MCCLUSKY, DOUGLAS R. A Dynamic Utility-Based Energy Manager for Wireless Sensor Networks. (Under the direction of Robert Fornaro.)

Research in wireless sensor networks (WSNs) has created many approaches to conserve scarce energy resources while accomplishing the critical tasks of data collection and message routing. However, it remains difficult for non-specialists to leverage the understanding of WSN application design. One recent area of research focuses on the definition of a re-useable software module to encapsulate energy management concerns. There have been a variety of approaches, but none allows complete flexibility for optimization using application defined utility measures without imposing significant overhead costs by flooding status report messages or collecting all node status information to a central location for decision-making. This paper proposes a middleware, a Dynamic Framework for Energy Management (DFEM), that allows users to describe application utility, and then solves for the global optimal policy for the network using only information from immediate neighbor nodes. The basis for this optimization is a dynamic programming formulation of the sensing and routing problem of an arbitrary network, which is adapted into a distributed algorithm that solves the dynamic programming through policy iteration. In addition, DFEM provides a skills gradient for different target user groups. New sensors and task descriptions can be extended from the library code with only basic programming experience, and deployment and configuration require no special programming or wireless networking knowledge. Extensions of the basic DFEM are investigated to relax the lifetime constraint description, to allow user queries to modify the application description, and to include low-latency reporting of rare events as a network responsibility. Both simulation and physical deployment results are discussed and compared.
A Dynamic Utility-Based Energy Manager for Wireless Sensor Networks

by
Douglas R. McClusky

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2014

APPROVED BY:

_________________________  _________________________
Donald Bitzer            Mihail Sichitiu

_________________________  _________________________
Wenyé Wang               Kazufumi Ito

_________________________  _________________________
Suzanne Kennedy-Stoskopf Robert Fornaro
Chair of Advisory Committee
BIOGRAPHY

The author is a thoroughly uninteresting figure.
ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr Robert Fornaro, whose guidance has formed the basis for my research and teaching styles, whose deep insight is always anchored in simple explanations of fundamental concepts and their pragmatic consequences, and without whom I would not have been able to finish.

Margaret Heil, whose counsel many times kept me sane and grounded through the valley of the shadow of Grad School.

Marhn Fullmer, who is a kindred spirit, a compatriot in the never-ending battle to avert on-coming chaos, and an ever-reliable source of practical advice.

My amazing wife, Sofie, whose intrepid nature, meticulous planning and unshakable optimism merit a lifetime of further study.

And my parents, who have always supported me and tried to understand me, even when I had trouble explaining my own thoughts.
# TABLE OF CONTENTS

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Chapter 1 Introduction</th>
<th>.....</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Introduction</td>
<td>.....</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Literature Review</td>
<td>.....</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 Energy</td>
<td>.....</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Middleware</td>
<td>.....</td>
<td>5</td>
</tr>
</tbody>
</table>

## Chapter 2 Dynamic Distributed Optimization

| 2.1 Motivation for Nested Interfaces | ..... | 7 |
| 2.2 Motivation for Dynamic Optimization | ..... | 8 |
| 2.3 Optimization Problem Formulation | ..... | 10 |
| 2.3.1 Definitions | ..... | 10 |
| 2.3.2 Optimization Problem | ..... | 12 |
| 2.4 Distributed Algorithm | ..... | 13 |
| 2.4.1 Dynamic Optimization | ..... | 13 |
| 2.4.2 Configuration and Extensions | ..... | 14 |
| 2.5 Test Implementations | ..... | 15 |
| 2.5.1 Description | ..... | 15 |
| 2.5.2 Results | ..... | 16 |
| 2.5.3 Limitations | ..... | 24 |
| 2.5.4 Simulation-Prototype Comparison | ..... | 25 |

## Chapter 3 Extensions on the Base Algorithm

| 3.1 Optimal Lifetime with Utility Inflation | ..... | 28 |
| 3.1.1 Test Cases | ..... | 33 |
| 3.2 Rare Event Reporting | ..... | 37 |
| 3.2.1 Test Cases | ..... | 38 |
| 3.3 Run Time User Input | ..... | 43 |
| 3.3.1 Test Cases | ..... | 44 |
| 3.4 Conclusion | ..... | 44 |
| 3.5 Future Work | ..... | 47 |
| 3.5.1 Heterogeneous Networks | ..... | 47 |
| 3.5.2 More Complex Network Topologies | ..... | 48 |
| 3.5.3 Empirical utility expressions | ..... | 49 |

## References

| References | ..... | 51 |

## Appendix

<p>| Appendix A Full-Page Graphs | ..... | 56 |
| A.1 Base Algorithm Simulations | ..... | 57 |
| A.1.1 Data Generated Per Node under DFEM and EMA | ..... | 57 |
| A.1.2 Tasks Scheduled at Each Node under DFEM | ..... | 60 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Simulated Topology</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Defining Subsets and Indices of a Network of Nodes</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Data Generated Per Node under DFEM and EMA</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Tasks Scheduled at Each Node under DFEM</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Tasks Scheduled at Each Node under EMA</td>
<td>19</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>31-Node Topology</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Data Generated Per Node under DFEM and EMA</td>
<td>21</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>Tasks Scheduled at Each Node under DFEM</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>Tasks Scheduled at Each Node under EMA</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>31-Node Topology and Test Network</td>
<td>25</td>
</tr>
<tr>
<td>Figure 2.11</td>
<td>Data Generated Per Node in Simulation and Test Deployment</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2.12</td>
<td>Tasks Scheduled at Each Node under DFEM</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Parental Lifetime Water-Filling Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Target Lifetime at Each Node (5-Node Topology)</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Tasks Scheduled under Dynamic Lifetime Constraint (5-Node Topology)</td>
<td>34</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Target Lifetime at Each Node (31-Node Topology)</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Tasks Scheduled under Dynamic Lifetime Constraint (31-Node Topology)</td>
<td>36</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Tasks Scheduled with Rare Event Reporting (5-Node Topology)</td>
<td>39</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Report Latency at Each Node (5-Node Topology)</td>
<td>40</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Tasks Scheduled with Rare Event Reporting (31-Node Topology)</td>
<td>41</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Report Latency at Each Node (31-Node Topology)</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Tasks Scheduled at Each Node with User Update</td>
<td>45</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Tasks Scheduled at Each Node with User Update</td>
<td>46</td>
</tr>
<tr>
<td>Figure A.1</td>
<td>Data Generated Per Node under DFEM</td>
<td>58</td>
</tr>
<tr>
<td>Figure A.2</td>
<td>Data Generated Per Node under EMA</td>
<td>59</td>
</tr>
<tr>
<td>Figure A.3</td>
<td>Tasks Scheduled at Node 0 under DFEM</td>
<td>60</td>
</tr>
<tr>
<td>Figure A.4</td>
<td>Tasks Scheduled at Node 1 under DFEM</td>
<td>61</td>
</tr>
<tr>
<td>Figure A.5</td>
<td>Tasks Scheduled at Node 2 under DFEM</td>
<td>62</td>
</tr>
<tr>
<td>Figure A.6</td>
<td>Tasks Scheduled at Node 3 under DFEM</td>
<td>63</td>
</tr>
<tr>
<td>Figure A.7</td>
<td>Tasks Scheduled at Node 4 under DFEM</td>
<td>64</td>
</tr>
<tr>
<td>Figure A.8</td>
<td>Tasks Scheduled at Node 0 under EMA</td>
<td>65</td>
</tr>
<tr>
<td>Figure A.9</td>
<td>Tasks Scheduled at Node 1 under EMA</td>
<td>66</td>
</tr>
<tr>
<td>Figure A.10</td>
<td>Tasks Scheduled at Node 2 under EMA</td>
<td>67</td>
</tr>
<tr>
<td>Figure A.11</td>
<td>Tasks Scheduled at Node 3 under EMA</td>
<td>68</td>
</tr>
<tr>
<td>Figure A.12</td>
<td>Tasks Scheduled at Node 4 under EMA</td>
<td>69</td>
</tr>
<tr>
<td>Figure A.13</td>
<td>Data Generated Per Node under DFEM</td>
<td>70</td>
</tr>
<tr>
<td>Figure A.14</td>
<td>Data Generated Per Node under EMA</td>
<td>71</td>
</tr>
<tr>
<td>Figure A.15</td>
<td>Tasks Scheduled at Node 0 under DFEM</td>
<td>72</td>
</tr>
<tr>
<td>Figure A.16</td>
<td>Tasks Scheduled at Node 1 under DFEM</td>
<td>73</td>
</tr>
<tr>
<td>Figure A.17</td>
<td>Tasks Scheduled at Node 2 under DFEM</td>
<td>74</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Introduction

Wireless sensor networks (WSNs) are composed of small, radio-enabled computers with limited energy reserves stored in batteries or capacitors. WSN nodes can use sensors to collect data, and can pass that data via radio through the network to some desired destination. As the capabilities of WSN nodes expand, the demand for WSN applications continues to grow. Advances in understanding of WSN energy harvesting, measurement and planning have increased the potential longevity of WSN deployments. As specialists’ understanding of WSNs increase and WSN capabilities improve, the potential to leverage WSN capabilities by non-specialist users has driven research into creating software mechanisms to simplify the programming, configuration and deployment process.

Energy management has always been a key factor in the design of WSN applications. The utility of many potential applications of WSNs depends on the duration a WSN can to continue to function. The most immediate limitation to WSN longevity is their limited energy storage capacity. To minimize the use of critical energy resources, each component of a WSN application is designed for energy efficiency. Nodes save energy by turning off unused hardware, especially their radios. However, a node with its radio off can neither send nor receive any messages. WSN applications therefore create mechanisms to get adjacent nodes to turn on their radios and communicate within a shared period of time, and then to turn off their radios to save energy. WSN applications also set up paths for nodes to pass messages (routing), which avoid overtaxing any one node as much as possible. Even when efficiently used, however, batteries still hold a finite amount of energy. Many current WSN systems further increase their potential deployment time by harvesting additional energy from their environments. Solar panels, piezoelectric crystals and other energy harvesting hardware options allow WSN nodes to recover energy periodically, depending on the conditions in their environment. Using energy harvesting at each node has
expanded the options for energy management in WSNs. Operating under a harvested energy constraint, a WSN application can function until it suffers hardware failure. New energy storage hardware strategies and environmental hardening of nodes to prevent weather damage also increase the longevity of WSN deployments.

The ability to monitor phenomena of interest with WSNs can greatly reduce the cost of collecting data for empirical studies in a wide range of fields. However, scientists with the training necessary to pursue research in their own field often have limited time to study WSNs application design and configuration. Middleware bridge the gap between the state-of-the-art understanding of energy management, scheduling and routing in WSN applications and the level of expertise actually required by users. Middleware automate aspects of design and configuration that can feasibly be automated, provide structure for required user decisions, and simplify the ways users must convey those decisions.

This paper proposes a middleware, a Dynamic Framework for Energy Management (DFEM), that allows users to describe potential sensor reading tasks for a network in terms of their size in bytes and application utility for the user. Using the user-provided information and measures of energy available at each node within the network, DFEM uses a distributed algorithm to adapt network behavior towards the optimum schedule of sensing and data routing at each node. By seeking to optimize both sensing and routing behavior using a distributed algorithm, DFEM joins a small group of relatively new middleware. The advantage of automating sensor schedule decisions as well as routing decisions is the reduced complexity of decisions made by the user. Compared to previous distributed approaches to optimize both sensor scheduling and routing, DFEM will better balance competing application priorities. The utility measurements provided by users allow DFEM to compare competing behavior options in a more nuanced fashion than the simple priority hierarchy.

1.2 Literature Review

The energy research can be divided into papers on energy harvesting approaches, measuring energy harvested and used by a node, making energy use decisions in general within a harvesting energy constraint, and specifically making wireless radio route decisions within an energy constraint. The middleware design literature includes meta-languages designed to simplify the process of coding WSN nodes and code libraries that provide reusable functionality through simple Application Programming Interfaces (APIs).
1.2.1 Energy

Energy Harvesting

Because energy is so valuable to Wireless Sensor Networks (WSNs), a wide array of energy harvesting mechanisms have been explored for WSN deployment. Hardware has been developed to harvest solar, thermal and vibration energy cheaply and in small enough packages for WSN nodes[35]. There has also been a progression of approaches to store harvested energy, beginning with rechargeable batteries[32]. Because the amount of energy available for harvest often varies, super capacitors provided more charge cycles and therefore greater longevity[43]. Combining the two approaches, with the super capacitor acting as the primary energy reservoir and a rechargeable battery as a secondary energy reservoir, allows the super capacitor to smooth much of the harvested energy volatility without sacrificing long-term capacity or greatly increasing the cost of nodes[21]. The combined approach to hardware has been further refined and combined with environmental hardening and both hardware and software failsafe measures to allow even longer term deployments[14].

Energy Measurement

Work has been done to understand the problem of measuring energy use in current WSN platforms[16][20]. Fine-grain measurement is important because it allows more detailed planning. The energy cost of monitoring energy use is also important. Techniques such as voltage to frequency conversion allow arbitrary resolution cheaply, and can also be used to measure the collection and storage of harvested energy, including the leakage from any super capacitors used for storage[17][48]. Energy measurement forms the basis for modular systems to profile energy harvesting and energy use, even across multiple harvesting hardware components and multiple energy storage components[17][27][28][22]. For each harvesting component, an energy monitor determines and reports the expected energy production and upper and lower bounds, which can be used to characterize a sustainable duty cycle as discussed in Kansal et al[25].

Scheduling With Energy Constraints

Task Scheduling

As platforms for environmental harvesting of energy at sensor nodes become more available (e.g., solar panels or piezoelectric crystals), wireless sensor network applications have begun to employ nodes that change their behavior based on the availability of harvested energy[43][14][46][2][31]. The basic approach described in these papers can be understood by thinking of energy use as a single duty cycle that is increased or decreased based on energy harvesting results. One line of research formulates a long term harvested energy constraint for wireless sensor
nodes and thereby provides a means to plan a sustainable rate of energy use. This formulation states that a node can work indefinitely if its average energy use is less than its average energy harvest \((\rho)\) as long as its energy reservoirs are large enough, in terms of the bounds on energy use and harvesting [25]. Alternatively, the sustainable rate can be described in terms of the time required to recover through energy harvesting a given amount of energy use [30]. Nodes can then adjust their behavior based on the maximum sustainable rate and available routing capacity [22].

**Routing**

Radio communication within the network can be divided into Media Access Control (MAC) layer and routing layer decisions. The simplest MAC protocol is carrier sense multiple access (CSMA). When nodes are about to transmit, they detect whether any node within range is transmitting on a given channel. A node waits until it detects no other signals, then it transmits its message. Another protocol that requires no schedule uses Request to Send and Clear to Send messages (RTS / CTS) to avoid the hidden terminal problem found in CSMA. Time division multiple access (TDMA) protocols use schedules to prevent collisions between nodes. Generally, during the schedule creation process, nodes compete with CSMA, but subsequent communication takes place using the schedules and is collision free. TDMA protocols trade the overhead of setting up a schedule for the overhead of sending RTS / CTS packets or of packet collisions. S-MAC, B-MAC and Z-MAC are all instances of TDMA-based MAC protocols [11][40]. Alternatively, a combination of CSMA and TDMA can be used. Z-MAC combines TDMA with allowing nodes to use CSMA when an empty schedule slot is detected, and funnel-MAC uses CSMA in sparser areas of the network [1][40].

The organization and functioning of a wireless network can be divided into a configuration/reconfiguration phase, in which the network establishes its routes and schedules, and a steady-state phase, in which the functions of the network do not change. Generally a network spends most of its time in the steady-state phase. The schedules of a sensor network can be understood as arrays of actions and arrays of corresponding action-times. Examples of sensor node actions are to sample sensors and to transmit or to receive sensor data over radio. A node sleeps when it is not scheduled to make an action [42].

Routing decisions occur during the configuration/reconfiguration phase of a self-organizing network. Nodes find routes for the data they are scheduled to generate in three steps. First, nodes are given addresses; then node addresses are disseminated some distance through the network; finally nodes discover routes available to them. Discovery can be done proactively, on-demand, as need arises, or a hybrid of proactive and on-demand routing can be used [3].

Much of the early energy-constrained routing literature describe ways to accommodate a certain sensor data load for the longest possible time until network partition. One approach is by routing through the nodes with the greatest remaining energy [9]. In a harvesting system,
the greatest remaining energy approach can be generalized to include projected energy gains at regular intervals[47] or to route data instead through the nodes that will require the least time to recover the energy used for routing[33]. Nodes can also simply try to route whatever sensor data is available until they reach their maximum sustainable duty cycle, making routing decisions based on the resulting latency[22][25]. Routing by sustainable latency has been used many times in planning sustainable energy policies[24][37][21]. Other areas of routing literature seek to minimize the latency of user-generated requests for data[10][36][4], or the latency of reporting rare environmental events [39][18].

**Task and Routing Scheduling**

Solving the optimization problems of sensor task scheduling and routing simultaneously poses a greater problem than solving for each, separately, because the sensing and routing behavior of nodes influence the decisions available to other nodes. Some solutions to the joint optimization problem use a central control approach [23][15]. These systems collect information from the network to a central location, solve the joint optimization problem and distribute the chosen policy to the network nodes. Another line of investigation seeks distributed solutions to the joint optimization problem. One solution to the joint problem is to consider a network problem with virtual links within each sensor node leading to their sensors. Routing optimization approaches, based on a preferred routing metric can then be applied [7]. One drawback to the routing approach is that it is nontrivial to solve when there is more than one type of data, possibly with different utility to application programmers. A third line of investigation floods energy states in and routing costs to nodes within the network, and has each node solve a local application utility optimization problem independently[8]. A group of systems called energy architectures use priority schemes to allow nodes to determine their behavior [27][28][22]. Priority schemes can also be fine-tuned to change the priorities of behaviors based on heuristic measurements of the system’s state [38].

### 1.2.2 Middleware

For non-specialist users to apply all the advances in WSN research towards investigating their own scientific problems, middleware are necessary to simplify the process of configuring and deploying a WSN. As understanding about the fundamental problems of WSN design has expanded, work has also progressed on creating middleware that can provide as much design power as possible within simple frameworks. One approach to providing a simple framework is to design a simplified language, which can be compiled into an existing platform’s code[12][44][5]. Alternatively, a middleware can just be a library of code to handle complicated configuration decisions with an accessible API[13][19][27][28][22].

The sustainable energy constraint and its extensions discussed in the scheduling and routi-
ing subsections have been incorporated into software engineering frameworks such as the Unified radio Power Management Architecture (UPMA)[27][28], Energy Management Architecture (EMA)[22] and the energy Decision Engine[38]. Energy architectures formalize the division of the energy management component of WSN application design into two distinct functional parts: an energy manager and an energy monitor. The energy monitor utilizes hardware to track the contribution of energy harvesting at application run time as well as energy depletion. The energy manager regulates node behavior according to a policy defined by the application designer and input from the energy monitor. This energy management framework communicates with the application through an API. UPMA focuses on the problem of accommodating tasks from multiple applications and aggregating their individual duty cycles into one coordinated schedule. EMA attempts to maximize the contribution of each node according to the policy provided by application designers, and to ensure predictable degeneration when the policy’s constraints cannot be met. An energy architecture’s user policy is a set of constraints, listed in order of priority with one objective to optimize. A request/grant API allows dynamic task allocation according to the static priority system described by the user policy.
Chapter 2

Dynamic Distributed Optimization

2.1 Motivation for Nested Interfaces

Part of the motivation for DFEM has been an ongoing project with a non-WSN-specialist target user base. As a joint project with the NC State College of Veterinary Medicine and the NC State Department of Forestry and Environmental Resources, an animal tracking application is being developed to monitor animals. Specifically, the network will be used to determine the proximity between animals and the duration of their interactions. In addition, the network will monitor the proximity and duration of each type of animal’s interactions with spot resources that they might share, as well as environmental variables like light and temperature. The collected information can inform disease transmission models.

The animal tracking application has several sensors each node may use to gather data. Data is periodically sampled and regularly reported, instead of request driven or rare events. Nodes are dense enough to allow multiple routing paths, but not so dense that there is much redundancy between nodes. Once the data is collected, it must be relayed via radio across several other nodes to reach a destination that can post the data to a database. The geographical information systems software, ArcGIS, will be used to generate maps and models of the data collected. Since data modeling occurs after the data has all been collected, latency is less important than the amount and quality of the data collected. The combined optimization problem of both deciding what data to collect where and how to route that data is complex, because the decision for one node to spend energy sensing data depends on the decisions of each routing node to spend energy routing the data. Solving the optimization problem should be automated to reduce the complexity of new deployments. Also, the framework for describing new types of data and new sensors should be simple enough for the target user base to implement changes.
The animal tracking application development process highlighted two major requirements of any WSN application middleware that could support the type of applications this user base would want to create. First, the middleware needs to make it simple to add new task descriptions and support for new sensors. Wildlife graduate students often hire undergraduate computer science students for their programming needs, so a middleware that is accessible to undergraduate programmers would be a good starting goal. Next, the middleware should be configurable and manipulatable by the wildlife students, themselves. An intuitive way to describe application goals and their relative importance is required to make configuration simple for non-programmers.

Components of the proposed middleware build on three previously published results. The first is the early papers deriving energy constraints for energy harvesting WSN nodes to function indefinitely. The second is the energy monitor component of energy management architectures that tracks energy use and harvesting to facilitate scheduling decisions. The third component is a previously published routing algorithm. The proposed framework and middleware integrates mechanisms from both these results with dynamic measures of utility to solve the scheduling and routing problem for the whole network as an optimization problem, maximizing utility while maintaining resource constraints. As discussed in more detail later, utility is defined by users in the application layer through an API.

2.2 Motivation for Dynamic Optimization

While on the one hand, centralized solutions to a network-wide optimization problem suffer from scalability issues, due to the messages passed from all nodes to the central decision maker, distributed solutions that rely on priority-based decision-making only are not capable of expressing tradeoffs between competing application goals, and can be prone to unintended pathological behavior when configured by naive users. For example, in the simulated case explored by [22], there is no direct link between communication rate and data generation rate within a given node. The communication rate for the simulated node is provided exogenously and is constant in all but one of their simulated cases. For that case, it is increased halfway through the simulated time period to illustrate how the node reacts to such an externally created change. The assumption of a constant communication rate is problematic for many applications. When a sensor node reports all data recorded or uses a compression technique, the radio load is proportional to the number of sensor readings taken, which is likely to vary. When radio load varies, the strict preference of communication over sensing can lead to very limited performance in some nodes. To illustrate the problems that might occur, consider the network of six nodes in a linear topology in Fig. 2.1. The simulated nodes have a fixed battery capacity and known energy costs for sensing and for communicating through radio. EMA’s fixed priority user policy
approach results in the node farthest from the base station (Node 1) behaving as predicted by the EMA paper. However, nodes closer to the base station must dedicate most of their entire energy budgets to forwarding Node 1’s packets and thus must stop sensing, sacrificing its lowest priority task completely.

A priority hierarchy accomplishes its purpose of predictable performance degradation. However, the routing repercussions of nodes’ scheduling decisions may cause performance to degrade in ways that do not fit the intentions of the application designers. Ideally, there should be trade-offs between competing application goals, without one goal taking complete precedence over the other. A system that could dynamically balance competing goals would allow performance to degrade without completely sacrificing any goal. Current network-wide optimization approaches allow a better balance between competing goals using measures of application utility supplied by the application programmers, but the radio traffic required to collect node status to a central location or to flood node status to all network nodes makes scaling these approaches difficult.

This paper will outline a Dynamic Framework for Energy Management (DFEM) that solves the distributed sensor use and data routing scheduling problem and allows tradeoffs between application objectives by generating task priorities dynamically from application objectives for each potential activation of its tasks. Since local data generating tasks and routing data from neighbors no longer each have a constant priority, application layer scheduling decisions and routing layer scheduling decisions are interdependent. At any decision point, a node may prefer one or the other depending on its situation. DFEM will therefore also manage routing decisions at each of the nodes, combining an on-demand routing protocol and proactive task scheduling. Since DFEM nodes only use information available at their immediate neighbors, DFEM has less overhead and is more scalable than joint optimization approaches that flood node state information or report it to a central location.
2.3 Optimization Problem Formulation

2.3.1 Definitions

Consider data generation and delivery tasks, which make up a large part of wireless sensor network applications. Let data be defined as an array of bytes. There are two functions defined over data.

Definition 1: $V: data \rightarrow \mathbb{R}$ is the application-defined utility expression of the data.
Definition 2: $S: data \rightarrow \mathbb{N}$ is the size in bytes of the data.

For any data a and b, if the data $c = a + b$, $V(c) = V(a) + V(b)$ and $S(c) = S(a) + S(b)$. The application utility of data is 0 unless it is delivered to its destination.

The size and utility of data generated at a node is determined by its data generating tasks in the application layer. At design time, a programmer determines the utility expression for each data generation task in the application. A data generation task could assign utility to the data it generates by directly measuring the data itself, for instance the difference between successive readings. Alternatively, a task could assign utility solely on other variables, such as time since the last measurement. In either case, the utility added by generated data depends on that data reaching their destination. Therefore, nodes must also dedicate some of their resources to passing the data from their neighbors for the network to be effective. Communication tasks, in the routing layer, move data to their destinations. The utility of a communication task is defined as the aggregate utility of the data being delivered.

For example, an application that is taking temperature readings might generate data with a size of 8 bytes, and might assign utility according to the time between successive readings. Generally, sensing more frequently generates more information about the temperature, but sensing too quickly will generate many duplicate values. In the opposite extreme case, if the time between successive sensor readings is very large, important information may be lost. A utility expression that reflects these two ideas would have 0 or near-0 utility as time between measures approaches 0 and would be concave, such as some constant times the square root of time between readings. At design time, the utility expression of each data generation task would be provided through an API. The application designer chooses a measure of how effectively a task meets its application goals in terms of relevant environmental and logical variables available during run time.

The objective for optimization is the utility of the data generated. Constraints on optimization are expressed in terms of the resources available to the nodes. The resources a node needs to accomplish its data delivery goals are its own energy reserve, access to a communication medium, and network bandwidth allocated to forward its communication to its destination,
which is a proxy for the energy dedicated by other nodes to supporting it. There are two sources of data for any given node, each of which depend on the scheduling policy chosen, $u$, which is arbitrary but fixed. The data generated by sensors at node $i$ over time period $T$ is $s_{i,T}(u)$. The data sent across a radio connection from node $i$ to node $j$ over the time period $T$ is $c_{i,j,T}(u)$. Nodes also have a certain amount of energy in their batteries. $E_i$ is a measure of remaining energy at any given point in time and $L_i$ the target remaining lifetime of node $i$ at the same time. $\Delta E_{i,T}$ is the change in energy at node $i$ over time period $T$.

To simplify notation in the following equations, recursively index the nodes of a network such that $N_{q+1}$ is the set of all nodes in $N_q$ and all nodes which can communicate directly with nodes in $N_q$, i.e. neighbors of $N_q$. $N_0$ is the set of all base stations. Let another sequence of sets $C_q = N_q - N_{q-1}$ for each $q$. Now, let the nodes of the network be indexed such that

**Definition 3:** The base station has index 0. If a node with index $k$ belongs to the ring $C_q$, the node with index $k+1$ must belong to either $C_q$ or $C_{q+1}$.

For example, consider the simple linear topology shown in Fig. 2.1. $C_0$ would contain the base station. $C_1$ would contain only Node 5. $C_2$ would contain only Node 4, etc. Since there is only one node in each ring, Node 5 would have index 1, Node 4 would have index 2. Node 1, Node 2 and Node 3 would have indices 5, 4 and 3, respectively. If instead of a linear topology, the connectivity between nodes were as in Fig. 2.2, the rings C would be as shown and one possible indexing of the nodes would be the numbers in the Figure.

In reality, it may be sub-optimal to constrain nodes to only relay data by the most direct route. However, this constraint can be loosened simply by relaxing the assumption that each
physical node is only one node in the graph. For example, if there were a connection between nodes 15 and 16 in Fig. 2.2, Node 11 could also route through Node 16. This would be represented by creating a new Node 17 in the graph, with energy reserves equal to the residual left over from Node 11’s schedule. Node 17 and Node 11 are actually the same physical node, with its scheduling solution procedure divided into two parts.

### 2.3.2 Optimization Problem

The objective of a utility-based energy framework is to maximize the utility delivered by the network subject to available energy and bandwidth constraints over some given time interval, T. The total utility is the sum of the utilities of sensor data at all nodes over T. To express this optimization as a dynamic programming problem, let the function

\[ J_{u,T}(k) = V(s_{k,T}(u)) + \sum_{i>k} V(c_{i,k,T}(u)) \]

and let

\[ J_{u,T}(0) = \sum_{j \in N_0} J_{u,T}(j) \]  

Maximum utility is derived from the network by maximizing \( J_{u,T}(0) \) subject to the two constraints:

\[ \sum_{j<k} S(c_{k,j,T}(u)) = S(s_{k,T}(u)) + \sum_{i>k} S(c_{i,k,T}(u)) \]

and

\[ \Delta E_{k,T} \leq \frac{L_k}{T} \]  

where, as discussed above, \( E_i \) and \( L_i \) are the remaining energy and target lifetime at node \( i \) at the time of solution, \( s_{i,T}(u) \) and \( c_{i,T}(u) \) are the data generated through sensing and connections, respectively, at node \( i \), \( T \) is a period of time and \( V \) is a utility expression defined by the application’s designers.

Since \( V(c_{i,k,T}(u)) = p_{i,k}(u) \ast J_{u,T}(i) \), where \( p_{i,k}(u) \) is the proportion of \( i \)'s utility that is routed to \( k \),

\[ J_{u,T}(k) = V(s_{k,T}(u)) + \sum_{i>k} p_{i,k}(u) \ast J_{u,T}(i). \]  

(2.3)
J can be maximized through policy iteration. Starting with an arbitrary feasible policy, \( \mu_0 \), repeat

\[
\begin{align*}
J_{\mu_j,T}(k) &= \mathcal{V}(s_{k,T}(\mu_j)) + \sum_{i>k} p_{i,k}(\mu_j) * J_{\mu_j,T}(i) \\
\text{and} \quad \mu_{j+1}(k) &= \arg\max_{u(k)} \mathcal{V}(s_{k,T}(u)) + \sum_{i>k} p_{i,k}(u) * J_{\mu_j,T}(i).
\end{align*}
\] (2.4)

The policy selection process at each step, Eq. 2.4, makes each policy, \( u_j \), a feasible policy, and the series of policies, \( u \), a series of policies with non-decreasing utility. Policy iteration is known to converge on an optimum solution in a finite number of steps \[41\][6]. Note that the equations for both steps of policy iteration for each node reference only nodes that are immediate neighbors. They also only include values that can be immediately measured by either the node or its immediate neighbors. Each step of the policy iteration therefore can be solved in a distributed manner.

### 2.4 Distributed Algorithm

#### 2.4.1 Dynamic Optimization

Expanding the energy management problem to include both sensor scheduling and routing makes it a useful tool to employ in this case. Like all algorithms that solve an optimization problem during runtime, the algorithm derived from the above dynamic programming converges toward an optimal policy, but it is not itself an optimal solution because it must use time and energy to find that policy.

DFEM begins by implementing a priority list of constraints, including minimal sensing, energy and routing constraints, as in EMA. Once a node’s constraints are satisfied, remaining resources are allocated according to a distributed dynamic programming optimization using policy iteration. If a node does not have enough resources to satisfy its base constraint set, DFEM reduces to predictable, priority-based failure modes as in EMA. To implement policy iteration of the dynamic programming formulated in Eq. 2.4, each node must solve a local optimization given local options for sensing, the data available for routing at child nodes, the amount of routing bandwidth provided by parent nodes, and the energy constraint given the node’s battery level and any energy harvesting available to it.

Nodes begin with a handshake process to determine neighbors and their distance from the nearest base stations. Radio schedules are broadcast during a contention period each cycle. Whenever potential child nodes are detected, a node chooses and broadcasts potential binding slots for a handshake. Subsequently, parent and child nodes exchange messages about their local
optimization problems. The parent node shares its current routing policy, while the child node shares its routing needs to satisfy each constraint, as well as estimates of utility deliverable for a set of routing policies around the previous policy. If more radio traffic is scheduled than can be transmitted using the current number of reserved slots, parent and child arrange another reserved slot in a time where neither detects radio traffic that might cause collisions. The broadcast radio schedules during the contention period allow them to reduce the potential for hidden terminal problems.

Each node solves its local optimization problem, given the routing constraints and utility available at child nodes, its local constraints and utility expressions for sensing, and the bandwidth granted by parent nodes’ routing policies. Messages to neighbor nodes are reevaluated under the new locally optimal policy, and propagated during the next schedule cycle. If more data is scheduled to pass between a child and parent node than can transfer during a single time slot, and additional slot is negotiated during the next handshake. By repeatedly solving the locally optimal node policy, $\mu$, at each node and propagating to neighbor nodes both bandwidth constraints and the utility, $J_\mu$, delivered by potential routing policies, the nodes collectively perform policy iteration as in Eq. 2.4. When calculating $J_\mu$, a node includes the utility of its sensor readings scheduled to fulfill constraints and solves the local utility optimization problem, each node uses the most binding constraint: radio bandwidth or energy.

The local optimization takes the place of the energy manager in EMA, and uses a similar energy monitor to track energy expenditures and harvesting. Schedules allow for a fractional number of time slots allocated to any given task by skipping some $k$ of every $n$ cycles. Allowing any rational number of scheduled slots allows nodes to optimize by following the highest utility gradient, and prevents the optimization problem from becoming a much harder to solve suitcase packing problem. While the utility functions of local sensing are known to each node, the utility of routing child nodes’ traffic is partially determined by the network topology, which is discovered during the (re)configuration phase of run time. This makes more sophisticated local optimization techniques problematic, so nodes engage in a simple gradient maximizing search, evaluating the utility functions at predefined intervals and comparing their locally linear contributions to application utility against potential child node routing opportunities.

### 2.4.2 Configuration and Extensions

DFEM is implemented on two open-source platforms, the simulation platform Omnet++ and the wireless sensor node Oracle SPOTs. In addition, there are two layers of user customization and configuration interfaces. The primary configuration method is defining the sensor constraints and utility expressions for sensing, expressed in terms of the number of sensor readings taken per schedule cycle. These constraints and expressions are managed within a configuration
file, and are written using simple mathematical notation. In addition, there is an API for writing extensions to currently supported sensors or for supporting new sensor hardware. The API is targeted towards programmers with some experience in C++ or Java, but does not require any specialized WSN training. Users describe the data produced by the new task and the process by which that data is generated, by implementing interfaces and abstract classes. These new data types and sensor tasks are treated as first-class citizens of the system and may be configured in the same way as any other sensors.

Utility expressions are generally non-linear, to account for an idea of diminished returns of sensing the same external quantity more frequently. DFEM follows the maximum utility gradient for its most binding resource, energy or routing bandwidth. Utility expressions are evaluated at sensing frequencies separated by constant small intervals, and local linearity is assumed between these sensing frequencies, to reduce computational complexity. When a schedule requires a fractional number of sensing actions per cycle, the next integer value of slots are reserved for that sensing task. Periodically, one of these reserved time slots is skipped, to maintain the correct average number of senses per cycle.

2.5 Test Implementations

2.5.1 Description

Two test implementations were created to investigate the efficiency of DFEM, and to compare it to existing approaches, such as EMA. A prototype built on Oracle SPOT wireless sensor nodes was used to test indoor and outdoor deployments and to validate the simulation. A simulation built on Omnet++ using MiXiM was used to test larger-scale deployment cases. For the purpose of validating the simulation’s results against the actual prototype network, two test topologies were created with Oracle SPOTs and replicated within the simulation. The first topology is a simple, linear topology with five sensor nodes and a base station. This topology provides simple results that can be used to illustrate the functioning of the base DFEM algorithm and extensions discussed later in this paper. A more realistic topology with thirty-one sensor nodes and one base station was also created as both a simulation and a physical network of SPOTs. Since the simulated nodes are assumed to have uniformly transmitting antennas, which is not an accurate assumption for SPOT radios, the physical configuration of simulated nodes was altered to replicate the radio connectivity of the physical networks.

Each node has three sensor sampling tasks to illustrate how the network can balance competing goals: taking a light reading, a temperature reading or recording the residual battery level of the node, itself. The minimum constraint in all cases is 1 repetition of each sensor task per schedule cycle. Each node’s highest priority is maintaining its lifetime constraint. Taking
one light reading is the next priority, followed by routing data, then taking one temperature reading and one residual battery reading. When EMA is run, the system maximizes the number of light readings after fulfilling constraints. For DFEM, the utility expressions of light, energy sensor, and temperature tasks are \(\frac{7}{3}r^2\), \(\frac{17}{7}r^\frac{4}{3}\), and \(\frac{15}{7}r^\frac{4}{3} + \frac{7}{5}r^\frac{4}{3}\), respectively, where \(r\) is the number of repetitions for a given task at any one node, up to a maximum of \(\frac{29}{5}\) times per schedule cycle. These utility expressions were chosen to have a similar range within the domain of possible policies at each node. For each of the tests discussed in this paper, nodes are assumed to be homogeneous in their capabilities, resources and duties. Nodes are modeled after Oracle SPOTs, which draw 86mA while awake, 31mA while in shallow sleep and 33\(\mu\)A while in deep sleep. The CC2420 radio unit SPOTs use draws 18.8mA while receiving, 17.4mA while transmitting and 0.021\(\mu\)A in sleep mode. DFEM does not require this complete uniformity of node capabilities and duties, but it simplifies the discussion of test cases’ solution points and convergence. A more in-depth discussion of DFEM’s current and potential ability for heterogeneity in WSN deployments is included in the Future Work section.

2.5.2 Results

Consider a simple, five-node linear topology as in Fig. 2.1. Nodes use a schedule of 420s divided into slots that are 6s each. The contention window at the beginning of each schedule is 6s. Each node has energy harvesting capabilities that charge its battery at a rate of 17mA. After a short period, the network working under DFEM (Fig. 2.4) stabilizes to a near-optimal policy. Changes in energy cost estimates and packet drop rates prevent the network from completely converging. For Nodes 0 and 1, where the energy constraint is binding, nodes prefer the less energy expensive option of sensing locally. Among nodes that are not energy constrained, the available bandwidth is distributed equally. Using EMA (Fig. 2.5), nodes instead simply maximize their number of light readings per schedule cycle. Since nodes prefer routing to sensing, the nodes closer to the base station reduce their sensing rates.
Figure 2.3: Data Generated Per Node under DFEM and EMA
Figure 2.4: Tasks Scheduled at Each Node under DFEM
Figure 2.5: Tasks Scheduled at Each Node under EMA
For a more realistic case, consider the 31-node topology displayed in Fig. 2.6. Nodes have multiple paths to the base station, and many have several neighbors, some of which are neither parent nodes nor child nodes. Six nodes from throughout the network have been highlighted and labeled. The sensor reading repetitions per schedule for each of those nodes are shown in the graphs below. Total data generated at all nodes is also shown. To accommodate the larger number of nodes, and therefore larger routing demand generated by the 31 node network topology, nodes have been equipped with an energy harvesting rate of 42mA. Again, under EMA (Fig. 2.9) nodes prefer routing to sensing, and maximal sensing rates are seen at nodes near the edges of the network. DFEM (Fig. 2.8) adapts the policy at each node, depending on which constraint is binding.

Figure 2.6: 31-Node Topology
Figure 2.7: Data Generated Per Node under DFEM and EMA
Figure 2.8: Tasks Scheduled at Each Node under DFEM
Figure 2.9: Tasks Scheduled at Each Node under EMA
2.5.3 Limitations

Estimation

The optimization step at each node is based on estimations of the actual constraints the node faces. Specifically, each node must estimate the energy costs of its tasks, as well as the bandwidth constraints between itself and its parents and children. This estimation needs to be lightweight in terms of memory and processing requirements, since sensor node hardware is limited. The current implementations of DFEM use a simple iterative weighted average. Decay rates that provided the most stable solutions were arrived at experimentally. More sophisticated estimation techniques that could offer values that are more stable, but still accurate, might allow for more stable network solutions.

Oracle SPOT hardware and MiXiM’s battery simulation do not currently support fine-grain analysis of energy draws from hardware on the nodes. Consequently, estimates of energy use by the node were performed using the difference in battery level before and after completing a task. This course-grain estimation is less precise, and also prevents some estimations from being performed on nodes with environmental harvesting capabilities if the battery is already full at the beginning of any given task. If there is not enough energy used to deplete the battery in spite of energy harvesting, it is impossible to distinguish how much energy is actually being used by the node. More fine-grain energy use monitoring hardware and models would improve the precision of estimations and remove the cases where no estimation could be performed. Fine-grain energy monitoring hardware has been reported in other wireless sensor node hardware, such as Trio nodes [14], but was not available for this research.

Overhead Cost

There is significant overhead cost from the contention period schedule exchange messages, as well as from exchanges between parent and child nodes. For real deployments to be feasible, this overhead would need to be amortized over the steady-state phase(s), where message exchanges between nodes could be kept to only data transfer. More research is needed to find reasonable criteria for state changing between (re)configuration phase and steady-state phase, as well as how often the network should check whether a state change is needed. There would be a tradeoff between the overhead cost of testing whether reconfiguration is necessary and the latency of state-change once reconfiguration is required.

Network Density

In overly dense networks, some nodes may not be integrated during the contention period for an indeterminate number of cycles. This problem can be mitigated by tuning the schedule slot length and duration. For very dense deployments, some process of selectively temporarily deactivating a subset of nodes may be required, as reported in [[cite]]. Another way to mitigate the node density problem might be to redesign the contention period, so that nodes are only
in contention with sibling nodes. Nodes and their grandparents will likely suffer from a hidden terminal problem. By definition, a node and its grandparent node do not have a direct radio connection, but share a neighbor. This means that they can cause packet collisions without either node being able to detect them. Sibling nodes may still have a hidden terminal problem, but do not necessarily. Therefore, splitting the contention period into three parts, with nodes only using the part corresponding to their distance in hops from the base station, modulus 3, should reduce the problems with packet collisions during the contention period.

2.5.4 Simulation-Prototype Comparison

To validate and verify the simulation results against actual network deployments, prototype deployments of both simulated topologies were installed using Oracle SPOTs. Since the uniform radio range assumption in MiXiM libraries does not accurately describe the Oracle SPOT nodes, the connectivity of the topologies was duplicated rather than the physical spacing from the simulation. To illustrate the differences between the simulation and test deployment results, a graph isolating the data generated at only the highlighted nodes is displayed, as well as the task scheduling policy at each highlighted node. The scenario configuration is identical to the 31-node case discussed previously.

![Topology](image1.png)  ![Test Network](image2.png)

Figure 2.10: 31-Node Topology and Test Network

There are two major differences between the physical deployments and their simulation counterparts. First, the SPOT program uses a memory paging system to back up data collected to flash memory, and also uses flash to record deployment statistics. Records in flash memory allow recovery of data and analysis of deployment statistics after a node has been recovered at
the end of a deployment. There is not currently a mechanism in the simulation to accurately reflect the energy costs of this flash memory use. Neither MiXiM nor the base Omnet++ libraries include flash memory modeling, and developing an accurate simulation of flash memory energy use was outside the scope of this project. Therefore for each node, the costs of all sensor tasks, radio data routing and the overhead cost of solving its local optimization problem are all higher than represented by the simulation. This difference causes the chosen network policy of a physical deployment to be different than the chosen network policy of a simulation of the same case. Though the simulation (Fig. 2.8) and prototype (Fig. 2.12) deployments both converge to a network policy using policy iteration, and the simulation accurately reflects the process by which DFEM works and its potential uses both in the base case example and the later extensions in this paper, the simulation results should not be taken to be predictive of actual deployment behavior for any given scenario, because the unrepresented flash memory energy costs lead to a consistent network policy choice difference.

The second major difference between the simulation and prototype deployments is that background radio noise is not completely accounted for by the MiXiM radio simulation library. The simulation models a bit error rate in radio transmissions, but in reality the bit errors caused by background radio noise are not independently distributed as they are in the model. Since the SPOT radios use the 802.15.4 standard, which is also used by other popular devices, there is also the possibility that radio noise could cause packet collisions, which is not in the simulation model. The experiment area was cleared of known devices that could cause interference, to reduce the risk of contamination by radio noise.

![Figure 2.11: Data Generated Per Node in Simulation and Test Deployment](image-url)
Figure 2.12: Tasks Scheduled at Each Node under DFEM
Chapter 3

Extensions on the Base Algorithm

3.1 Optimal Lifetime with Utility Inflation

DFEM allows a network to balance competing application goals, such as sensing and routing data. However, the lifetime of each node is still formulated as a constant constraint. This formulation begs the question, what is the "best" lifetime constraint for each node in a given deployment? Might that best constraint change due to circumstances during run time? Reformulating the optimization problem, (1), to be a maximization of total utility accumulated over the network’s remaining lifetime instead of only over one schedule frame iteration provides a basis for answering questions about choosing an optimal lifetime.

\[ J_{u,T}(k) = \sum_{t=1..n_k} \left( \mathcal{V}(s_{k,T}(u)) + \sum_{i > k} \mathcal{V}(c_{i,k,T}(u)) \right) \]

\[ = n_k \left( \mathcal{V}(s_{k,T}(u)) + \sum_{i > k} \mathcal{V}(c_{i,k,T}(u)) \right), \]

and

\[ J_{u,T}(0) = \sum_{t=1..n_k} \left( \sum_{j \in N_0} J_{u,T}(j) \right) \]

\[ = n_k \left( \sum_{j \in N_0} J_{u,T}(j) \right), \]

where \( n_k = \) lifetime in schedule frame iterations at node \( k \), and \( n_k \leq n_j \forall j \) such that \( p_{k,j}(u) > 0 \).
However, in practice the actual working lifetime of a sensor node is uncertain. There is always a chance one or more nodes may miscalculate its resources or be affected by some external environmental factors that reduce its practical lifetime. One extreme example is if some damage causes a node to cease functioning prematurely. To account for the possibility of premature failure, future utility can be discounted recursively, like an inflation applied to future utility.

\[ J_{u,T}(k) = \sum_{t=1}^{n_k} I^t \left( \mathcal{V}(s_{k,T}(u)) + \sum_{i>k} \mathcal{V}(c_{i,k,T}(u)) \right) \]

\[ = \left( \frac{1 - I^{n_k}}{1 - I} \right) \left( \mathcal{V}(s_{k,T}(u)) + \sum_{i>k} \mathcal{V}(c_{i,k,T}(u)) \right) , \]

and

\[ J_{u,T}(0) = \sum_{t=1}^{n_k} I^t \left( \sum_{j \in N_0} J_{u,T}(j) \right) \]

\[ = \left( \frac{1 - I^{n_k}}{1 - I} \right) \left( \sum_{j \in N_0} J_{u,T}(j) \right) , \]

where \( n_k = \) lifetime in schedule frame iterations at node \( k \), and \( n_k \leq n_j \forall j \) such that \( p_{k,j}(u) > 0 \), and \( I \) is an inflation factor, \( 0 \leq I \leq 1 \).

This new formulation can be solved via policy iteration, as was Eq. 2.4, however each decision stage is no longer a simple, nondecreasing function up to a constraint. Increasing energy consumption by scheduling the next most valuable task at a node has two effects on the lifetime utility generated by that node. It increases the utility generated per schedule cycle, and also decreases the node’s lifetime. Because of the parental lifetime constraint in Eq. 3.2, this tradeoff leads to three separate cases when evaluating whether an increased energy expenditure will result in an overall increase in lifetime utility generated. When a node’s lifetime is greater than any of its parents, increased energy expenditures only increase lifetime utility, because any utility generated after the last parent dies would not count, anyway. When a node’s lifetime is less than any of its parents, both the increase in utility and the decrease in lifetime effect lifetime utility, and the increase in utility applies throughout the node’s life. When the node’s lifetime is greater than some of its parents, but not more than all of its parents, both the increase in utility and decrease in lifetime effects occur, but the increase in utility applies only to part of the node’s lifetime, and the decrease in lifetime only affects the part of the node’s current utility generated that it would be able to deliver to its living parent nodes. If the utility added by node tasks is organized via a water-filling algorithm, filling radio supply of longest lived parents first, this tradeoff is clearly visualizable, as in the figure below.
While solving the local optimization problem, a node may follow the maximum utility / energy gradient, checking a different condition depending on which of the three lifetime cases in which its current search state is. As stated before, while the node’s projected lifetime is greater than any of its parents, all increases in utility through increased energy expense increases total lifetime utility. When the projected lifetime of a node’s current search state is between its minimum and maximum parent lifetimes, the tradeoff between increased schedule utility and decreased lifetime is worthwhile when
where B is the current energy reserves stored in the node’s battery, e is the cost of the current search state schedule, $L_1$ is the nearest parent lifetime less than the node’s search state’s projected lifetime, and $V_2$ and $e_2$ are the utility added and energy cost, respectively, of the schedule that fully served the supply provided by parent nodes whose lifetimes are greater than the node’s search state’s projected lifetime.
When the projected lifetime of a node’s current search state is less than any of its parents, both the increase and decrease effects apply to the whole projected lifetime of the node. The tradeoff between these two effects is worthwhile when

\[ \frac{d}{de} \left( \frac{V \left( \frac{1 - I_b}{e - c} \right)}{1 - I} \right) > 0 \]

\[ \Rightarrow \]

\[ \frac{dV}{de} \left( \frac{1 - I_b}{e - c} \right) + \left( \frac{V}{1 - I} \right) \frac{d \left( \frac{1 - I_b}{e - c} \right)}{de} > 0 \]

\[ \Rightarrow \]

\[ \frac{dV}{de} \left( \frac{1 - I_b}{e - c} \right) + \left( -VB \ln(I) \left( \frac{I_b}{e - c} \right) \right) > 0 \]

\[ \Rightarrow \]

\[ \frac{dV}{de} \left( 1 - I_b \right) (e - c)^2 > VB \ln(I) \left( \frac{I_b}{e - c} \right) \]

\[ \Rightarrow \]

\[ \frac{dV}{de} > \frac{-VB \ln(I) \left( \frac{I_b}{e - c} \right)}{(1 - I_b) (e - c)^2} \]

where \( B \) is the current energy reserves stored in the node’s battery and \( e \) is the cost of the current search state schedule.

Because a node’s optimization process proceeds in one direction from longer to shorter projected lifetimes for its search states, the computational cost of evaluating Eq. 3.4 or Eq. 3.3 can be reduced by strategically storing some utilities for the current parental lifetime case. Using strategic caching, the computational cost is not much more expensive than the base DFEM solution. As DFEM searches along the maximal utility gradient at a node, it can check both whether the base lifetime constraint holds and whether the next locally linear step of the search passes the relevant lifetime utility increase condition.
3.1.1 Test Cases

Consider again the five node linear topology in Fig. 2.1. Each node has a 720mAh battery, and is using the dynamic lifetime optimization to decide the best lifetime utility policy, where $I = 0.995$. Again, the graphs in Fig. 3.3 are numbers of senses per cycle for each type of sensor data. Projected lifetime for each node is also displayed in Fig. 3.2. Note that the linear topology means that there is a bottleneck at the node which is energy constrained by the dynamic lifetime constraint. Nodes farther away in the network are merely deciding the optimal use of the routing bandwidth they are granted by energy constrained nodes.

![Figure 3.2: Target Lifetime at Each Node (5-Node Topology)](image)

Now consider the 31 node topology from Fig. 2.6. The projected lifetime for both the highlighted six nodes and for all network nodes is displayed in Fig. 3.4, and the sensing policy of highlighted nodes is displayed in Fig. 3.5. As in the simple, five-node topology, the lifetime constraints of nodes nearest the base station determine the final policy. Once nodes lose all paths to the base station, they will revert to fulfilling only constraints that have a higher priority than the routing constraint.
Figure 3.3: Tasks Scheduled under Dynamic Lifetime Constraint (5-Node Topology)
Figure 3.4: Target Lifetime at Each Node (31-Node Topology)
Figure 3.5: Tasks Scheduled under Dynamic Lifetime Constraint (31-Node Topology)
3.2 Rare Event Reporting

In the formulation of Eq. 2.1, DFEM assumes that data has utility as long as it is delivered. There is no measure of report latency, only of the utility of data delivered eventually. However, there is another paradigm for data collection, which focuses primarily on the speed of reporting. For example, a network deployed to notify users about critical conditions within a factory, or a network deployed to detect potential forest fires within an instrumented region, require speedy reporting to be most useful[26][29][45]. Many good solutions for the low-latency reporting problem have been offered in the wireless sensor network literature. A slightly different problem is the case where a wireless sensor network might be deployed to split its time between taking regular data readings and reporting an alarm or other rare event if any is detected. For this case, DFEM may be adapted to make a decision about how much of each node’s time and resources should be scheduled for implementing a rapid-report algorithm.

Several rapid-reporting algorithms call for all network nodes to activate their radios simultaneously, report along predetermined routes, and prevent transmission of redundant messages to minimize the simultaneous radio traffic and lower report latency. Assuming that average latency primarily depends on the time between active periods, during which a node and its parents are all able to relay messages, both constraints and utility expressions may be formulated in terms of the schedule slots dedicated to low-latency reporting. While a constraint formulation is a simple number of slots, just as any other constraint in DFEM, a utility expression for low-latency reporting tasks is slightly more nuanced. For any given number of scheduled repetitions of low-latency reporting during a node’s schedule, a more frequent number of repetitions only contributes to lower latency event reporting if one of its parent nodes will also be active and can forward event messages. On the other hand, the utility added by increasing the reporting task frequency at a node that is the parent of at least one other node depends on the number of more distant nodes routing through that node that would also increase their report task frequency as a result. If \( r_{k,T}(u) \) is the number of schedule slots dedicated to rapid event reporting at node \( k \) given the policy \( u \), \( v_q \) is the local utility expression for reporting tasks, evaluated at \( q \) repetitions per schedule cycle, and \( V_q(r_{k,T}(u)) \) is the total utility added by the \( q \)th repetition of reporting tasks scheduled at node \( k \),

\[
V_q(r_{k,T}(u)) = v_q \ast \left( 1 + \sum_{i > k} p_{i,k}(u) \ast V_q(r_{i,T}(u)) \right)
\]

(3.5)

\[
V(r_{k,T}(u)) = \sum_{q=1..r_{k,T}(u)} V_q(r_{k,T}(u))
\]

Assuming event reports include relevant data, the data size of report tasks is effectively 0, since no additional data will need to be reported during normal routing.
As with routing utility in base DFEM, the component of reporting task utility that depends on other nodes only references one-hop neighbors. During the exchange of routing information each schedule cycle, each child node, \( i \) can include in its report to parent node, \( k \), \( p_{i,k}(u) \) \( \mathcal{V}_q(r_{k,T}(u)) \) for each \( q \) within some interval around the current policy and not greater than the maximum \( r_{i,T}(u) \) that node \( i \) would schedule, given its parent nodes would support it. Nodes can use the additional information passed during message exchange to evaluate \( \mathcal{V}_q(r_{k,T}(u)) \) for each \( q \) during local optimization, and policy iteration can solve the network-wide problem as in base DFEM.

A major difference between a rapid-reporting task and standard DFEM sensor tasks at the sensor nodes is that nodes must execute a rapid-response task at the same time as their parent nodes for it to effectively reduce the average latency of reports. One simple way to achieve this required coordination is to choose a set of potential execution times within the repeating schedule. Nodes then choose to activate according to a predetermined order, as their local policy calls for more rapid-response activations per schedule cycle. For example, there might be six potential coordinated slots per schedule, and a node which chooses to schedule 2 might be active for the first and fourth, whereas a node which chooses to be active for three would be active for the first, fourth and sixth slots. The order should be the same for all nodes, and the progression should ideally reduce the average latency as much as possible at each step.

### 3.2.1 Test Cases

Consider again the two topologies discussed earlier. In addition to the three sensor readings, nodes now have a responsibility to check for a rare event using their sensors and report it over the radio immediately. This rare event report task has a constraint of once per schedule cycle and a utility expression of \( \frac{14}{5} r^{\frac{2}{3}} \), up to a maximum of \( \frac{9}{2} \) repetitions per schedule cycle. By applying DFEM as before, nodes reduce their sensor reading rates of the first three sensor types to provide lower latency rapid event reports. Sensor reading rates of each sensor, as well as average and maximum report latencies are shown in Fig. 3.6-3.9. Maximum latency reduces as a step function because of the coordination required among nodes when choosing active slots for rapid event reporting. One detail to note is that the radio use required by a rare event report task requires much more energy than the other three sensor readings.
Figure 3.6: Tasks Scheduled with Rare Event Reporting (5-Node Topology)
Figure 3.7: Report Latency at Each Node (5-Node Topology)
Figure 3.8: Tasks Scheduled with Rare Event Reporting (31-Node Topology)
Figure 3.9: Report Latency at Each Node (31-Node Topology)
3.3 Run Time User Input

Since programming and deploying sensor nodes requires an investment of manual labor that can be significant, WSN applications and middleware that allow for some latitude in configuration during runtime provide a clear benefit to their users. One widespread paradigm for allowing users to change the behavior of a sensor network during run time is considering the network a distributed database. Users push queries into the network, which are propagated to relevant nodes. Sensor readings then are collected at the nodes to satisfy the users’ queries and delivered back to the base station from which the queries originated[34]. An analogous query mechanism can be added to DFEM by propagating changes in constraints and utility expressions through the existing parent-child interactions. A periodic mandatory update as in the distributed SQL paradigm could be accomplished by changing the constraint for a given sensor task. In addition, the DFEM paradigm also allows users to change the utility expressions used to determine how resources are allocated once all constraints are met. Changes to constraints and utility expressions made by queries may be permanent or may include a lifetime after which the network would revert back to its previous values.

Using a compressed notation for constraints and utility expressions, the parent’s reply to its children’s bid messages can also contain without increasing the number of messages that need to be passed constraints and utility expressions for several tasks, depending on the complexity of the utility expressions being used. This constraint and utility expression update mechanism was added to the Omnet++ simulation of DFEM. For the purpose of these tests, the assumption of a common constraint set and utility expressions for all nodes is maintained. Changes to the constraints and utility expressions are propagated to all nodes in the network, and the time required to reach the new equilibrium is measured. However, DFEM does not require that constraints and utility expressions be the same at all nodes. With more work, it would be possible to send queries to update constraints and utility expressions only for a target subset of nodes.

3.3.1 Test Cases

Consider again the five node linear topology from Fig. 2.1 and the topology from Fig. 2.6, using the charge and timing assumptions from the base test case of each topology. At t = 15h, the utility expressions of light, energy sensor, and temperature tasks swapped to $15r^4 + 2r^4$, $7r^2$, and $17r^2$, respectively, where $r$ is the number of repetitions for a given task at any one node, up to a maximum of $29$ times per schedule cycle. Once the utility expressions are changed, the update is propagated from the base station through the network during parent / child message passing. Again, the number of task iterations per schedule cycle at each node is displayed in Fig. 3.10-3.11. Note the scheduled number of light sensing and temperature sensing tasks in
the network policy swap after the utility expression changes, at t = 15h. Since policy iteration
starts from the previous solution, the network converges to a new solution quickly after the
query is made to the network.

3.4 Conclusion

This paper has outlined several areas where DFEM solutions fall in the worst case to existing
approaches, while providing more flexibility and automated optimization. Like energy manage-
ment architecture approaches, DFEM begins with a set of minimum sensing constraints that
provide a controlled performance degeneration under conditions of energy scarcity. However, in
conditions where there is more than enough energy to fulfill the application constraints, DFEM
is also able to solve for the maximum global application utility added policy. This global ap-
plication utility optimization is achieved using local data only, without the need to either flood
state information to all nodes nor the need to collect state information to a central location
and to distribute the result. By adding the concept of utility inflation, DFEM is also able to
accommodate both a hard energy constraint as is common and a soft, utility optimal lifetime
that is determined as an objective. User queries distributed into the DFEM framework may be
structured as updates to the sensing constraints, in which case they will function as existing
sensor query messages, but users also have the option to add or modify utility expressions and
allow the network to re-stabilize to a new optimal policy that balances the users’ competing
requests. Finally, formulating low-latency emergency message reporting as a task with utility
and constraints allows a network deployed for regular sensor data gathering also to function
as an emergency reporting network, and vice versa. Minimum requirements and application
utility added by lower latency can be expressed just as with other DFEM tasks. In addition,
the DFEM framework is designed with a gradient of expertise requirements. New sensor net-
work tasks can be extended from the library with only basic programming skills required, and
network configuration and deployment requires no special expertise.

Overall, the capabilities of DFEM’s approach are demonstrated in the results discussed in
this paper, but more work on details of implementation is needed before the software would be
robust enough for widespread use, and hardware considerations are also an issue. The primary
software problem is that the contention period is fragile when nodes are dense. The network
optimizes policy over nodes that join the network, but network joining is not guaranteed in dense
networks. Better estimation techniques that can be more accurate while still functioning within
the small computation and storage requirements of sensor nodes would also allow the network to
stabilize faster. Once equilibrium is reached, the network should have a mechanism for running
the determined solution without the overhead of the messages passed during the configuration
phase, and a trigger for reentering the configuration phase if large changes occur to the nodes

44
Figure 3.10: Tasks Scheduled at Each Node with User Update
Figure 3.11: Tasks Scheduled at Each Node with User Update
or to the constraints or utility expressions. Finally, the hardware of Oracle SPOTs, while ideal for fast prototyping of WSN applications, lacks the hardware needed for fine-grain energy use monitoring, which would contribute to better estimations of energy costs. Also, SPOTs do not have the environmental hardening of nodes designed for long-term outdoor deployments, such as the TRIO motes.

3.5 Future Work

3.5.1 Heterogeneous Networks

The test cases investigated in this paper all used nodes with the same capabilities, constraints and utility expressions. The simplicity of this assumption made it easier to illustrate how the distributed optimization converges during each test case, and simplified the discussions for each test case. However, DFEM already supports heterogeneity of capabilities and configurations in several ways. The main constraint is that nodes are required to share an encoding for the messages passed between parent and child nodes. More work is needed to demonstrate how effectively DFEM is able to work in heterogeneous deployments, focusing on several currently available scenarios.
If some nodes are deployed with more powerful radios and larger energy reserves or energy harvesting capabilities, the network self-organizes into a hierarchical set or clusters around these super-nodes. If nodes are deployed with different sensor capabilities, they can locally detect the available hardware and perform their local constraint matching and optimization using only the constraints and utility expressions of sensor tasks that are available to that node. Even without differences in hardware capabilities, nodes can be configured with different constraints and utility expressions from each other, if there is some reason for the application to prefer a type of sensor reading in one particular area. For example, taking frequent light readings at nodes that are under a loose canopy of vegetation might be of more utility than taking light readings in an empty field.

3.5.2 More Complex Network Topologies

Four potential extensions would increase the flexibility of DFEM’s radio medium use. First, modifying the encoding of messages passed during contention periods to include channel information would allow nodes to transmit simultaneously with neighbors, but on different radio channels. This would increase the radio bandwidth available to the network, for cases where energy is not the most limiting factor. Second, if nodes maintain routing tables for their descendant nodes, user input queries could be sent to a target subset of the network. Third, by modifying the encoding of messages passed during contention windows and between parent and child nodes, multiple sets of network sinks could be designated for different kinds of data generated by the network. Finally, consider the case where a node, n, has two neighbors, p and q, from different C circles as described in Fig. 2.2. However, the higher-numbered node, q, is not using n as a parent, but has a different path available to it. In this case, it would be possible for node n to use any remaining resources it has after fully utilizing its path through its parent node p, and route remaining sensor packets it generates through q as an alternative parent, but DFEM would not consider this option as feasible, as the distributed optimization is currently formulated.

Multiple Data Destinations

Currently, if more than one base station is introduced to the network, nodes will self-organize to deliver data generated to the nearest sink available. For some applications, however, there could be more than one class of destination. For example, humidity readings may need to be collected in one location, and temperature readings collected at another. Alternatively, users at different base stations may set up different queries introducing utility expressions into the network. For each class of data flow added to the network, the distance in hops as in Fig. 2.2 would need to be added to messages passed during the contention period. Also, the overhead of parent-child
message exchange would need to be paid for each class of data flow. However, if these two changes are made to the DFEM message passing, the system could use policy iteration to solve for the optimal resource sharing policy among different data flows through each node. Further study would be necessary to determine how quickly DFEM would converge on a solution under conditions of multiple data flows, as well as how many data flows could be accommodated before the additional overhead would prevent normal functioning of the network.

**Virtual Nodes**

Consider the case described in Fig 3. Node n has two paths of different lengths available to it, node q is not a child of n and node n is not using all of its resources to route through its parent node, p. In this case, DFEM does not consider any solutions that route data from n through q to the base station, since this appears to the algorithm to be sending data away from the destination, not towards it. However, this limitation can be overcome by introducing the concept of a virtual node, so there is node n and node n’, where n’ is numbered as if it were part of the next concentric circle from q. Node n exchanges parent-child and contention messages in both its original persona and as n’. During the local optimization phase, node n first uses all its possible resources routing through p, then runs a local optimization again as n’, where n’ has resources equal to those left over after the first optimization phase, and behaves as a child of q.

### 3.5.3 Empirical utility expressions

For the tests of DFEM in this paper, utility expressions were set by user input either before deployment or during run time. However, there could be cases where the utility of sensing would depend on external factors in the node’s environment. For example, a user might want to use a larger utility expression for temperature sensor readings while the temperature is above some threshold. Alternatively, there may be a link between two different types of sensor readings. For example, the rare event report for detecting wildfires might have a higher utility while the humidity sensor is reporting readings below some threshold.

More work is needed to implement the encoding and evaluation of condition-dependent utility expressions. Also, the effects of condition-dependent utility expressions on network convergence need to be investigated. Safeguards to prevent multiple rapid transitions around the threshold utility will probably be necessary to allow convergence after each transition. One possible safeguard could include having a buffer between the threshold to enable the utility-function condition and the threshold at which it would be disabled. For example, the alternative temperature utility expression could be enabled at 90F and disabled if it falls below 85F. Another possible safeguard might be including some minimum duration for which the conditional
utility expression would be enabled.

Another possible way to base sensor readings’ utilities on empirical measures would be to make utility a function of some aggregate statistic of the data, such as average volatility of readings. However, if a statistic is used that causes utility to change too rapidly, it would prevent network convergence. More investigation is needed to determine under which conditions empirical utility measures might be used effectively.
REFERENCES


Appendix A

Full-Page Graphs

A.1 Base Algorithm Simulations

A.1.1 Data Generated Per Node under DFEM and EMA
Figure A.1: Data Generated Per Node under DFEM
Figure A.2: Data Generated Per Node under EMA
A.1.2 Tasks Scheduled at Each Node under DFEM

Figure A.3: Tasks Scheduled at Node 0 under DFEM
Figure A.4: Tasks Scheduled at Node 1 under DFEM
Figure A.5: Tasks Scheduled at Node 2 under DFEM
Figure A.6: Tasks Scheduled at Node 3 under DFEM
Figure A.7: Tasks Scheduled at Node 4 under DFEM
A.1.3 Tasks Scheduled at Each Node under EMA

![Diagram showing tasks scheduled at Node 0 under EMA](image)

Figure A.8: Tasks Scheduled at Node 0 under EMA
Figure A.9: Tasks Scheduled at Node 1 under EMA
Figure A.10: Tasks Scheduled at Node 2 under EMA
Figure A.11: Tasks Scheduled at Node 3 under EMA
Figure A.12: Tasks Scheduled at Node 4 under EMA
A.1.4 Data Generated Per Node under DFEM and EMA

Figure A.13: Data Generated Per Node under DFEM
Figure A.14: Data Generated Per Node under EMA
A.1.5 Tasks Scheduled at Each Node under DFEM

Figure A.15: Tasks Scheduled at Node 0 under DFEM
Figure A.16: Tasks Scheduled at Node 1 under DFEM
Figure A.17: Tasks Scheduled at Node 2 under DFEM
Figure A.18: Tasks Scheduled at Node 3 under DFEM
Figure A.19: Tasks Scheduled at Node 4 under DFEM
Figure A.20: Tasks Scheduled at Node 5 under DFEM
A.1.6 Tasks Scheduled at Each Node under EMA

![Tasks Scheduled at Node 0 under EMA](image)

Figure A.21: Tasks Scheduled at Node 0 under EMA
Figure A.22: Tasks Scheduled at Node 1 under EMA
Figure A.23: Tasks Scheduled at Node 2 under EMA
Figure A.24: Tasks Scheduled at Node 3 under EMA
Figure A.25: Tasks Scheduled at Node 4 under EMA
Figure A.26: Tasks Scheduled at Node 5 under EMA
A.2 Prototype Deployment

A.2.1 Data Generated Per Node in Simulation and Test Deployment

![Graph showing data generated per node in simulation](image)

Figure A.27: Data Generated Per Node in Simulation
Figure A.28: Data Generated Per Node in Test Deployment
A.2.2 Tasks Scheduled at Each Node under DFEM

![Diagram showing tasks scheduled at each node under DFEM](image)

Figure A.29: Tasks Scheduled at Node 0 under DFEM
Figure A.30: Tasks Scheduled at Node 1 under DFEM
Figure A.31: Tasks Scheduled at Node 2 under DFEM
Figure A.32:  Tasks Scheduled at Node 3 (node failure) under DFEM
Figure A.33: Tasks Scheduled at Node 4 under DFEM
Figure A.34: Tasks Scheduled at Node 5 under DFEM
A.3 Extensions on the Base Algorithm (Dynamic Lifetime Constraint)

A.3.1 Tasks Scheduled under Dynamic Lifetime Constraint (5-Node Topology)

Figure A.35: Tasks Scheduled under Dynamic Lifetime Constraint (Node 0)
Figure A.36: Tasks Scheduled under Dynamic Lifetime Constraint (Node 1)
Figure A.37: Tasks Scheduled under Dynamic Lifetime Constraint (Node 2)
Figure A.38: Tasks Scheduled under Dynamic Lifetime Constraint (Node 3)
Figure A.39: Tasks Scheduled under Dynamic Lifetime Constraint (Node 4)
A.3.2 Tasks Scheduled under Dynamic Lifetime Constraint (31-Node Topology)

Figure A.40: Tasks Scheduled under Dynamic Lifetime Constraint (Node 0)
Figure A.41: Tasks Scheduled under Dynamic Lifetime Constraint (Node 1)
Figure A.42: Tasks Scheduled under Dynamic Lifetime Constraint (Node 2)
Figure A.43: Tasks Scheduled under Dynamic Lifetime Constraint (Node 3)
Figure A.44: Tasks Scheduled under Dynamic Lifetime Constraint (Node 4)
Figure A.45: Tasks Scheduled under Dynamic Lifetime Constraint (Node 5)
A.4 Extensions on the Base Algorithm (Rare Event Reporting)

A.4.1 Tasks Scheduled with Rare Event Reporting (5-Node Topology)

Figure A.46: Tasks Scheduled with Rare Event Reporting (Node 0)
Figure A.47: Tasks Scheduled with Rare Event Reporting (Node 1)
Figure A.48: Tasks Scheduled with Rare Event Reporting (Node 2)
Figure A.49:  Tasks Scheduled with Rare Event Reporting (Node 3)
Figure A.50: Tasks Scheduled with Rare Event Reporting (Node 4)
A.4.2 Report Latency at Each Node (5-Node Topology)

Figure A.51: Report Latency at Each Node (Node 0)
Figure A.52: Report Latency at Each Node (Node 1)
Figure A.53: Report Latency at Each Node (Node 2)
Figure A.54: Report Latency at Each Node (Node 3)
Figure A.55: Report Latency at Each Node (Node 4)
A.4.3 Tasks Scheduled with Rare Event Reporting (31-Node Topology)

Figure A.56: Tasks Scheduled with Rare Event Reporting (Node 0)
Figure A.57: Tasks Scheduled with Rare Event Reporting (Node 1)
Figure A.58: Tasks Scheduled with Rare Event Reporting (Node 2)
Figure A.59: Tasks Scheduled with Rare Event Reporting (Node 3)
Figure A.60: Tasks Scheduled with Rare Event Reporting (Node 4)
Figure A.61: Tasks Scheduled with Rare Event Reporting (Node 5)
A.4.4 Report Latency at Each Node (31-Node Topology)

Figure A.62: Report Latency at Each Node (Node 0)
Figure A.63: Report Latency at Each Node (Node 1)
Figure A.64: Report Latency at Each Node (Node 2)
Figure A.65: Report Latency at Each Node (Node 3)
Figure A.66: Report Latency at Each Node (Node 4)
Figure A.67: Report Latency at Each Node (Node 5)
A.5 Extensions on the Base Algorithm (User Update)

A.5.1 Tasks Scheduled at Each Node with User Update

![Tasks Scheduled at Node 0 with User Update](image)

Figure A.68: Tasks Scheduled at Node 0 with User Update
Figure A.69: Tasks Scheduled at Node 1 with User Update
Figure A.70: Tasks Scheduled at Node 2 with User Update
Figure A.71: Tasks Scheduled at Node 3 with User Update
Figure A.72: Tasks Scheduled at Node 4 with User Update
A.5.2 Tasks Scheduled at Each Node with User Update

Figure A.73: Tasks Scheduled at Node 0 with User Update
Figure A.74: Tasks Scheduled at Node 1 with User Update
Figure A.75: Tasks Scheduled at Node 2 with User Update
Figure A.76: Tasks Scheduled at Node 3 with User Update
Figure A.77: Tasks Scheduled at Node 4 with User Update
Figure A.78: Tasks Scheduled at Node 5 with User Update