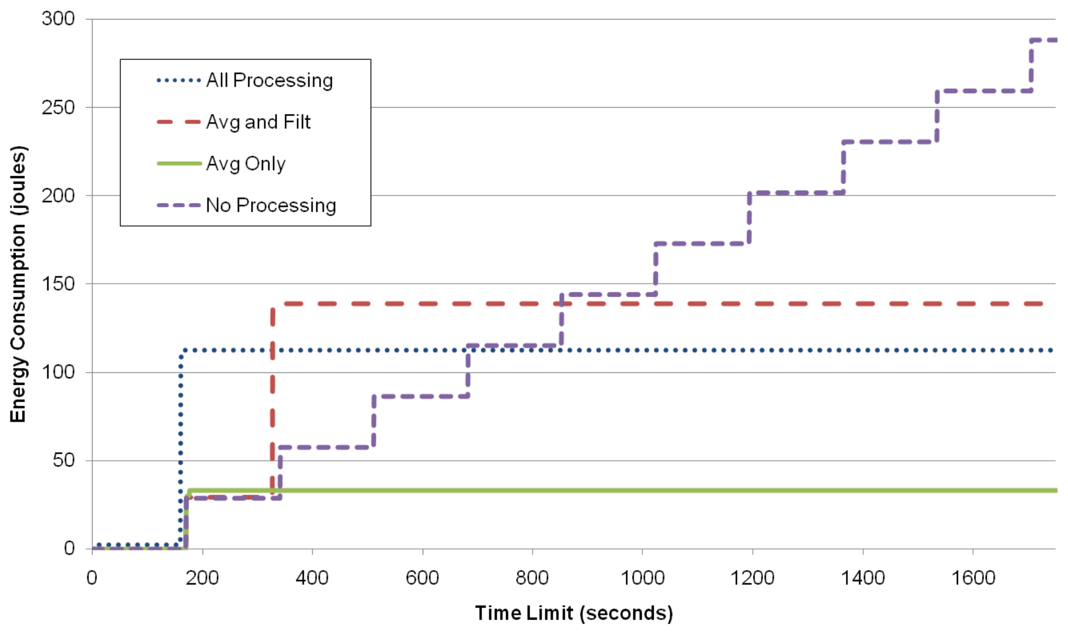


Figure 5.11: TC: Energy Consumption for Various Levels of Processing

energy harvesting rate to be high. Figures 5.10 and 5.11 show the results of these tests. Regardless of the time limit, the *all processing* level always obtains the highest utility, followed by *averaging only* and *averaging and filtering*, depending on the time limit. Performing *all processing* consumes less than half of the energy that *no processing* consumes, emphasizing the benefit of performing on-node data processing. If energy is extremely limited, however, it may be more beneficial to perform *averaging only*, as it consumes the least amount of energy at the cost of lower utility.

5.3.3 Utility Constraint (UC) Algorithm

The utility constraint (UC) algorithm determines the execution time, wait time, and energy consumption of a set of tasks at a specified utility. Similar to the previous algorithms, the utility constraint algorithm is also tested while varying the energy harvesting rate, the task priorities, and the amount of processing performed.

Energy Harvesting

For the first set of tests, the energy harvesting rate is varied to see how the environmental conditions affect the execution characteristics. The available energy in the buffer is set to to 100 joules and the priorities for all tasks to 1.0. The utility constraint algorithm is then executed while varying the desired task utility from 0.01 to 1.00 for the same three distinct times of day used in section 5.3.2: the peak of a sunny day ($\text{Rate}_{\text{EH}} = 0.1022$ mV/s), the peak of a cloudy day ($\text{Rate}_{\text{EH}} = 0.0340$ mV/s), and the middle of the night ($\text{Rate}_{\text{EH}} = 0$ mV/s).

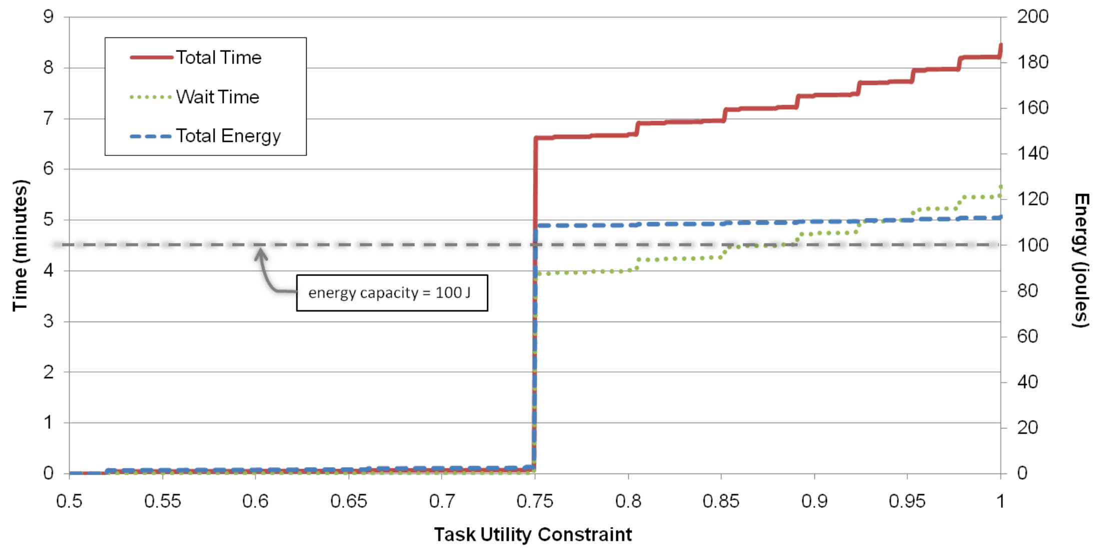


Figure 5.12: UC: Energy and Time Characteristics During A Sunny Day

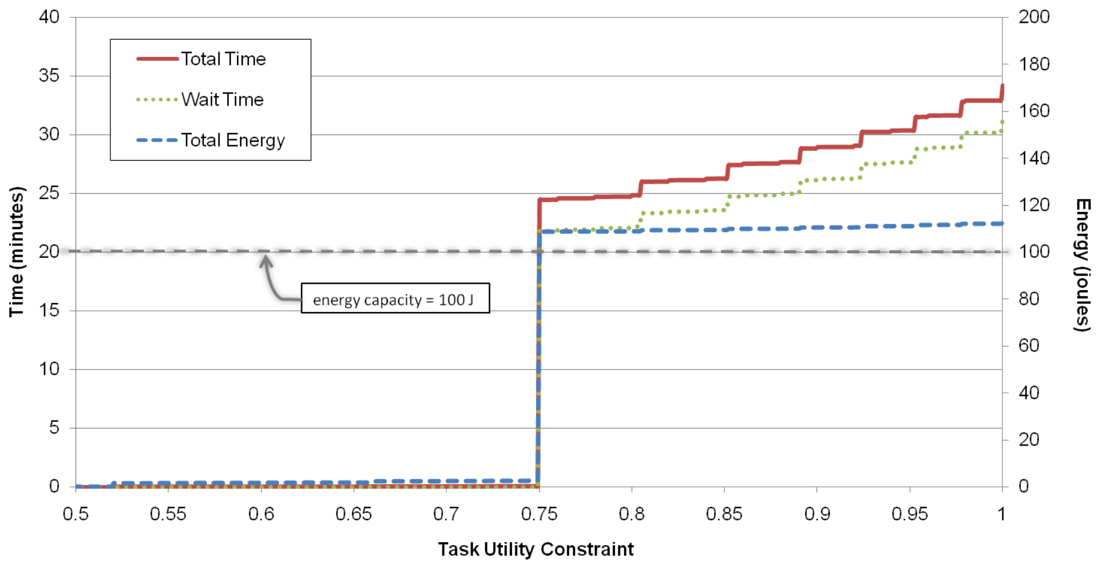


Figure 5.13: UC: Energy and Time Characteristics During A Cloudy Day

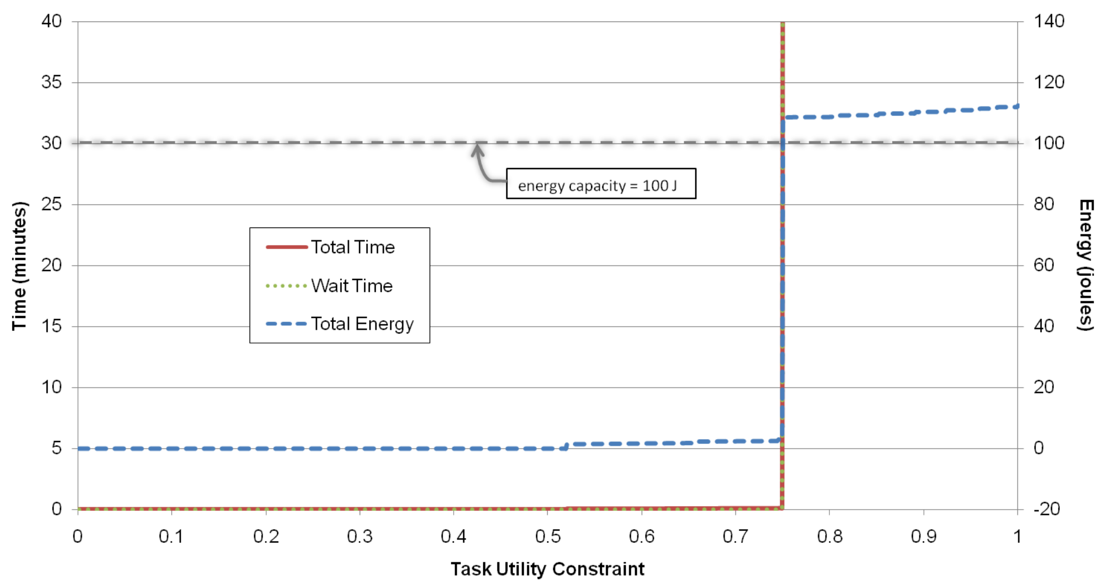


Figure 5.14: UC: Energy and Time Characteristics During During the Night

First, figure 5.12 shows the energy and time required for tasks on a sunny day, when the energy harvesting rate is high. When the energy required to execute the tasks at the desired utility exceeds 100 J, there is no longer sufficient energy in the buffer to satisfy the request. To harvest additional energy, the system must wait for a period of time. Because the experiments in figure 5.12 occur on a sunny day, this wait time never exceeds six minutes. In figure 5.13, however, the wait time is significantly increased (to nearly 35 minutes) due to the reduced energy harvesting rate on this particular cloudy day. The energy and time needed to execute the tasks is equivalent to the energy and time on a sunny day, but because the harvesting rate is lower, the completion of the tasks will require more time. Finally, in figure 5.14 when the energy harvesting rate is very low, the system cannot harvest any additional energy. Once the desired task utilities pass 0.74, no solution is possible, as there does not exist sufficient energy for task execution

(note that this sharp increase in energy consumption is due to the addition of filtering, the most significant processing available.)

Priority

For this specific application, one method to reduce energy consumption is to limit the number of PZT pairs that are evaluated. To demonstrate this, the utility constraint algorithm is evaluated using task graphs with different combinations of priorities representing different numbers of paths. The priority for an active path and all its dependent tasks is set to 1.0, and the priority for the an inactive path and its dependent tasks to 0.0. Tests were then run by increasing the number of active paths from 1 to 120. The results from these tests shown in Figure 5.15 show how the energy consumption and execution time change with an increase in the number of evaluated paths. As expected, the execution time and energy needed will increase approximately linearly as the number of paths increases.

Processing

As with the previous algorithms, the UC algorithm is tested with the the various levels of processing from table 5.4. Figure 5.17 shows the time and energy required of the levels when the available energy is 400 joules, and the energy harvesting rate is high. The execution time for *all processing* is significantly lower than all other levels, due to the dramatic reduction in the amount of data transmitted (24 MB and over 2800 seconds for *no processing* as compared to 180 B and less than 200 seconds for *all processing*)

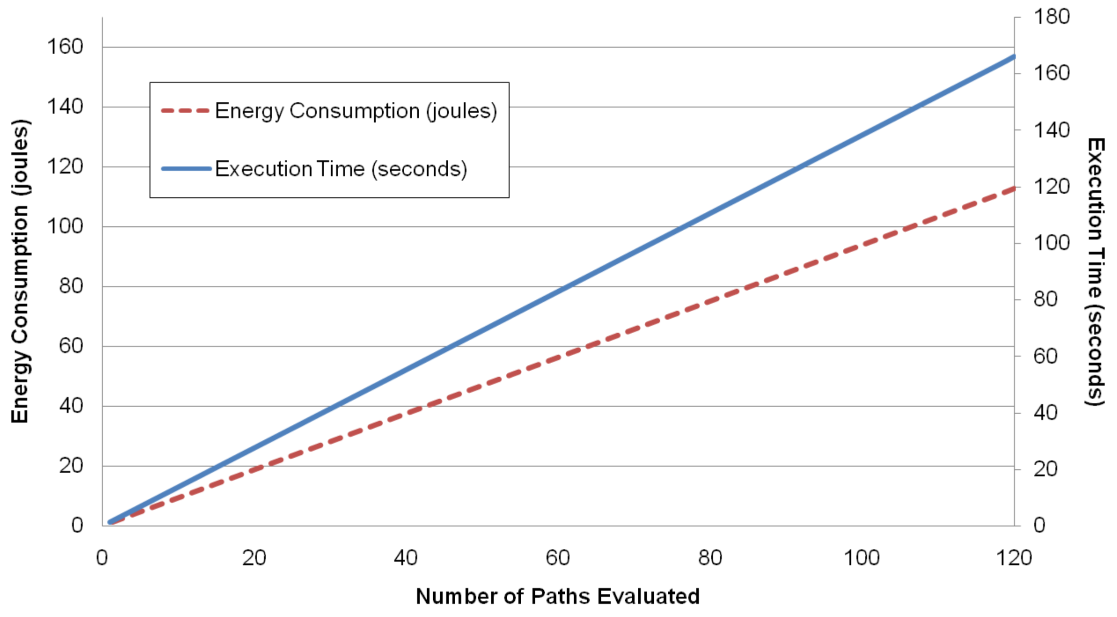


Figure 5.15: UC: Altering the Number of Active Paths

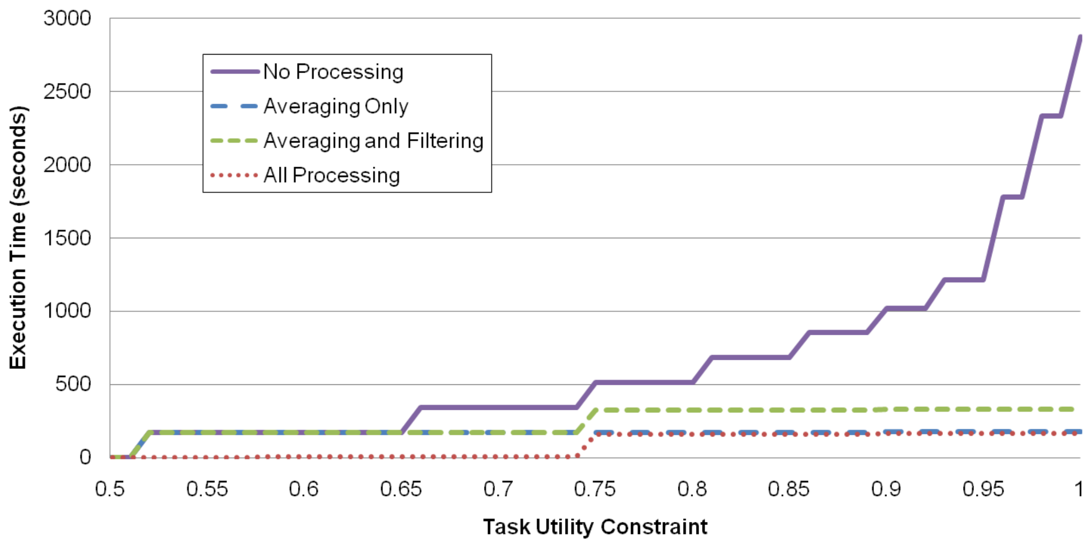


Figure 5.16: UC: Execution Time for Different Levels of Processing

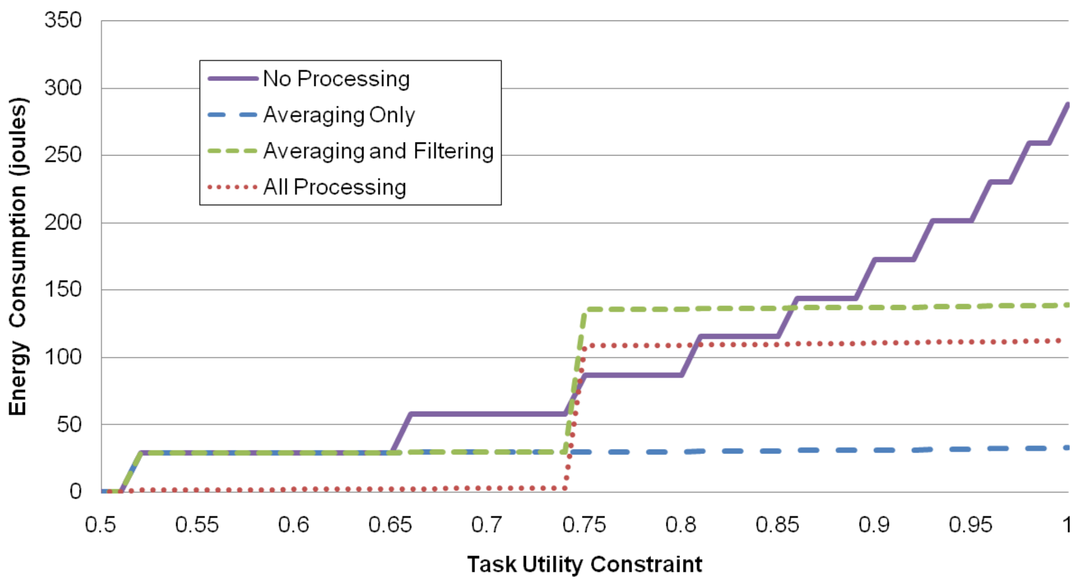


Figure 5.17: UC: Energy Consumption for Different Levels of Processing

and the low data rate of the radio (112.5 Kbps). Unlike execution time, however, the impact of processing on the energy consumption is not as drastic due to the low energy consumption of the chosen radio. Even more, the energy benefits of processing do not occur until the desired utility is high. A specific part of the graph to note is the significant increase in energy consumption for *averaging and filtering*. While filtering is the most complex form of processing (and thus consumes the most time and energy), it does not actually reduce the amount of data.

5.4 Steady State Operation

Steady state execution defines the periodic execution of a task graph on the system. The steady state solution described in section 4.3 uses the energy constraint algo-

Table 5.5: Execution Rate Data: Per Day on Average

Execution Rate	**Utility per Execution**	**Number of Max Utility Executions**	**Total Utility**	**Energy in Buffer**
2	0.63	2.05	1.66	656.74 J
5	0.57	3.8	3.39	658.98 J
10	0.44	4.3	5.00	649.46 J
15	0.37	4.05	6.19	646.20 J
50	0.26	2.35	13.54	642.14
100	0.24	1.2	22.57	636.70 J

rithm to maintain steady state energy E_{steady}. The following sections present the results showing the effect of the execution rate on achievable performance and the impact of the energy profile on the execution rate.

In order to determine the effect of the execution rate on the system energy and task utility, static execution rates ranging from 1 to 100 are tested. Table 5.5 shows the results for rates of 2, 5, 10, 15, 50, and 100. As the execution rate increase, the total utility per day (on average) also increases, while the average utility per execution and the average energy in the buffer decrease. The energy in the buffer, however, still remains above the threshold, E_{steady}. The most interesting measurement is the number of maximum utility executions per day, which actually peaks at 4.55 times per day when the rate $r = 9$, shown in figure 5.18.

While the peak number of maximum utility executions occurs at $r = 9$ for this application and this specific solar panel data sample, it will certainly differ if another application is used and when the weather patterns change. Thus, the tests are run with various initial rates to see how the average rate depends on the starting rate. The results from figure 5.19 show, however, that the execution rate for differing start rates con-

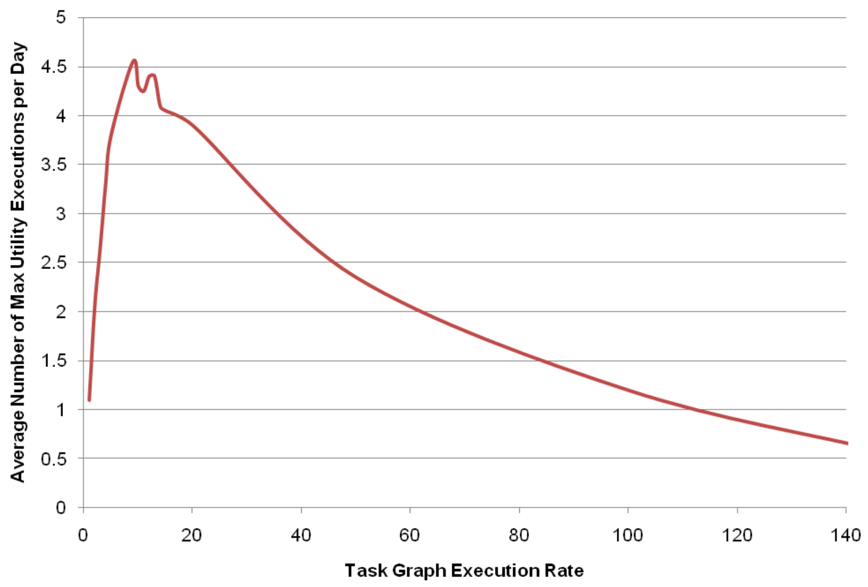


Figure 5.18: The Effect of Execution Rate on Maximum Utility Executions Per Day

verges. Thus, after an initial time period, the starting execution rate does not influence the steady state operation, as it adapts to the energy profile. Specifically, days 4-9 of the solar panel data have a lower harvesting rate than days 10-19 (cloudy vs. sunny days) which is reflected in the increase of the execution rate towards the end of the data evolution. On average, the execution rate on a sunny day increases by 62% as compared to cloudy days, while the average stored energy only increases by 2%. The average stored energy remains approximately constant despite the altering weather conditions. As the weather conditions change, the execution rate is adapted to maintain energy neutrality.

As shown in the tests on the algorithms in the previous section, steady state operation will vary based on the amount of processing performed. To show how processing influences the execution rate, the same levels of processing used in the external trigger state tests are applied to steady state operation. Because *averaging only* requires the

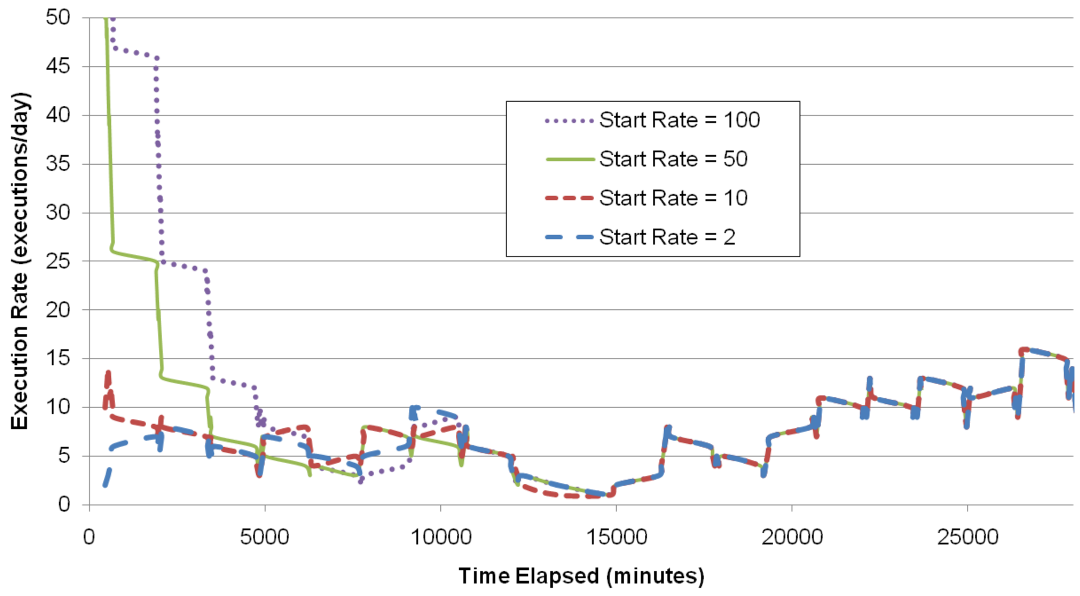


Figure 5.19: Convergence of Execution Rate

least amount of energy, its execution rate is much higher than other levels. Note that the *no processing* level is limited to less than four executions per day on the sunniest day.

5.5 External Trigger State Operation

As steady state execution will also involve external requests at times, steady state operation is supplemented with random external requests ($E_{steady} = 400$ J and $E_{min} = 200$ J). Figures 5.21 and 5.22 show that energy neutrality is still maintained as the execution rate adapts to the energy consumed during external requests. The external requests in the figure are normalized to the primary axis such that a value of 200 indicates a time constraint request and a value of 400 indicates a utility constraint request. In figure 5.21, there exists a 0.1% chance of an external request every minute, hence

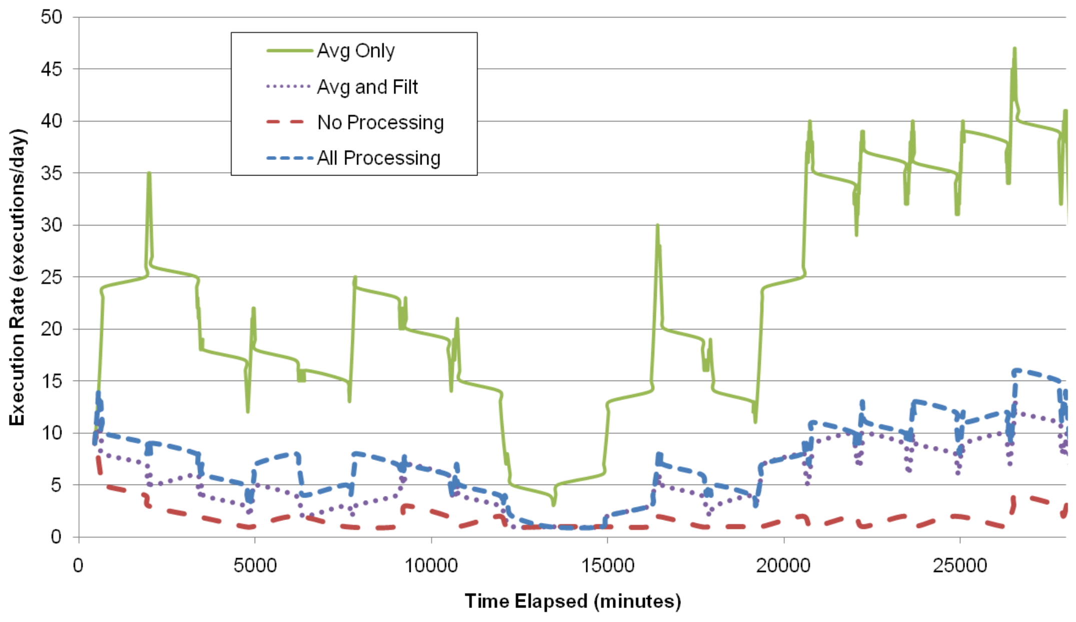


Figure 5.20: Steady State for Levels of Processing

the numerous external requests, while in figure 5.22, there exists a 0.05% chance of an external request every minute but only in the last six days of the evolution. In the second figure, it is clearly evident that the energy remains above E_{steady} during steady state operation and above E_{min} during an external request.

5.6 Summary

This chapter described how the SHiMmer application was adapted to fit the task and energy model by developing an SHM task graph and defining the execution time characteristics based on utility. The method of evaluation and the data used for simulated energy harvesting were outlined. The results for each of the energy and task management algorithms were presented, highlighting the impact of processing, priority,

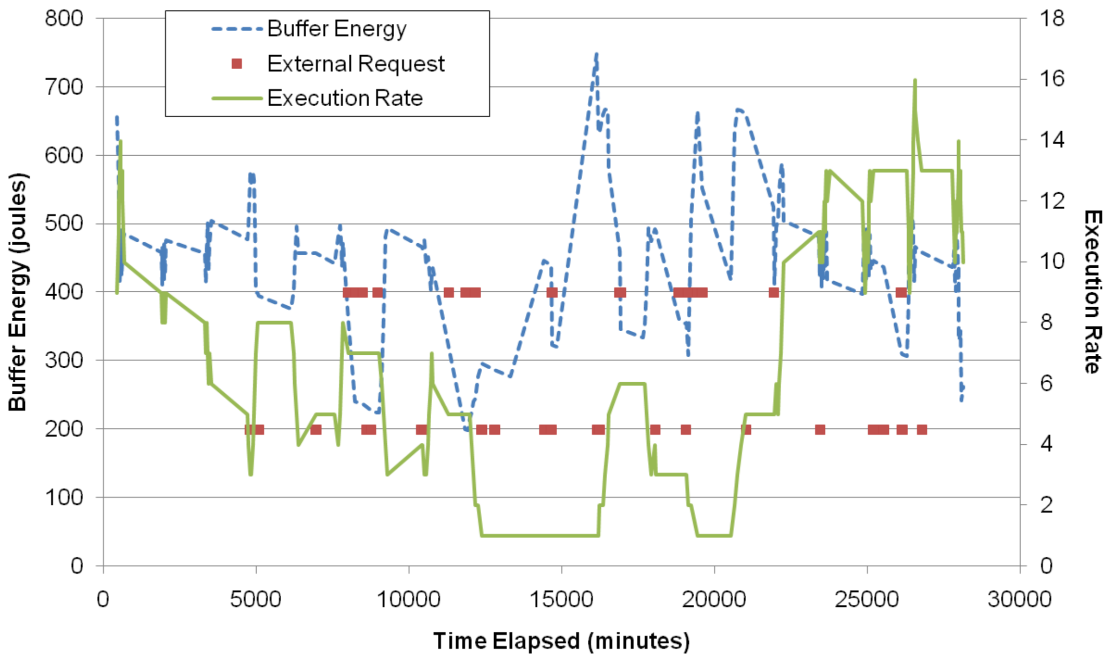


Figure 5.21: Steady State Evolution with Many Random External Requests

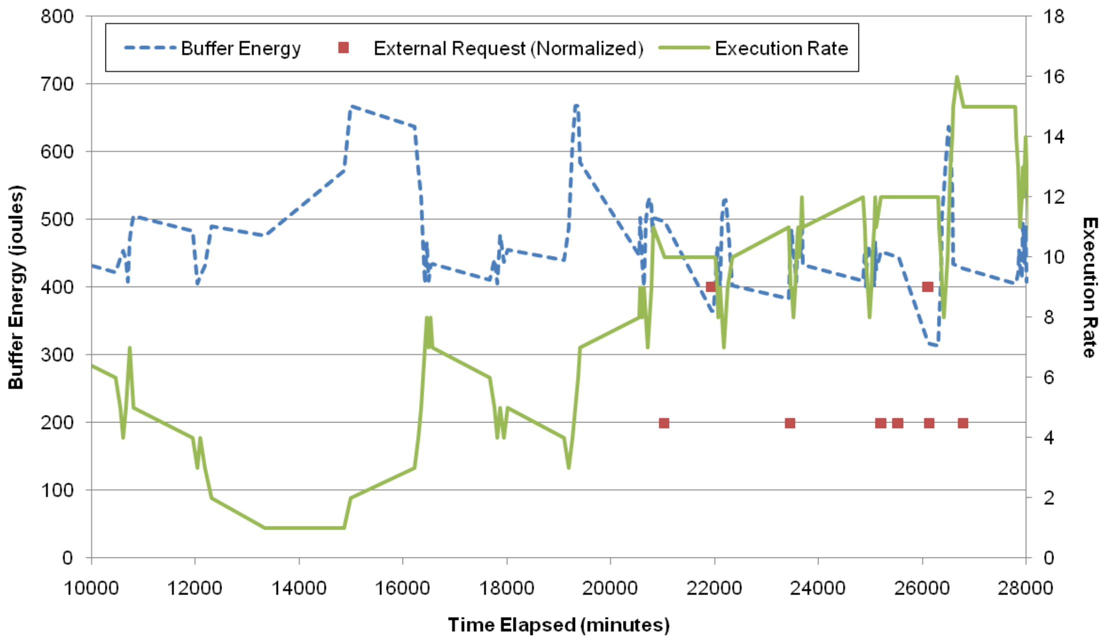


Figure 5.22: Steady State Evolution with Isolated Random External Requests

and the energy profile. Finally, results from both steady state operation and external trigger state were discussed. For steady state operation, the different rates of execution were compared and shown how the execution rate varies as the energy harvesting rate changes. For external trigger state operation, random external requests were initiated, verifying that energy neutrality is maintained while system performance is adapted to satisfy the system and external request constraints.

Chapter 5, in part, has been submitted for publication of the material as it may appear in Networked Sensing Systems, 2009, Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this paper. Chapter 5, in part, is currently being prepared for submission for publication of the material. Steck, Jamie Bradley; Rosing, Tajana Simunic. The thesis author was the primary investigator and author of this material.

Chapter 6

Conclusions

Applications such as structural health monitoring are in need of smart wireless sensor networks that can be deployed and remain physically untouched for long periods of time. Energy harvesting offers exciting possibilities for wireless sensor networks, but requires strict energy and task management in order to maintain energy neutrality (consuming only as much energy as can be harvested) but also satisfy performance requirements.

This thesis addressed energy and task management for energy harvesting wireless sensor networks and applied these methods to a structural health monitoring platform. Chapter 2 summarized relevant research relating to energy harvesting, energy management, and task assignment. Chapter 3 described structural health monitoring techniques, technology, and related systems. Chapter 4 defined the energy and task management problem for a wireless sensing system and presented solutions for steady and external trigger state operation using three application-independent algorithms. Chapter

5 then provided results obtained from evaluation of the algorithms and both modes of operation. This chapter summarizes the contributions of this work and discuss ideas for future work.

6.1 Contributions

In order to maintain energy neutrality while satisfying performance constraints, a system software controller was presented that adapts performance based on energy availability for steady state and external trigger state conditions. Three application-independent, adaptive energy and task management algorithms were presented. The energy constraint algorithm determines the maximum achievable utility using only the current energy available in the system. The time constraint algorithm determines the maximum achievable utility given a time limit. Finally, the utility constraint algorithm determines the expected execution time, harvesting time, and energy consumption given a desired utility.

Steady state operation and external trigger state operation were defined for externally triggered energy harvesting sensing systems. Steady state operation adapts the execution rate based on previous execution results to achieve the highest utility, utilizing the energy constraint algorithm to maintain energy neutrality. External trigger state operation, on the other hand, uses the time and utility constraint algorithms, to provide an external device immediate estimates of the total utility, time, and energy characteristics of a set of tasks.

This energy and task management was then applied to SHiMmer, a wireless platform that combines active sensing and localized processing with energy harvesting to provide long-lived structural health monitoring. Results from this SHM application demonstrated the controller's ability to adapt at runtime and maintain sufficient energy. Steady state results showed that the execution rate changes with varying weather conditions. On average, the execution rate on a sunny day increases by 62% compared to the rate on cloudy days. External trigger state results show that processing significantly affects the efficiency of a structural health monitoring system; specifically, complex processing requires 17 times less execution time and 2.5 times less energy than transmitting raw data.

There are several key benefits to the proposed energy and task management design. First, any application can use the algorithms by specifying a task graph and task execution characteristics. Second, task utility can be adapted online at detailed levels more precise than duty cycles or execution rates. Third, the algorithms ensure that, if possible, all tasks in the task graph will be scheduled, and a task's utility will be determined by its relative priority in the system. Finally, as shown in the results, energy neutrality can be maintained while adapting performance as the energy profile changes.

6.2 Future Work

Managing energy harvesting sensor nodes will continue to be a challenge as hardware advances and WSN demands expand. Energy harvesting aware software is critical to achieve significant performance and should be integrated into WSN Real Time Operating Systems, treated as a resource as essential as memory.

Immediate next steps for the specific work in this thesis is to test these methods on a physical structure in a field test, planned for August of 2009. Additionally, energy harvesting aware WSN routing methods will be needed to extend this model to a small network of energy harvesting nodes. In a network of energy harvesting systems, the energy harvesting rate will vary according to the location and characteristics of the device. Thus, it is desirable to exploit this discrepancy by distributing the routing burden among nodes according to energy harvesting rates.

References

[1] Benson, B., Irturk, A., Cho, J., and Kastner, R., 2008: Survey of hardware platforms for an energy efficient implementation of matching pursuits algorithm for shallow water networks. In *WuWNeT '08: Proceedings of the third ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, 83–86. ACM, New York, NY, USA. ISBN 978-1-60558-185-9. doi:http://doi.acm.org/10.1145/1410107.1410123.

[2] Farrar, C. R., and Worden, K., 2007: An introduction to structural health monitoring. *Philosophical Transactions of the Royal Society A: Physical, Mathematical and Engineering Sciences*, **365**, 303–315. doi:10.1098/rsta.2006.1928.

[3] Flynn, E. B., 2008: Personal interview: Damage identification process.

[4] FreeRTOS, 2009: The freertos.org project. http://www.freertos.org/.

[5] Hsu, J., Zahedi, S., Kansal, A., Srivastava, M., and Raghunathan, V., 2006: Adaptive duty cycling for energy harvesting systems. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, 180–185. ACM, New York, NY, USA. ISBN 1-59593-462-6. doi: http://doi.acm.org/10.1145/1165573.1165616.

[6] Jiang, X., Polastre, J., and Culler, D., 2005: Perpetual environmentally powered sensor networks. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 65. IEEE Press, Piscataway, NJ, USA. ISBN 0-7803-9202-7.

[7] Johnson, S. G., and Frigo, M., 2007: A modified split-radix fft with fewer arithmetic operations. *Signal Processing, IEEE Transactions on*, **55**(1), 111–119. ISSN 1053-587X. doi:10.1109/TSP.2006.882087.

[8] Kansal, A., Hsu, J., Srivastava, M., and Raghunathan, V., 2006: Harvesting aware power management for sensor networks. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, 651–656. ACM, New York, NY, USA. ISBN 1-59593-381-6. doi:http://doi.acm.org/10.1145/1146909.1147075.

[9] Kansal, A., Potter, D., and Srivastava, M. B., 2004: Performance aware tasking for environmentally powered sensor networks. *SIGMETRICS Perform. Eval. Rev.*, **32**(1), 223–234. ISSN 0163-5999. doi:http://doi.acm.org/10.1145/1012888.1005714.

[10] Lynch, J. P., 2007: An overview of wireless structural health monitoring for civil structures. *Philosophical Transactions of the Royal Society A: Physical, Mathematical and Engineering Sciences*, **365**, 345–372. doi:10.1098/rsta.2006.1932.

[11] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J., 2002: Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 88–97. ACM, New York, NY, USA. ISBN 1-58113-589-0. doi:http://doi.acm.org/10.1145/570738.570751.

[12] Maxstream, 2007: *Maxstream XBee OEM RF Module*. Available at http://www.digi.com/products/wireless/point-multipoint/xbee-pro-series1-module.jsp.

[13] Moser, C., Thiele, L., Brunelli, D., and Benini, L., 2007: Adaptive power management in energy harvesting systems. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, 773–778. EDA Consortium, San Jose, CA, USA. ISBN 978-3-9810801-2-4.

[14] Moser, C., Thiele, L., Brunelli, D., and Benini, L., 2007: Lazy scheduling for energy-harvesting sensor nodes. In *DIPES '06: Proceedings of the 5th working conference on distributed and parallel embedded systems*, 125–134. ISBN 978-3-9810801-2-4.

[15] Moser, C., Thiele, L., Brunelli, D., and Benini, L., 2008: Approximate control design for solar driven sensor nodes. In *HSCC '08: Proceedings of the 11th international workshop on Hybrid Systems*, 634–637. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-78928-4. doi:http://dx.doi.org/10.1007/978-3-540-78929-1_52.

[16] Musiani, D., Lin, K., and Rosing, T. S., 2007: Active sensing platform for wireless structural health monitoring. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, 390–399. ACM, New York, NY, USA. ISBN 978-1-59593-638-X. doi:http://doi.acm.org/10.1145/1236360.1236409.

[17] NASA Propulsion Laboratory, California Institute of Technology, 2007: Mars exploration rover mission: Technology. http://marsrovers.nasa.gov/technology/bb_power.html.

[18] Pakzad, S. N., Fenves, G. L., Kim, S., and Culler, D. E., 2008: Design and implementation of scalable wireless sensor network for structural monitoring. *Journal of Infrastructure Systems*, **14**(1), 89–101.

[19] Park, G., Sohn, H., Farrar, C. R., and Inman, D. J., 2003: Overview of piezoelectric impedance-based health monitoring and path forward. *The Shock and Vibration Digest*, **35**(6), 451–463.

[20] Patel, S., Lorincz, K., Hughes, R., Huggins, N., Growdon, J., Welsh, M., and Bonato, P., 2007: Analysis of feature space for monitoring persons with parkinson's disease with application to a wireless wearable sensor system. 6290–6293.

[21] Piorno, J. R., Bergonzini, C., Lee, B., and Rosing, T. S., 2009: Management of solar harvested energy in actuation-based and event-triggered systems. *4th Annual Energy Harvesting Workshop*.

[22] Raghunathan, V., and Chou, P. H., 2006: Design and power management of energy harvesting embedded systems. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, 369–374. ACM, New York, NY, USA. ISBN 1-59593-462-6. doi:http://doi.acm.org/10.1145/1165573.1165663.

[23] Ritter, H., Schiller, J., Voigt, T., Dunkels, A., and Alonso, J., 2004: Solar-aware clustering in wireless sensor networks. In *Proceedings of Ninth IEEE Symposium on Computers and Communications*.

[24] Rosenker, M. V., Hersman, D. A. P., Higgins, K. O., Sumwalt, R. L., and Chealander, S. R., 2008: Collapse of i-35w highway bridge, minneapolis, minnesota, august 1, 2007: Highway accident report ntsb/har-08/03. Technical report, National Transportation Safety Board.

[25] Rusu, C., Melhem, R., and Mossé, D., 2002: Maximizing the system value while satisfying time and energy constraints. In *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, 246. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-1851-6.

[26] Rusu, C., Melhem, R., and Mossé, D., 2005: Multi-version scheduling in rechargeable energy-aware real-time systems. *J. Embedded Comput.*, **1**(2), 271–283. ISSN 1740-4460.

[27] Simunic, T., 2001: *Energy efficient system design and utilization*. Ph.D. thesis, Stanford University, Stanford, CA, USA. Adviser-Micheli,, Giovanni De.

[28] Staszewski, W. J., Lee, B. C., Mallet, L., and Scarpa, F., 2004: Structural health monitoring using scanning laser vibrometry: I. lamb wave sensing. *Smart Material Structures*, **13**, 251–260. doi:10.1088/0964-1726/13/2/002.

[29] Sugihara, R., and Gupta, R. K., 2008: Improving the data delivery latency in sensor networks with controlled mobility. In *DCOSS '08: Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems*, 386–399. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-69169-3. doi:http://dx.doi.org/10.1007/978-3-540-69170-9_26.

[30] Taylor, S. G., Farinholt, K. M., Flynn, E. B., Figueiredo, E., Mascarenas, D. L., Moro, E. A., Park, G., Todd, M. D., and Farrar, C. R., 2009: A mobile-agent based wireless sensing network for structural monitoring applications. *Measurement Science and Technology*, **20**(4), 045201 (14pp).

[31] Texas Instruments, 2007: *TMS320C2811 Data Manual*. Available at http://focus.ti.com/lit/ds/symlink/tms320f2812.pdf.

[32] Tian, Y., Ekici, E., and Ozguner, F., 2005: Energy-constrained task mapping and scheduling in wireless sensor networks. *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, 8 pp.–218. doi:10.1109/MAHSS.2005.1542802.

[33] Voigt, T., Ritter, H., and Schiller, J., 2003: Utilizing solar power in wireless sensor networks. In *LCN '03: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks*, 416. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2037-5.

[34] Xu, N., Rangwala, S., Chintalapudi, K. K., Ganesan, D., Broad, A., Govindan, R., and Estrin, D., 2004: A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 13–24. ACM, New York, NY, USA. ISBN 1-58113-879-2. doi:http://doi.acm.org/10.1145/1031495.1031498.