ENERGY EFFICIENT DATA DRIVEN DISTRIBUTED TRAFFIC SIMULATIONS

A Dissertation
Presented to
The Academic Faculty

by

SaBra Alexandria Neal

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering

Georgia Institute of Technology
May 2018

ENERGY EFFICIENT DATA DRIVEN DISTRIBUTED TRAFFIC SIMULATIONS

ACKNOWLEDGEMENTS

This thesis would not be possible without the time and knowledgeable contribution of my advisor Dr. Richard Fujimoto. Dr. Fujimoto thank you for your advice and support throughout my doctoral studies. I would also like to thank my committee members for their time and input: Dr. Michael Hunter, Dr. David Goldsman, Dr. Margaret Loper, and Dr. Richard Vuduc. I must extend a thank you to my fellow graduate students who were always there to provide feedback whenever needed: Philip Pecher, Aradhya Biswas, Caleb Robinson, and Mark Jackson.

I would also like to personally thank my personal support system. The completion of my thesis would not be possible without the unconditional love and support of my mother and father, Cynthia Neal and Charles Heyward. Thank you to my aunt and uncle, Vermell and Larry Foster who have always treated me as their own child and were always there without hesitation whenever I needed anything. LaShonda Foster and Amber Neal thank you for being not only my cousins; but, also my sisters. You guys were always a flight, phone call, or text message away whenever I needed you. Thank you two personally for being there for me during the difficult times just as much as the good. Thank you to Katrina Oliver for all your help over the years even during my undergraduate years for providing advice, edits, or whatever else I needed in little to no time. Last, but not least I have to thank Mr. Barry Lindler, my high school computer science teacher for seeing my potential in the field of computing at a young age and not allowing me to ignore it. This degree wouldn’t even be fathomable without your faith in my abilities.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

DDDAS	Dynamic Data Driven Application Systems
DDM	Data Distribution Management
DoD	Department of Defense
HLA	High Level Architecture

SUMMARY

With the growing capabilities of the Internet of Things and proliferation of mobile devices interest in the use of real-time data as a means for input to distributed online simulations has increased. Online simulations provide users with the ability to utilize real-time data to make adaptations to the system, e.g., to adjust to unexpected events. One problem that arises when using these systems on mobile devices is that they are dependent upon the device’s stored energy. It is vital to understand how all components of such a system use the stored energy in order to understand how to develop such systems for energy constrained environments.

One aspect of this thesis is to examine the role that discrete event driven and cellular automata models have on energy consumption in embedded systems. Discrete event driven simulations are dependent on a future event list for execution. It is important to understand the affect of the data structure for the future event list on energy consumption when running such simulations in embedded systems. This thesis presents a characterization of the relationship between the operations performed on the future event list and energy consumption.

Further, this thesis investigates the energy consumed in running a distributed Dynamic Data Driven Application System (DDDAS) for online traffic simulations in energy constrained environments. Such a system consists of embedded traffic simulations and requires a means of communication amongst the distributed simulations in order to characterize the behavior of the entire system. Data Distribution Management (DDM) provides a means to manage communications in such a system. Understanding how the components of DDM systems consume energy is essential to understanding how to design and use such a system in energy constrained environments. This thesis explores energy consumption in DDM systems.

This thesis investigates an energy aware approach applicable for systems that are restricted to energy constrained environments. The approach offers mechanisms to implement energy aware distributed simulations and communication mechanisms. This thesis assesses the role that discrete event driven and cellular automata models have on energy consumption in embedded systems.

1 INTRODUCTION

The Internet of Things is a growing paradigm that involves devices embedded in everyday objects that are interconnected via the Internet. This paradigm is quickly growing and researchers predict that by the year 2025 that Internet nodes will reside in everyday objects such as furniture and paper documents. Mobile phones, vehicles, and traffic sensors are some of the devices that currently have the ability to utilize unique addressing to communicate using the internet of things paradigm. This paradigm includes growing use of mobile devices with increasing computing capabilities, well beyond basic voice communication. Energy and power consumption become a major concern for mobile devices operating on batteries.

Prediction systems that forecast future traffic network states are utilizing real-time data in order to drive embedded distributed traffic simulations within mobile devices. These systems obtain real-time data from online systems, embedded sensors within the environment, crowdsourcing, etc. Communication is required among these data sources and distributed embedded traffic simulations.

Dynamic Data Driven Application Systems (DDDAS) are a type of adaptive system that utilizes real-time data to drive the application [1]. DDDAS relies on real-time data to execute the embedded application systems. One of the major concerns of relying on real-time data is retrieving and receiving real-time data from environmental embedded sensors to be received by mobile devices that are hosting the DDDAS application. DDDAS applications on mobile devices depend on data distribution management (DDM) in order to receive online data to drive simulation applications in order to make future state predictions. The reliance on DDM requires an understanding of how DDM affects energy consumption on these mobile devices. Gaining an understanding of the energy limitations of DDM on mobile devices make the use of DDDAS applications more appealing and applicable for situations such as natural disasters, emergency medical services, and improving transportation systems [2-4]. Thus far, very little work has examined the energy consumption properties of data-driven distributed simulations and DDM implementations.

A distributed DDDAS system requires a significant amount of data communication. Thus it is important to understand how this communication will affect energy consumption. We are particularly concerned about utilizing DDDAS to predict or/and improve vehicle transportation systems by utilizing real-time and online traffic data to drive the embedded traffic simulation within the DDDAS application. Our concern specifically lies in the area of energy consumption from running such applications in mobile environments and understanding the energy consumed by components of such a system.

Transportation systems are essential for people to navigate through a town or city. Commuters typically want the most efficient route possible and those routes may change under different constraints such as time of day, location, type of transportation system, and weather conditions. Traffic simulations give users the ability to understand the future state of the transportation system and make a decision concerning what route to take. Traffic simulations typically utilize historical traffic data; but, with the expansion of the Internet of Things and sensors real-time data can be used to drive traffic simulations to give a more accurate picture of future states of the transportation system.

Interest management is concerned with routing information from data sources to data consumers. Interest management avoids broadcast communication used in early distributed simulation systems [5]. Receiving agents (federates) declare their interest in receiving certain data produced by data sources. Data sources are responsible for declaring information concerning the data they produce, and the interest management system is responsible for matching data interests with the data that is produced.

Our focus in interest management concerns the Data Distribution Management (DDM) services defined in the High Level Architecture. In the context of the transportation simulation application these services are used to efficiently communicate state information among federates (sensors and embedded traffic simulations) in order to make accurate updated state predictions of the traffic network as whole. DDM is a set of services defined in the High Level Architecture that provides efficient communication for distributed simulations. Known DDM approaches are suitable for distributed simulation environments that are not energy dependent. We are concerned with understanding how to implement and use DDM to provide real-time information for embedded DDDAS applications including mobile devices. In energy constrained environments the question becomes how to implement and use the DDM services taking into consideration energy consumption. Energy consumption in DDM has not been explored previously.

Our motivating application is a DDDAS for transportation system management that utilizes embedded traffic sensors, providing real-time data for mobile devices that make predictions about the future state of the traffic network. Our interest lies in the energy constrained components of such a system, specifically, embedded traffic simulations and the DDM system used to communicate real-time sensor data and information shared among the simulations. In the case of our motivating application our federates include the mobile devices distributed within the environment receiving data from traffic sensors within the traffic network, the mobile devices are responsible for requesting traffic data from sensors in which they are interested in receiving data from. The mobile devices are then responsible for using the received data as input into the embedded traffic simulations.

Energy consumption in this application can be broken down into that used for communication and that used for computation. Communication involves transmitting real time data between data sources and mobile devices that host the embedded traffic simulations. Questions that arise in such an application include the approach used to aggregate and send data and determining who receives what data and when. These concerns are addressed by the interest management system.

A second major focus of this research concerns the embedded traffic simulations operating within mobile devices. We explore the energy required by embedded data-driven traffic simulations for communication and computation. Communication concerns include the amount and frequency to transmit data to the embedded simulation. Computation concerns include the type of traffic model that is used and how the operations needed to run the applications. Data structures used to represent the traffic system play a major role in the amount of energy that is consumed by embedded traffic simulations when run on mobile devices. We explore the energy requirements of a traffic simulations based on cellular automata and an event driven queuing model for traffic flow. The sections follow discuss these topics in greater detail.

1.1 Literature Survey

Simulation is the act of mimicking a real world system or process over time. Simulation models the system under study by representing the physical processes that occur in that system using characteristics, behaviors, and functions. Simulation is often used to optimize operations, testing, education, and etc.

1.1.1 Discrete Event Simulation

Discrete event simulation models system operations using events that reflect operations that occurred during each distinct moment in time. Events are responsible for changing system state variables to reflect the state of the system at the time the event occurs. Discrete event simulation models typically include several software components: state variables, simulation clock, future event list, event procedures, and random number generators.

The state of the system is represented using state variables. These variables pertain to different aspects of the system of interest to the modeler. In the case of traffic simulations these state variables would represent quantities such as the location of vehicles in the network, state of traffic signal etc. Other state variables contain information used to compute metrics that are used to produce the outputs produced from the simulation such as the volume of vehicles in the system, average vehicle speed, etc.

The simulation clock is responsible for keeping track of the current simulation time. Simulation time is represented using a selected unit of measure. The clock is advanced to the time of an event when that event is processed. The amount of simulation time does not directly correspond the actual running time to execute the simulation model. Events that are waiting to be processed are stored in a data structure called the future event list. This list is responsible for holding all the events that are to be executed. As events are processed they add (schedule) new events into this list. These events are typically prioritized by timestamp and stored in a priority queue data structure.

Random number generators are used in discrete event simulators in order to generate random variables for system values such as vehicle inter-arrival times. These values are often selected using a given probability distribution to generate random values. Utilizing pseudo random number generators allows repeated executions of the simulation to generate identical results from one run to the next.

Many of the above components are included in the simulation engine which usually includes the main loop of the simulation. The engine initializes the system state variables, simulation clock, and schedules the initial simulation events. The simulation engine is responsible for removing events from the event list as they are processed [6].

One of the most widely used discrete simulation models is the queuing model. Queuing systems model service operations by utilizing one or more servers providing service to customers arriving in the system. The components that encompass a queuing system include the arrival process, service mechanism, and queue discipline. The arrival process characterizes the inter-arrival times of customers entering the system. The service mechanism describes the number of servers in the system and the probability distribution used to determine the service time. The queue discipline defines how the server selects the next customer, e.g., first-in-first out, last-in-first-out, or according to some prioritization scheme.

1.1.2 Traffic Simulations

Traffic simulations are used to analyze vehicle transportation systems. The simulation model represents behaviors and functions of the system that researchers use as a tool to gain insight, find issues, and optimize the real world traffic network. Researchers are able to use variables that represent components of the transportation system to describe the state of the network. Traffic simulation models are described as either discrete or continuous models. A discrete model often represents the system using an event driven approach. A sequence of events reflect the total operation of the whole system over time and each event reflects a distinct point in time in which the system variables are changed to reflect the state of the system at that point in time. A continuous model represents the system where state changes occur continuously over time. These models are usually represented using differential equations.

Models utilize different levels of detail to reflect the dynamic behaviors of the system. Macroscopic, microscopic, and mesoscopic are the most widely used approaches. Macroscopic models view traffic as continuous flows. Microscopic models capture the behavior of individual vehicles and mesoscopic models capture the behavior of small groups of vehicles within the system but represent individual vehicles within the model.

1.1.2.1 Macroscopic Simulation Models

The macroscopic perspective of vehicle traffic simulations models traffic by using aggregated characteristics to reflect the state of the system such as the number of vehicles within the system. This is a high level perspective. Individual vehicle behavior is not represented. This perspective is used to model traffic flow in many different kinds of traffic networks. Continuous simulation models systems by utilizing differential equations to represent continuous changes to the system with respect to time [7]. Continuous traffic simulation models are often used to solve problems such as determining bottlenecks in traffic networks by using neural networks to determine real time traffic signal control [8, 9]. MASTER is a macroscopic traffic simulation model that is used to develop continuous traffic simulations based on gas kinetics and non local traffic [10]. Systems that support the development of macroscopic simulations include NETVACT, FREEFLO, CORFLO, KRONOS, METACOR, and AUTOS [11] .

1.1.2.2 Microscopic Simulation Models

Traffic simulations using the microscopic perspective are considered a fine-grained simulation approach that models the micro level dynamics of the vehicle traffic system. This approach models individual vehicles in the simulation. This approach typically uses car following models to represent the behavior of individual vehicles.

Cellular Automata simulations are microscopic simulation models. Cellular automata operate by implementing a two-dimensional grid structure that defines cells by state values [12]. Each individual vehicle in the system occupies a cell within the grid structure; when a system update is made each cell's new state is determined based on the state of its neighboring cells. Nagel and Schreckenberg created a well-known cellular automata single-lane model that divides the road into cell segments. Each cell has a state that is considered occupied if a vehicle is present or empty if one is not. The state of the cells change on every iteration based on the neighboring cells surrounding it [13]. Esser and Schreckenberg implemented an urban traffic network based on Nagal's original model [14]. Statistics such as throughput, travel time and individual vehicle speed and location are computed in the model. Cellular automata models derive macro level traffic flow behavior from micro level dynamics.

Microscopic traffic simulations are used to plan for many different types of situations. For example emergency evacuation, evaluating and understanding ramp control for freeway traffic control, and to manage dynamic traffic of often studied using microscopic simulation [15-17]. Well known microscopic traffic simulation systems include but are not limited to CARSIM, CORSIM, AVENUE, MITSIM, SIGSIM, and SIMNET [10].

1.1.2.3 <u>Mesoscopic Simulation Models</u>

The mesoscopic approach, like the microscopic model, is a low level perspective. Traffic simulation models using this perspective are viewed using small groups. Mesoscopic traffic simulation models view traffic using macro and micro level dynamics. Mesoscopic models can deal with individual vehicles as in microscopic models and are concerned with vehicle dynamics. Mesoscopic models follow two main approaches. One models vehicles in groups [18]. Synchronous timing is often used where time is advanced based on the chosen time unit or time step. In event driven mesoscopic models the state of the model changes when an event occurs. Time is advanced by some variable amount depending on what occurs during the event. Mesoscopic models are considered more flexible than macroscopic models due to their ability to model individual vehicles, but are still limited in modeling detailed traffic operations [19]. Mesoscopic models are typically more computationally efficient than microscopic models.

Systems such as DynaMIT, Dynemo, DYNASMART, and Metropolis are examples of mesoscopic transportation simulation models [10]. Dynameq and MEZZO are event based mesoscopic models [18]. MesoTS is a mesoscopic traffic simulation model used for predicting traffic conditions [20]. MESCOP is a mescoscopic traffic simulation model that evaluates and optimizes signal control plans [21]. This model aims to relieve the computational burden that occurs when optimizing complex control logic to create plans for traffic signals.

1.1.3 Distributed Simulations

Distributed simulations are simulation models that are executed on multiple processors in order to speed up overall execution or to execute large simulations that require more memory than is available on a single system. The simulation model is typically broken up into logical processes and each processor is responsible for executing some number of logical processes. Processors communicate utilizing messages to communicate their system state to other processors in order to obtain the state of the overall simulated system. Each logical process is responsible for maintaining its own state variables and simulation clock. Synchronization and data distribution are used to create a distributed simulation that produces the same results as a sequential execution of the same simulation program.

Synchronization mechanisms are used ensure the distributed simulation produces the same output as a sequential execution of the simulation. Synchronization mechanisms are categorized as conservative or optimistic. Conservative synchronization uses blocking mechanisms such as deadlock avoidance, synchronous algorithms, and deadlock data detection and recovery in order to achieve synchronization. For example, Chandy and Misra developed a deadlock prevention mechanism using null messages [22].

Optimistic synchronization such as Time Warp detect out of order event executions and roll back the computation to recover from synchronization errors [23]. Under this mechanism processes execute without regard for out of order event processing. Once an error is discovered the logical process is rolled back to a time before the error occurred.   Virtual time is synonymous with simulation time and is a global one-dimensional coordinate system. Systems that use virtual time follow two fundamental rules [24]:

1. “The virtual send time of each message must be less than its virtual receive time [24]. “

2. “The virtual time of each event in a process must be less than the virtual time of the next event at that process [24].”

Data distribution is used in distributed simulation to communicate data such as state information among different objects making up the distributed simulation. An important difference between message passing systems is their use of synchronous or asynchronous message passing. In synchronous message passing communication happens between objects that must coordinate with each other on each communication. Synchronous message passing is typically used in object-oriented programming languages such as Java. With asynchronous message passing it is possible for the receiving object not to be executing when the sending object sends the message. In asynchronous message passing storing and retransmitting data requires additional operations. Synchronous message passing is comparable to a function call in which the message sender is the function caller and the message receiver is the called function. In synchronous message passing the sending procedure of a message does not stop until the receiving procedure receives the message. The implications of the sending and receiving procedure of synchronous message passing makes it unusable for some applications.

Asynchronous message passing, unlike synchronous message passing, does not wait for a response from the receiver before continuing the execution of their procedure. Under asynchronous message passing the receiving procedure computes new values in order to send the results to the receiver, the message is buffered until the receiver is free to receive the message.

1.1.4 Interest Management

Interest management is a mechanism used in distributed simulations to distribute state information among entities in the simulation. For example, it was used in the Distributed Interactive Simulation (DIS) protocol [5]. Entities specify the information the wish to receive and data producers characterize the contents of messages they send. Interest management is used in large-scale distributed simulations in applications such as military war gamin and training systems. Morse created a taxonomy for such systems [5]. Data Distribution Service (DDS) [25] and the High Level Architecture (HLA) [7], are examples of systems using interest management.

Data Distribution Service is defined by the Object Management Group [26]. The DDS Data Centric Publish Subscribe (DCPS) interface focuses on delivering information to specified recipients. DDS uses participant, writer, reader, publisher, subscriber, and topic entities. They are classes, extended from the DCPS interface. The publisher is responsible for data issuance and may publish data of different data types. The writer is used in order to communicate data values and changes. The subscriber receives all the published data and makes sure it is available for the participant. The reader then has access to the received data.

1.1.5 High Level Architecture (HLA)

High Level Architecture (HLA) is a general-purpose architecture that was created to promote and support interoperability amongst different simulations. The Defense Modeling and Simulation Office (DMSO) defined HLA. It is maintained by the US Department of Defense (DoD) [7]. The designers of HLA had five principle goals behind the architecture [27].

1. "It should be possible to decompose a large simulation problem into smaller parts; smaller parts are easier to define, build correctly, and verify [27]."
2. "It should be possible to combine the resulting smaller simulations into a larger simulation system [27]."

3. “It should be possible to combine the smaller simulations with other, perhaps unanticipated simulations to form a new system [27].”
4. “Those functions that are generic to component-based simulation systems should be separated from specific simulations [27].” “The resulting generic infrastructure should be reusable from one simulation system to the next [27].”
5. “The interface between simulations and generic infrastructure should insulate the simulations from changes in technology used to implement the infrastructure, and insulate the infrastructure from technology in the simulations [27].”

Individual simulations are called federates and a distributed simulation system is called a federation under HLA terminology. HLA is defined by three IEEE standards: the framework and rules, the interface specification, and the object model template.

The high-level architecture rules describe the underlying principles used in HLA. The rules are in place to ensure that simulation models follow the HLA standard. The rules are quoted below [28]:

1. “Federations shall have an HLA federation object model (FOM), documented in accordance with the HLA object model template (OMT) [28].”
2. “In a federation, all representation of objects in the FOM shall be in the federates, not in the run-time infrastructure (RTI) [28].”
3. “During a federation execution, all exchange of FOM data among federates shall occur via the RTI [28].”

4. “During a federation execution, federates shall interact with the run-time infrastructure (RTI) in accordance with the HLA interface specification [28].”
5. “During a federation execution, an attribute of an instance of an object shall be owned by only one federate at any given time [28].”
6. “Federates shall have an HLA simulation object model (SOM), documented in accordance with the HLA object model template (OMT) [28].”
7. “Federates shall be able to update and/or reflect any attributes of objects in their SOM and send and/or receive SOM object interactions externally, as specified in their SOM [28].”
8. “Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM [28].”
9. “Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM [28].”
10. “Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.”

The object template model describes the information that is communicated between simulations. HLA standards require that every federate have an object model. These object models are represented using a set of tables. The object model template specifies how the information must be recorded in the object model. The HLA requires that a federation object model (FOM) be defined that describes the shared object classes, attributes and interactions used in the federation. The FOM uses the management object model (MOM), which holds the collection of classes and interactions. In the currently implemented standard of HLA 1516-2010 this information is formatted using a XML file.

All participating federates must use the same FOM. The simulation object model (SOM) describes the shared object attributes and interactions used for a single federation.

The interface specification defines how participating simulators interact using the run-time infrastructure (RTI). The RTI provides an API that complies with the interface specification. The rules are in place to ensure that simulation models follow the standard. The interface specification defines six categories of services: federation management, declaration management, object management, ownership management, time management, and data distribution management. Federation management is responsible for defining how the federation is created and managed, e.g., services allow new federates to join the federation. Declaration management services are used by federates to declare their intentions to publish and subscribe to data and interactions. Object management services manage how federates manage, e.g., update objects that they "own." Ownership management services enable federates to gain or divest ownership of objects. Time management services specify how federates use time in regards to object interactions and updates. Data distribution management defines the services concerning the routing of data among federates in large-scale simulation executions.

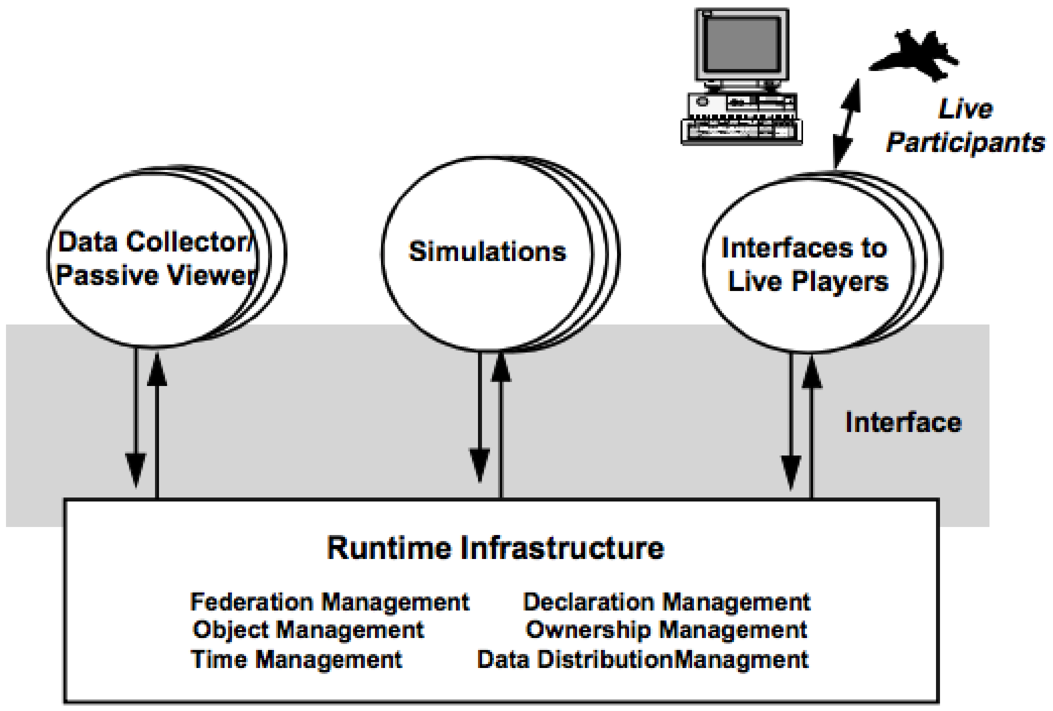

Figure 1-1: HLA Federation Structure [29]

The major steps in the execution of a federation include: initialization, declaring the objects of common interest among federates; exchanging information and terminating execution.

1.1.6 Data Distribution Management

Data Distribution Management services are used to reduce traffic flow over the network. DDM is one category of services defined in the High Level Architecture Interface Specification. The services are implemented by Run-Time Infrastructure software. DDM utilizes an N-dimensional coordinate system called a routing space to represent, for example, a geographical area. Federates express their interest by defining subscription regions that characterize the information they are interested in receiving, and messages are associated with a publication region to characterize the content of the message. If an overlap is detected between a publication region and a federate's subscription region, the message is sent to that subscribing federate.

The region based and grid based DDM approaches are the most well-known approaches to implementing DDM services. Several approaches have been proposed and implemented to overcome the drawbacks of these approaches. The hybrid based approach reduces the communication cost of the grid based approach and the matching cost of the region based approach by performing direct matching between publication and subscription regions within grid cells [30]. This approach helps to alleviate the irrelevant message problem that occurs in the grid based approach, but, duplicate messages can still occur. Boukerche and Roy proposed the dynamic grid based approach whose goal is to reduce the number of multicast groups created under the grid based approach [31]. Under this approach multicast groups are only allocated to a cell if an intersection is detected between the publication and subscription region [31]. Boukerche also proposed the grid filtered region based approach [32]. This approach combines the grid based and the hybrid based approach by using a threshold parameter to determine when matching needs to be performed between regions within a grid cell who's area of coverage of the grid cell falls below the given threshold [32]. The effect of grid cell size on multicast group assignment and communication cost has been explored by Van Hook and Rak [33]. Their findings concluded that irrelevant and duplicate messages are increased with the use of larger grid cells [33]. Many other approaches and mechanisms have been created to overcome the drawbacks of purely region based and fixed grid based approaches. Adaptive data distribution management optimizes DDM time, matching, and multicast group assignment [34]. Raczy also created the sort based DDM matching algorithm [35].

This algorithm sorts the coordinates on each extension. An extension of the sort based algorithm was also created to promote efficiency in large spatial environments [36]. Pan also created the dynamic sort-based approach [37]. This approach efficiently matches for selective region modifications.

1.1.7 Dynamic Data Driven Application Systems (DDDAS)

Dynamic data driven application systems are a type of on-line system that incorporates real time data into the executing system to improve the accuracy of the model, speed up the execution, and/or utilize the executing application to guide the measurement process [1]. DDDAS applications may be embedded within the physical system being monitored in order to utilize real-time data near its source. For example, embedded traffic simulations may be part of a sensor network where real-time traffic data is used as input to drive transportation simulations.

The DDDAS control loop first incorporates real-time data into the executing application. The executing application uses this data to measure and analyze the behavior of the physical system. The embedded simulation makes future state predictions in order to develop recommendations to modify the system or instrumentation. The DDDAS control loop then repeats this process. An example of this cycle is illustrated in Figure 1-2.

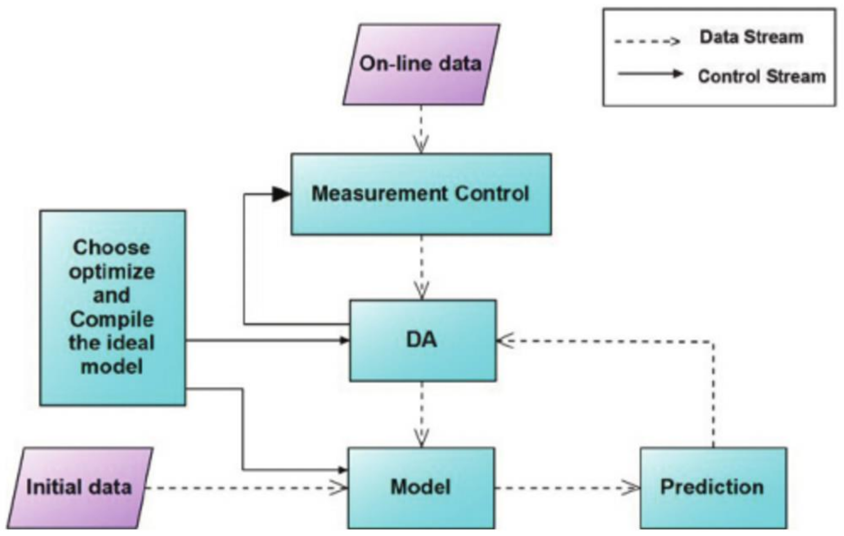

Figure 1-2: DDDAS control loop [38]

Wahle, Neuber, and Schreckenberg used a cellular automata traffic flow model for online simulation[39]. Their model gives users the ability to interpolate the traffic state between different check points and gather inform about areas that are not well instrumented [39]. This allows one to make improvements concerning the accuracy of traffic network predictions.

Monitoring ecological development [2], forest fires [40], and tracking multiple targets in an ad-hoc sensor network are examples where a DDDAS system is embedded within the physical system [41]. Other examples include ocean forecasting [42], hurricane forecasting [2], emergency medical services [3], and optimizing surface transportation systems [43]. In situations where battery-powered mobile devices are used as the DDDAS platform energy consumption by DDDAS computations and communications is an important issue.

1.1.8 Power And Energy

The growing computation and communication capabilities of mobile devices make them attractive platforms for DDDAS applications. Mobile devices can provide a wide variety of sensor capabilities such as video, still images, audio, GPS, etc. An important concern for such devices is power and energy consumption. Mobile devices are typically dependent on battery power. As such, it is important to understand the energy consumption and computing limitations of such devices. Our concerns here is the energy consumed for communicating dynamic data to drive DDDAS components embedded within the mobile device and the energy consumed for traffic simulations.

Dynamic voltage frequency scaling (DVFS) is a power saving technique. DVFS reduces the frequency at which the processor is operating which in turn reduces the power consumption of the system. Weiser was the first to propose lowering CPU frequency when clock cycles are being wasted [44]. Weiser later improved his initial approach by incorporating a performance monitoring unit which counts the instructions and memory requests per cycle to predict the system's workload and allow a more energy efficient frequency to be selected. Other researchers improved upon Weiser's approach by creating a framework to automatically select the best model parameters from the hundreds of possible events that modern PMU's could create, resulting in a saving of up to 20% in energy consumption [45].

Communication is a significant source of energy-consumption in mobile systems. Researchers have developed energy aware network protocols to reduce energy consumption. Mobile environments may utilize MANETS, i.e., mobile ad-hoc networks where mobile nodes communicate directly without the assistance of a preexisting infrastructure. Energy aware routing protocols aim to reduce the amount of energy expended when routing data between devices in MANETS. For example, Singh, Woo and Raghavendra investigated the effects of a shortest path routing algorithm that was reported to achieve a 40 – 70% reduction in energy consumption [46] . Their technique consist of using new power aware metrics to determine the best routes in MANETS based on battery power consumption at the mobile nodes [46]. Other protocols include the EE-OLSR protocol which aims to improve the energy consumption of the traditional OLSR protocol [47]. Researchers have also conducted analyses of traditional routing protocols such as DSR, AODV, TORA, and DSDV with respect to energy use. Their study shows that under pure demand conditions that DSR and AODV perform the best in MANET networks for mobile devices [48].

Power and energy have been heavily studied in several areas. Energy aware practices have been developed to give programmers guidance concerning how they should program applications for mobile devices [49]. Carroll analyzed the power consuming components of mobile devices [50]. He examined the power consumption distribution among CPU, memory, touch screen, graphics hardware, audio, storage, and various network interfaces. He created a plethora of scenarios that reflect typical usages of a cellular device and measured the power consumption of the different components of the device under various conditions. His results showed that much of the device's power consumption is attributed to the GSM (global system for mobile communication) module and the display [50]. These results also showed that aggressive backlight dimming could save a great deal of power. The RAM, audio, and subsystems were the lowest power consuming components in his analysis, although in some cases the RAM can consume more power than the CPU. Here, we are concerned with software concerns considering that our application is applicable to different mobile hardware platforms.

Techniques to conserve energy for computations performed on mobile devices have been widely investigated. One technique includes off loading computation to the cloud to save energy [51]. Kumar and Lu investigated this concept and discovered that off loading computations to the cloud could possibly save energy [51]; but, not all applications will become energy efficient when migrating computation to the cloud. Designers of the system should consider the overhead that could occur from this technique such as privacy, security, reliability, and data communication before off loading. Energy scale down is another approach that researchers have explored as design mechanisms for mobile applications [49]. This approach involves both hardware and software scaling of features and energy use to meet design goals [52]. One determines energy scale down by having a high end and a low end design point. The designer aims their efforts toward the low end design point in order to use as little energy as possible. These designs points are determined by considering each component in the general purpose device and comparing it to the requirements of the applications using that device.

Energy for communication on mobile devices must be considered when designing applications. Energy costs for communications have been examined for peer-to-peer applications [53], network interfaces [54] , ad-hoc network [54], radio interfaces [55], and radio networks [56]. Studies comparing the energy consumption of network activity in 3G, GSM, and WiFi have found that tail energy is wasted in high power states after data transfers are complete [56]. To overcome this problem Balasubramanian and Venkataramani created the TailEnder protocol that reduces tail energy consumption by scheduling transfers to minimize the cumulative energy consumed while meeting user specific deadlines [56].

Energy profilers are often utilized in order to measure energy consumption of mobile systems. Trepn is an example of a software tool that was developed by Qualcomm to measure power of Android systems [57]. Recent work on profiling distributed simulations has been conducted [58]. Power Tutor is a software application that was developed to aid the design and selection of power efficient software for embedded systems [59]. The application informs users of power consumption to aid application design and use. WattsOn, like PowerTutor is a software application that allows developers to estimate the energy consumption of applications during development [60]. Utlizing techniques such as the energy foot print of mobile hardware systems [61], fine grained system trace calling [62], and self constructive approaches where mobile systems automatically generate their energy model without any external assistance through a smart battery interface have been conducted to gain an understanding of how energy is dispersed in mobile devices [61, 62] .

1.2 Research Contributions

This thesis focuses on the energy consumption of components of a DDDAS for embedded traffic simulations in mobile environments. This research considers the development and utilization of the High Level Architecture Data Distribution Management services to disseminate dynamic data to drive transportation system simulations.

1. **Energy comparison between queuing model and cellular automata based embedded traffic simulation [63].** A study comparing the energy consumed by an event driven queuing network model and a cellular automata traffic simulation for an arterial road network in a section of midtown Atlanta was completed to examine their energy consumption on mobile devices. This study showed the effect of levels of traffic flow and simulation size on the overall energy consumption of these traffic simulators.
2. **Demonstrated that data clustering can produce efficient energy consumption for communicating in mobile environments [63].** This work examined the effect that varying message size and aggregating messages on energy consumption when communicating data from mobile devices. These results demonstrate that communicating aggregated data for moderate sized message payloads can result in significant energy conservation in mobile devices.
3. **Energy comparison between four different priority queue implementations for future event list of discrete event simulations [64].** A study evaluating the implicit heap, explicit heap, linked list, and splay tree data structures used as priority queues for future event list was completed. This work represents the first

study of energy consumption for future event list implementations in discrete event simulations. The evaluations were made using the hold model. This study examined the key operations of each data structure and their effects on overall energy consumption. A model for predicting the energy consumed for a hold operation for the implicit heap, explicit heap, linked list, and splay tree data structures was developed. Each prediction model reflects the energy consumption to the energy causing components of each data structure and the energy constants associated with that behavior for priority queue sizes that can fit within the L1 cache of the embedded system hardware and those which cannot completely fit within the L1 cache.

4. **Created an energy efficient distributed region based approach [65].** A distributed region based data distribution management approach was developed that has the communication efficiency of the centralized region based approach without the reliance on a central controller. The computation efficiency for initial multicast group assignment and communication update efficiency for the distributed region based approach was analyzed relative to the centralized region based approach, fixed grid based approach, and grid filtered region based approach.
5. **An energy comparison of initial matching computation for multicast group association [65].** The effect of the initial computation on energy consumption in region based and grid based approaches was examined. These results show that grid based data distribution management approaches consume less energy

initially than region based approaches because a comparison against every region within the routing space is not needed.

6. **An energy evaluation of constraining grid cells for grid based data distribution approaches [65].** The affect that varying DDM components on energy for communicating DDM updates was examined. These experiments explored the effect of varying grid cell size, constraining publishing regions to one grid cell, and constraining publication regions to N grid cells on energy consumption. Experimental analysis shows that utilizing constraints on grid cells can gain communication efficiency approaching region based data distribution management approaches.

1.3 Organization of Thesis

This thesis is organized as followed. Chapter 2 details the work done in the area of traffic simulations and energy consumption. Here we present the simulations evaluated and experiments conducted for evaluating energy use. Chapter 3 details the work in the area of energy consumption of priority queues of future event list for discrete event simulations. Chapter 4 describes the work done in the area of HLA DDM approaches and energy consumption. We describe the details of the approaches compared and evaluated the energy consumption for computation and communication.

2 ENERGY FOR ONLINE DATA-DRIVEN TRAFFIC SIMULATIONS

Energy consumption is an on-going concern in mobile and embedded computing systems powered by the device's battery. With the growing use of real-time data for traffic prediction applications one must understand tradeoffs between energy consumption for communications and computations under certain performance and accuracy constraints in order to ensure effective operation. For example, question might concern the approach used to model the system and the amount and frequency with which data should be collected to drive the simulation computations. This information is necessary to develop power and energy aware techniques to optimize energy use.

This study examines the energy consumed by data-driven simulations in predicting future states of a transportation network. The system utilizes sensor data specifying traffic flow on various road segments as input and makes future state predictions of an arterial traffic network. The future state predictions may then be distributed to other simulations, e.g., using the ad-hoc distributed simulation approach in order to enable state predictions of the entire network [66]. Energy utilized by the system for simulation in the area of computation and communication is analyzed to gain an understanding of the energy consuming components of such a system in an energy constrained environment.

2.1 Data-Driven Simulation Architecture

Ad-hoc distributed simulation is defined as a set of simultaneously executing, autonomous simulations connected through a wireless network [66]. Each simulation is responsible for modeling a portion of the overall system determined locally by the simulator itself. Each simulator communicates state predictions to other simulations to model the system as a whole.

Each simulation is a logical process (LP) in conventional distributed simulation terminology and is executed on a mobile device. The mobile devices are connected through a wireless network. Each device is responsible for connecting to a sensor or sensors within the environment in which it operates in order to obtain local traffic state information. Each sensor collects data information such as speed, acceleration, and direction of vehicles that pass through its sensor range. Each sensor communicates this information to nearby mobile devices and the data is then used directly or is aggregated to be used as input for the embedded simulations within the mobile device. Predicted simulator states, e.g., future flow rates on various links may be then transmitted to other simulators.

Here, we focus on one simulator of an ad-hoc distributed simulation. We consider the energy used by the simulator and that of the communications used to drive the simulation and to communicate results produced by the simulator that are distributed to other simulations in the ad-hoc distributed simulation.

The proposed energy model represents the different components of a DDDAS ad-hoc distributed simulation. This model separates the energy consumption of the total system into three major components: data communication, data aggregation, and the embedded traffic simulations. The model illustrates each major component in such a system that affects energy consumption. The system depicted in Figure 2-1 collects data from sensors spread across the area under study, sends data that was either unaggregated or aggregated at the sensor to the mobile device and uses the data sent in order to drive the embedded simulations on the device to simulate an updated state of the traffic network.

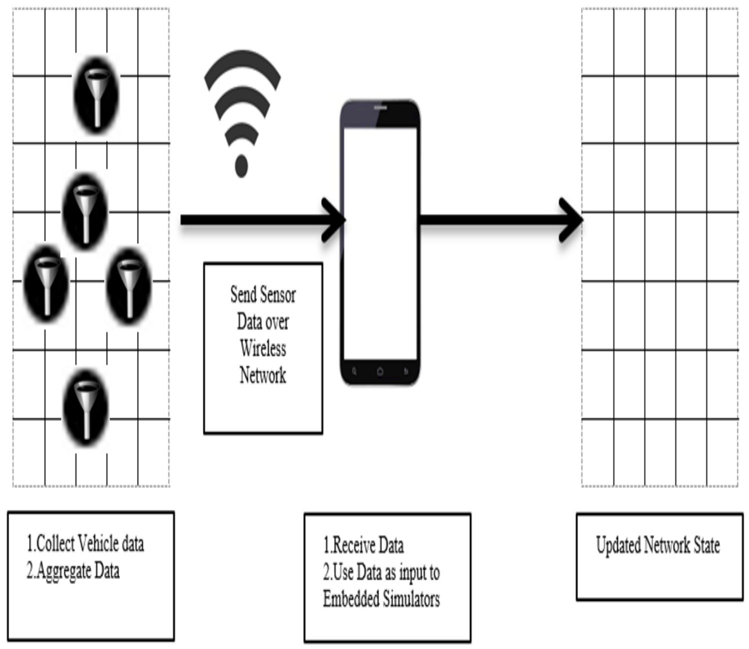
Send Sensor Data over Wireless Network
1.Collect Vehicle data
2.Aggregate Data
1.Receive Data
2.Use Data as input to Embedded Simulators
Updated Network State

Figure 2-1: Data Driven Simulation Architecture

Energy for data communication in the mobile device includes receiving data from the sensor network and sending predicted state information to other simulations (LPs). There are several options to transmitting data from the sensor network to the LP. Assuming data is sampled at some given rate, one could simply send each data update directly to the LP. Alternatively, the data samples could be collected in the sensor and periodically a collected set of measurements could be sent as a single message. Still another option is to aggregate the data within the sensor, and transmit an aggregated value, e.g., an average flow rate or the parameters for a probability distribution to the LP. Each of these options will result in different amounts of energy consumption in the system and will impact the results computed by the distributed simulation. For example, aggregating data within the sensor and sending the aggregated results will likely reduce energy consumption to transmit the data, but at the cost of providing less detailed information to the simulation and introducing delays before the online data can be incorporated into the simulation predictions.

The embedded data-driven simulations are responsible for making future state predictions of the traffic network. The amount of energy required by simulations may be significant, and requires exploration. The energy consumed by the simulation includes energy required by the CPU as well as energy used in the memory system and transmitting instructions and data between the two. These depend on the specific modeling approach that is used. Here, we focus on the energy used by transportation models using two widely used abstractions. As discussed momentarily, a model based on cellular automata is evaluated as well as a second based on queuing network abstractions implemented as a discrete event simulation.

2.2 Embedded Traffic Simulations

The cellular automata and queuing network models were configured to simulate the traffic of the arterial road network along Peachtree St. located in midtown Atlanta, GA (see Figure 2-2). This area was selected because of the availability of data. Specifically, traffic data from the NGSIM data set was utilized as the input to develop our simulation models [67]. The data was collected on November 8, 2006 during a fifteen-minute time frame from 4:00 PM to 4:15PM. The area includes five intersections, four that are signalized and six road segments. The data set consists of data pertaining to individual vehicle trajectories with time and location stamps, from which the link travel times of individual vehicles could be calculated. Figure 3 reflects a visual representation of the NGSIM data set area. In this study, link travel time refers to the time from when a vehicle enters the arterial link to the time when the vehicle passes the stop-bar at the end of the link. Intersection travel time is excluded.

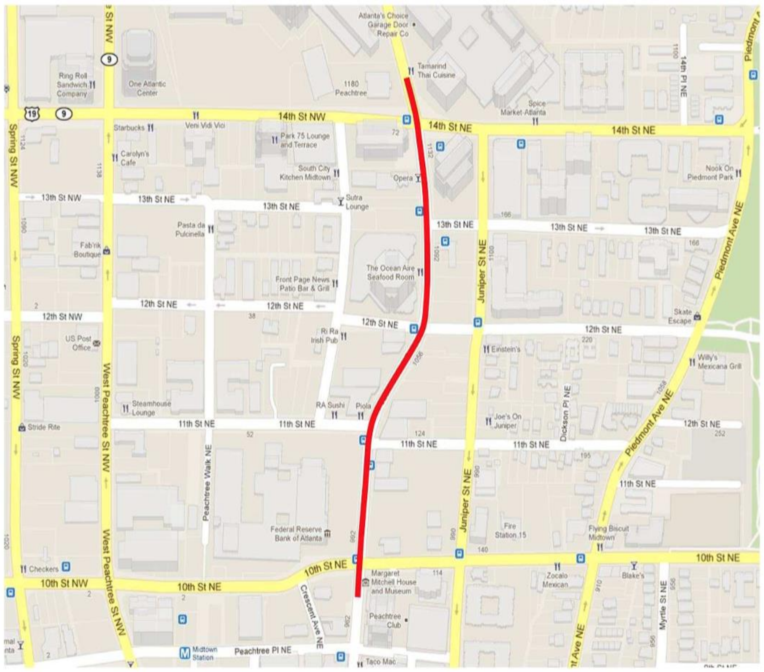

Figure 2-2: NGSIM Study Area

Both simulation models use the same input parameters and assumptions that were used as the basis for all simulations used in this study. Both models were developed in C and implemented as an Android native application. The output of each model is the average travel time for vehicles that are traversing the section of Peachtree St. described earlier. The model was validated by comparing the average travel times produced by the model to those observed in the NGSIM data set.

It was assumed that there were no pedestrians or emergency vehicles. Further the simulation excludes U-turns, aggressive driver behavior, adverse weather conditions, road construction, and vehicle accidents. Due to data limitations these aspects were not included in the models. The inter-arrival time of vehicles entering the simulated area were assumed to be independent and identically distributed following an exponential distribution. We assume that the destination zones of our model have unlimited capacity so that once a vehicle reaches its destination it departs from the system instantaneously.

The input parameters of each model include the historical traffic data collected from the NGSIM data set. Signal timings for each traffic light and probability of vehicle turns for each origin and destination zone were derived from the dataset. The parameters that were varied outside of the given parameters include the traffic intensity and the simulation time.

To simplify our model we assumed that all vehicles were the same length. We also assumed that all vehicles are identical, and travel with the same acceleration and maximum velocity parameters and had instantaneous deceleration. We assumed that the safe distance between vehicles is uniform for all vehicles.

2.2.1 Cellular Automata Model

Our cellular automata model was implemented in C using the two-lane cellular automata modeling approach. The two lane model approach was proposed and implemented by Rickert and Nagel [68]. The model consists of the following modules: cells, vehicles, and road segments. The simulation environment includes a two-dimensional array of 69 X 89 cells. Each cell was set to the size of 7.5 meters and can hold at most one vehicle at any instant. A cell can be in one of five states at any time: empty, normal, source, sink, or a traffic light. A normal cell is one that is a part of a road segment. A source cell represents a location where vehicles enter the system. A sink cell represents the location within the model where vehicles leave the system. A traffic light cell represents a cell where a traffic light is located. Vehicles are stopped based on the state of the traffic light and assigned a turn probability if applicable. An empty cell represents a cell that is currently not occupied by a vehicle. Each cell has a row and column location, id for the street on which it is currently located, the direction in which vehicles travel, and an array of turn probabilities for a vehicle that occupy that cell. Each vehicle has an id, vehicle arrival time, departure time, total time in the system, arrival street, and departure street. The vehicles velocity corresponds to the number of cells the vehicle can proceed forward.

The overall system executes in a time-stepped manner. The tick time pertains to the overall simulation time in seconds. For each time step vehicles are added to the simulation system and traffic lights are updated. Each road segment of cells pertaining to the vehicle lanes are evaluated in an s-shaped pattern checking vehicles against the flow of traffic. Each road segment begins its evaluation at the end of the road allowing vehicles closest to exiting the road segment the ability to move first. This then allows the vehicles behind it to have the ability to move forward since once they evaluate the cell before them it is considered empty. Vehicles that have the possibility to move forward to the next traffic cell are moved to the next available cell. A vehicle has the ability to move if the next forwarding cell is empty. If the forwarding cell is a cell pertaining to an intersection the vehicle turns in accordance with the intersection's turn probability. These probabilities are pre-computed and hard coded into the simulation from an input file based on the data from the NGSIM data set. A vehicle moves forward a set number of cells based on the vehicle's current velocity. As long as the vehicle's velocity is below the maximum velocity for all vehicles the vehicle accelerates v + acceleration steps ahead in the system so long as room permits for that number of cells for the vehicle to proceed. If the vehicle is not able to proceed v = v + acceleration steps ahead it proceeds to move as many cells as it can towards the value v that are available. If a vehicle reaches a cell that contains a traffic light, a vehicle is now in an intersection and the vehicle is assigned a turn probability, which is preset at initialization of the traffic network based on the data from the NGSIM data set. If a vehicle is assigned to turn its direction property is changed and it now proceed in that direction.

Intersection traffic light states are updated each time step. The state of the light changes based on the length of the phase of each state as determined based on the information provided from the NGSIM dataset. This sequence continues until the simulation cycle is completed.

2.2.2 Queuing Model

In the queuing model simulation traffic lanes are represented using queues that hold vehicles occupying the lane. The model is event driven. Events with smaller timestamps are processed first and continued until all events have been processed or the simulation has completed.

The discrete event queuing model was also implemented in C. The model consists of the following modules: simulation engine, simulation application, event, vehicle, intersection, section, priority queue, and linked-list and implements a standard event-oriented execution paradigm. The model is driven by the simulation engine which holds the main loop that continuously executes until no events remain or the set simulation end time has been reached. The priority queue is implemented using a binary heap. The simulation application module is responsible for initialization of system variables that start the simulation and calculating output values such as the average travel time. The simulation application is responsible for processing the callback functions and event handlers that implement event processing routines. The event handlers include global arrival and departure events and events for each intersection that handle vehicle events which include: arrival, entering, crossing, and departing.

Traffic signal changes are also a part of intersection events. These events are responsible for switching the state of the traffic light when called. Events are created using the event module, which creates an event object. Each object has the attributes of an event type including the timestamp and callback function. Vehicles are created using the vehicle module. Each vehicle created has a set origin, destination, id, lane id, and velocity. Section modules represent road lanes between traffic intersections. Each section module object maintains values to attribute to the number of vehicles occupied in each section and a flag indicating congestion. Intersection modules hold attributes corresponding to a road network intersection. Each lane of the intersection is represented using a queue into which vehicles are placed once they enter each intersection. The intersection module is also responsible for handling traffic light signal changes where signal lights states are based on phase lengths. If a vehicle is within an intersection during a green light phase, vehicle events are scheduled to progress the vehicle forward to the next street section of the system.

2.2.3 Embedded Simulations and Energy Consumption

The embedded simulations represent the main computational portion of the DDDAS system. Each model is responsible for modeling the vehicle throughput of the arterial network. The cellular automata model must update the position of each vehicle every time step in the simulation. The queuing model is event driven and does not need to process state updates of each vehicle so frequently. However, a priority queue is needed to hold the set of pending events, and a significant amount of energy must be expended inserting and removing events.

Experiments were completed to measure the energy consumption as vehicle arrival rate (Figure 2-3) and simulation size (Figure 2-4) are varied. Energy was measured using the Trepn profiler app installed on an Android LG Nexus 5x cellular phone [57]. It is seen that the cellular automata model consumes more energy than the queuing model in these experiments. The cellular automata model must access each vehicle within the road segments each loop iteration causing the need for more computation operations to be performed resulting in larger energy consumption.

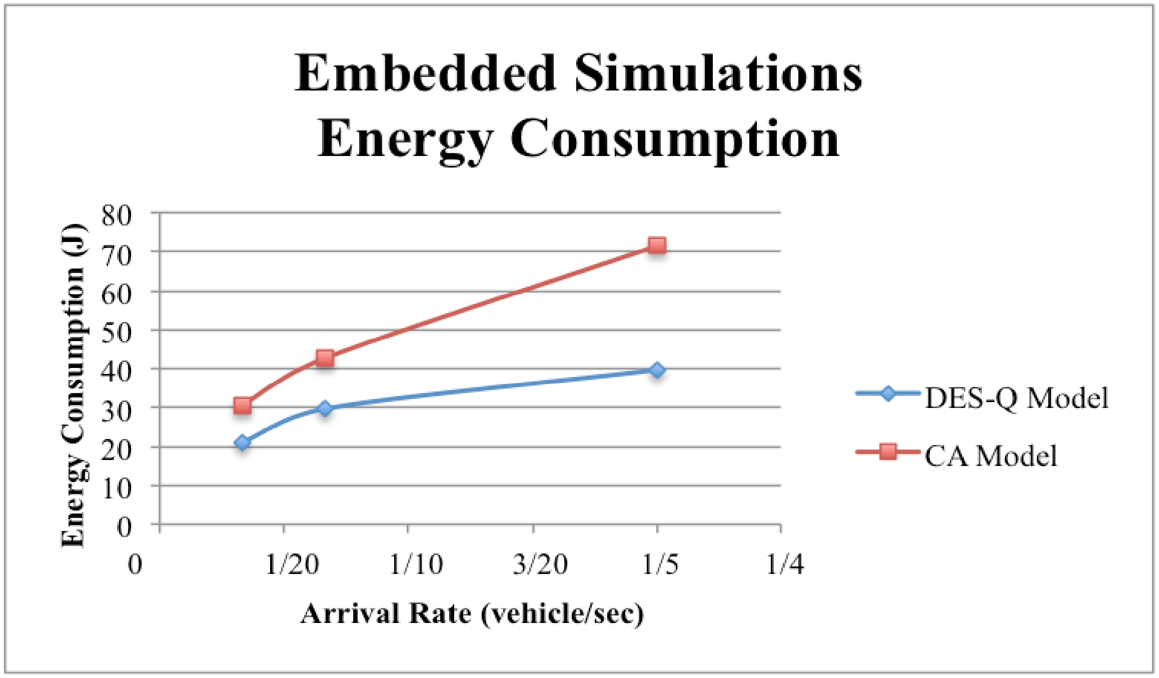

Figure 2-3: Energy as Traffic Load Varies

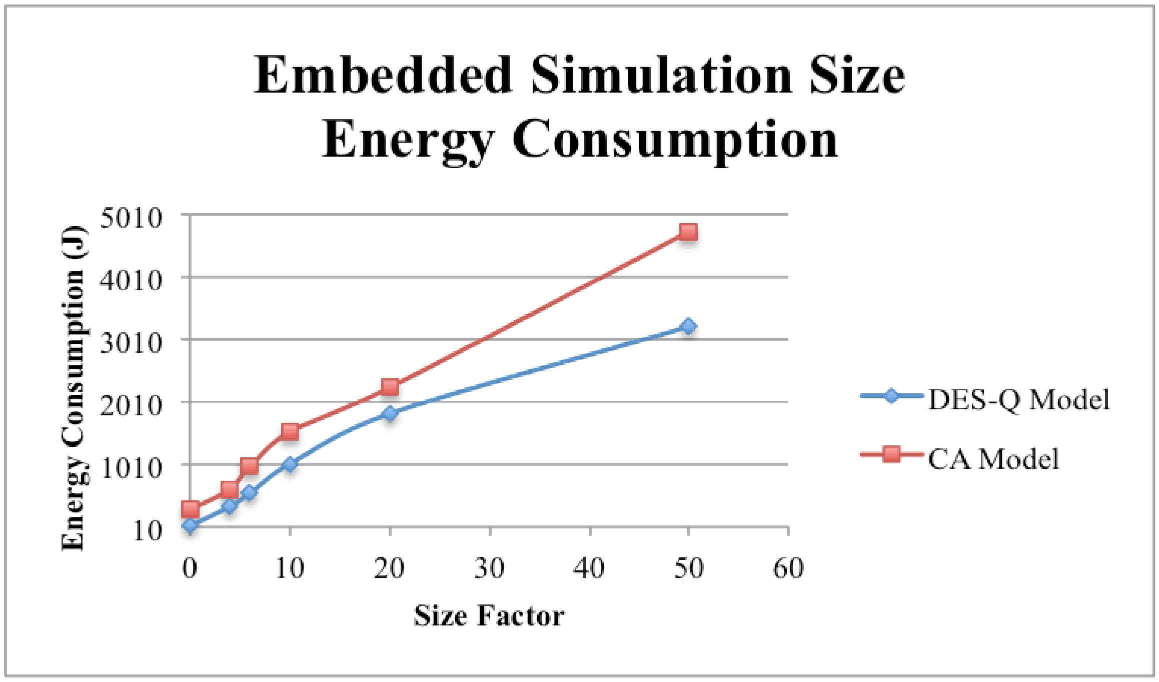

Figure 2-4: Energy as Simulation Size Varies

These results quantify the energy cost of using a more detailed model. Figure 2-3 indicates that increasing the inter arrival rate of vehicles in the system results in an increase in energy consumption due to the increased number of vehicles in the system. Energy consumption in the cellular automata model is impacted to a larger degree than the queuing model as the arrival rate increases. A larger arrival rate results in more vehicles residing in the system. As the cellular automata model must update the state of every vehicle in the system every time step and make updates according to its neighboring cells an increase in arrival rate should reflect an increase in energy. Whereas the queuing model must only process events for vehicles at the front of each queue that have been scheduled at each iteration. This impact of additional vehicles on energy usage will be less.

Figure 2-4 shows the results of increasing simulation size. In the cellular automata simulation the number of cells increases in proportion to network size. Our results show the original network size based on the configuration of the area under study and simulations of areas a factor of four and six times as large. For these experiments the network was replicated by the set input parameters to increase the number of cells in the cellular automata model, and to increase the number of queues and events in the queuing discrete event model. All instances of this experiment use an arrival rate of 1 vehicle every 5 seconds.

2.3 Communication

The data streaming and data aggregation models were written as a Java Android application. This application mimics communications of data between sensors and the distributed simulations through a wireless network.

2.3.1 Data Streaming

A data streaming application was created that is composed of a TCP server socket that communicates to TCP clients sockets over the wireless network. The server socket creates a thread that controls the execution of communication between the server socket and client socket. The server thread is responsible for establishing a connection with the client socket through a given port. Once the connection is established a thread is created to either send or receive data.

The receive thread is responsible for receiving data through the port in which a connection has been established. The received thread establishes an input stream in order to accept data streams sent from the client. The thread continuously accepts data until it is interrupted which occurs if a connection is lost.

The send thread is responsible for sending data through the port in which the connection has been established. Like the receive thread the sending thread establishes a connection from the client in order to begin sending data. The thread continuously sends data until the connection with the client has ended.

2.3.2 Data Aggregation

Data aggregation is the process of gathering data into summary form. For the purpose of this work data aggregation was used in order to aggregate traffic information in order to drive the embedded traffic simulations. The data aggregation process was set to aggregate information on the client side, which is the sensor in this case.

Aggregating data on the client side assumes that the client is a part of the sensor network. The client collects traffic information and as the information is collected aggregates the values. The aggregated values involve summarizing the number of vehicles and the arrival rates of vehicles. These summarized values are then sent to the DDDAS application over the wireless network. The DDDAS application has the option of receiving the values in a set interval fashion. This option alleviates the need to receive continuous data until the sensor has gained enough information to aggregate or the level of traffic is not heavy enough for the change in summary values to occur.

2.3.3 Data Communication and Energy Consumption

Experiments were performed to evaluate energy consumption. The experimental setup is intended to mirror a DDDAS embedded traffic simulation system where the execution process includes the system receiving real-time sensor data that is aggregated and used as input in the embedded traffic simulations. The simulations produce future state predictions that are sent to other simulations making up the ad-hoc distributed simulation. All energy consumption measurements were evaluated utilizing direct measurements from the Trepn profiler app installed on an Android LG Nexus 5x cellular phone. The profiler was utilized with the Delta settings enabled, which allows the application to collect power data of the entire system during a baseline interval. The average power value is then determined and subtracted from power values determined for the running application in order to give an accurate power measurement of the application.

Data streaming experiments were conducted utilizing the Android phone as the server with the DDDAS application installed on the mobile device. All communication occurred through a WLAN network using the 802.11g protocol. A laptop was utilized to represent a sensor in the sensor network and communicated collected sensor data to the mobile device. The experiments show how energy consumption varies with message sizes. The results show measurements when sending and receiving data continuously, and sending data at different payloads between the mobile device and the laptop (sensor).

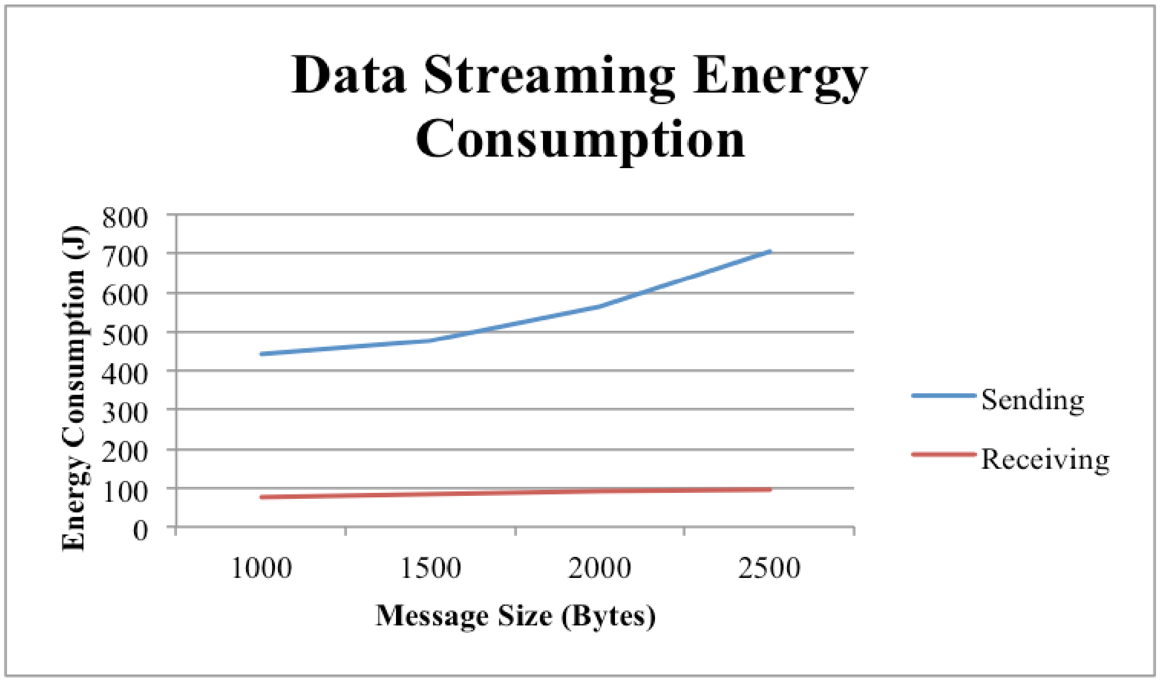

Figure 2-5: Data Streaming Energy

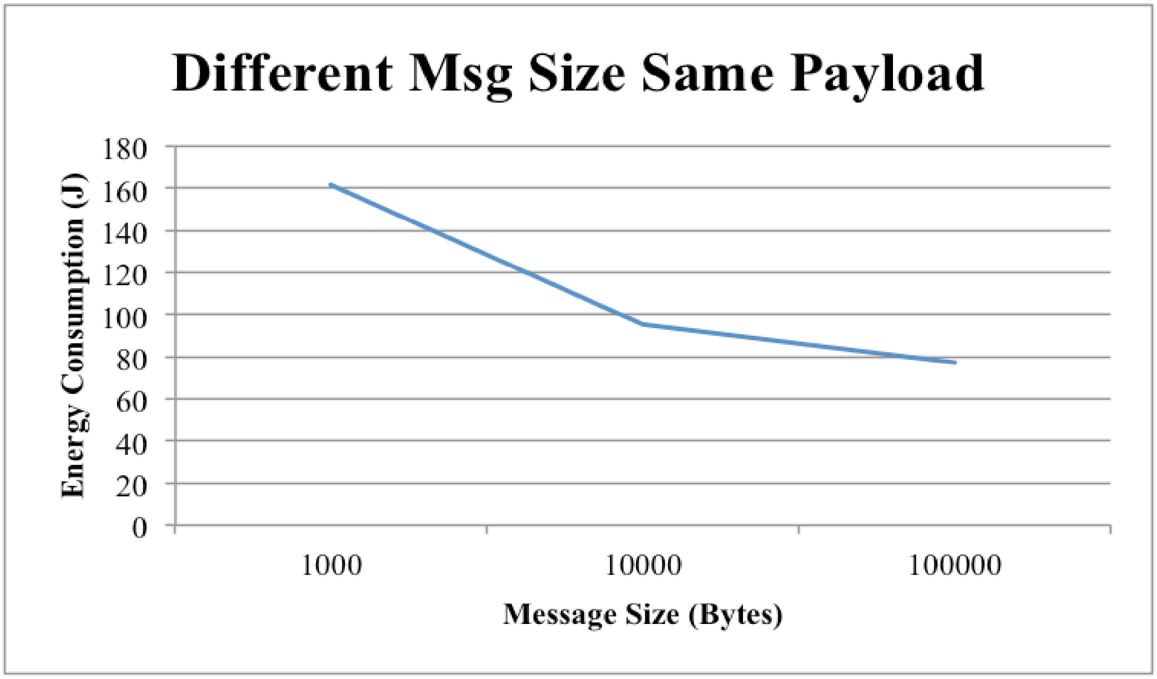

Figure 2-6: Payload Energy Consumption

Figure 2-5 shows the results of receiving and sending data messages from the sensor and mobile device. The results show that receiving data streams on the mobile device requires significantly less energy than sending data from the mobile device. Both figures show that in the case of sending and receiving data the energy consumption increases with message size, as one would expect. When receiving data messages energy consumption similarly increase with message size. The sending energy consumption shows a steady but more significant increase.

Figure 2-6 shows the results from an experiment sending 100,000,000 bytes of data using messages of different sizes. As the message size increases the number of messages that need to be send decreases in proportion, and although the power need to transmit larger messages increases the overall transmission time is smaller resulting in less energy consumption. This illustrates that collecting data samples in the DDDAS system can conserve energy and sending them as a larger message rather then immediately sending each sample as it is collected. The drawback of this approach is an increased delay to transmit each individual sample. The same experiment was implemented and energy measured on the receiving side of the mobile device. Similar results were obtained.

2.4 Power of Computation vs. Communication

DDDAS systems in mobile environments require energy for both communication of data and simulation computations. Figure 2-7 and Figure 2-8 show the average *power* consumption (energy per unit time) drawn from sending and receiving data continuously and the power drawn by both simulation models under different arrival rate inputs. Our experiments show that energy consumption for communication dominates energy consumption of the overall system for this traffic network configuration. Communication energy can be reduced by sending data with larger message payload sizes.

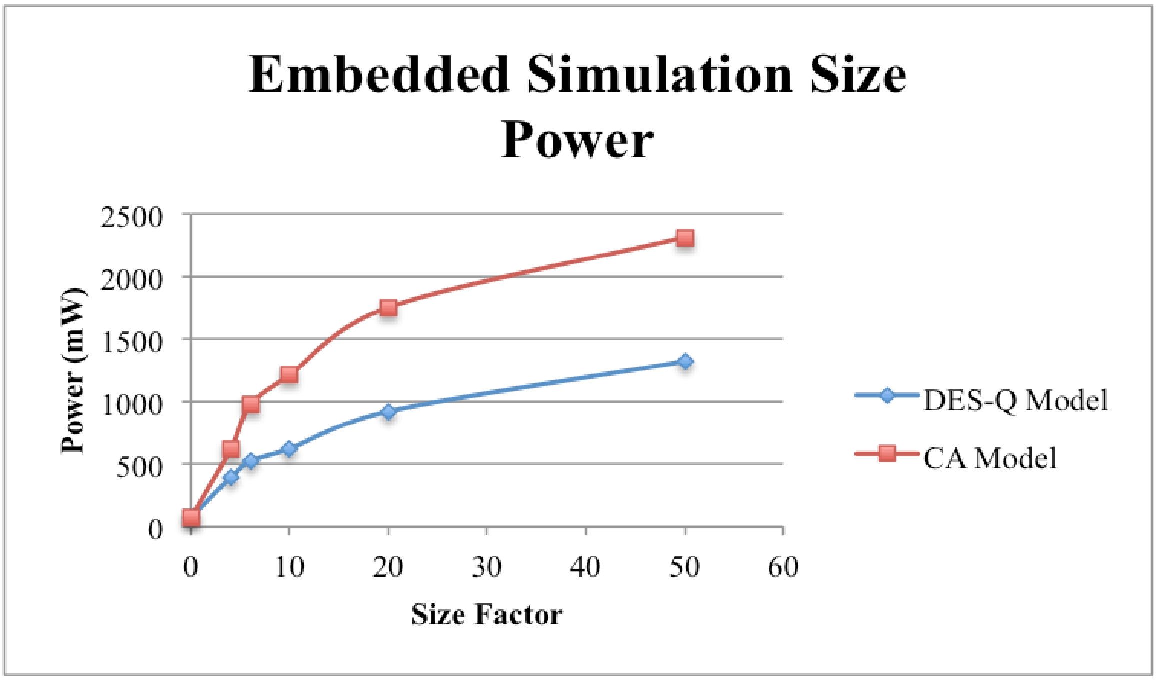

Figure 2-7: Power as Simulation Size Varies

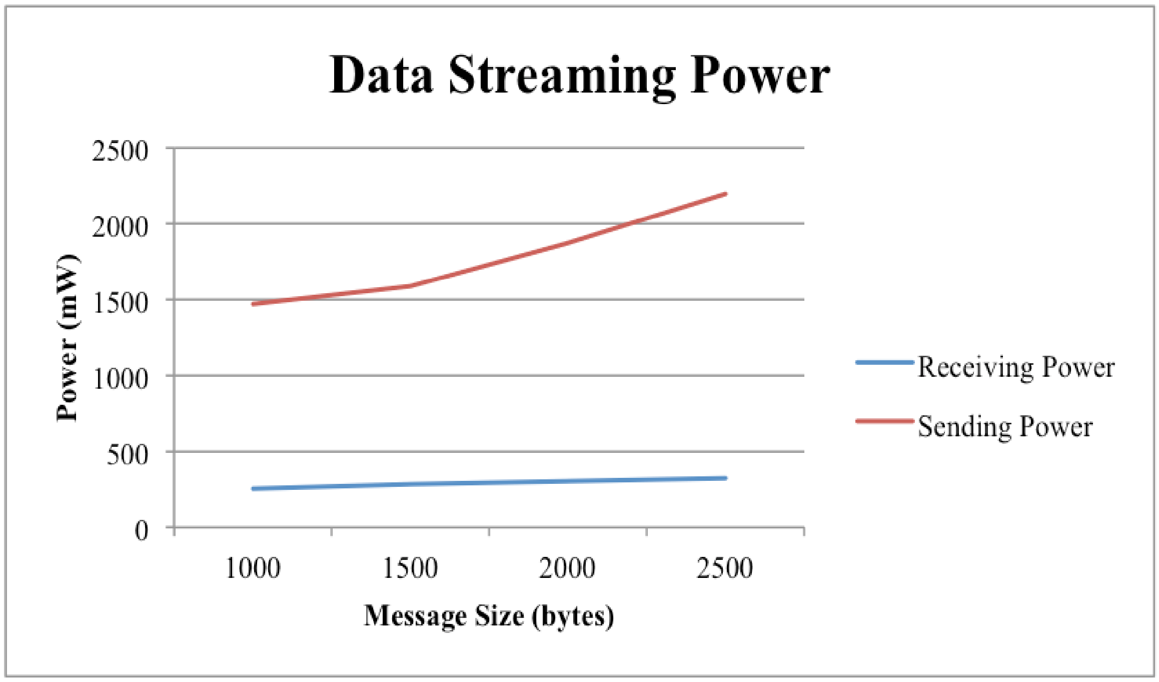

Figure 2-8: Data Streaming Power

2.5 Discussion

The results from our study give insights into the energy consumption of computation and communication in embedded traffic simulation systems. The increase in traffic volume entering the simulation system reflects an increase in energy consumption with the cellular automata model consuming overall more energy than the queuing model. This was a reflection of the nature of the simulation model. Cellular automata models update and check every vehicle within the model for each simulation time step update where as the queuing model updates the vehicle at the top of each queue representing the traffic network accounting to less vehicles being updated, this requires more memory access instructions to be executed to update each vehicle within the cellular automate model. Frequent memory access reflects increase in overall access time, which reflects an increase in energy consumption required to complete the task.

The experimental results in the communication study shows that continuous data streaming is not an effective technique to save energy for communication operations in energy constrained environments. Our experiments also illustrate that staggering communication with large data packets could lead to saving energy. These savings could be attributed to sending larger packets less often lead to large data overhead but since fewer messages are being sent the overall overhead need to complete communication is small compared to sending smaller packets more frequently.

2.6 Conclusion

The presented work describes an architecture for distributed simulation systems that are embedded in energy constrained mobile platforms. A DDDAS application system for predicting vehicle traffic flow was created and implemented in order to understand the energy consumption for different components of such a system. The major components were divided into computation and communication.

The computation components of such a system were defined to be the embedded traffic simulations that are responsible for making future state predictions of the traffic network. We compared the widely used cellular automata simulation model and a queuing network simulation. Our experiments show that when energy is the focus the cellular automata model consumes more energy. The CA model's energy consumption increases more rapidly as model size and traffic density increase.

The communication components of the system involved communication between the distributed simulation systems embedded on the mobile device and a sensor or sensors within the traffic network. Our experiments show that communication consumes a significant amount of energy. Sending messages from the mobile device holding the distributed simulation consumes far more energy than receiving messages. Greater energy efficiency is obtained by packing multiple data samples into a single message rather than sending multiple messages, but at the cost of increased delay to receive sampled data. Further, we observed that for these experiments the energy to send data greatly exceeds that required for the simulation computation and receiving messages, though this result depends on the size of the modeled network and the traffic load.

Our work illustrates the energy trends one might encounter under different communication and embedded simulation models used in a DDDAS system designed for predicting traffic network states that are driven by real time data streams.

3 POWER CONSUMPTION OF PRIORITY QUEUES FOR DISCRETE EVENT SIMULATIONS

The priority queue implementation holding the list of future events is central to discrete event simulations. Mobile computing now allows the possibility of executing online discrete event simulations that are driven by dynamic real time data as was illustrated in the previous chapter. The simulation system must be energy efficient in order to be effectively deployed in energy constrained environments such as mobile systems operating from battery power. Memory hierarchy and memory access plays a critical role on the overall energy consumption of priority queue implementations. This study investigates the effect that memory access has on power consumption for four widely used priority queue implementations: linked list, implicit heap, explicit heap, and splay tree.

The emergence of mobile and edge computing has created the possibility of running simulations on mobile devices, raising new problems and concerns. For example, dynamic data-driven application systems (DDDAS) feature a control loop involving sensing an operational system, utilizing analytics to optimize the system or to adapt the instrumentation system, and then deploying recommendations made by the analytics system [69]. On-line simulation clearly plays an important role in such systems. The energy and power consumption of simulations are major concerns in mobile DDDAS deployments because reduced energy consumption can lead to longer battery life, or the use of smaller, lighter weight batteries.

In cloud and high performance computing environments power consumption is similarly a major impediment to achieving increased levels of performance due to limitations in dissipating heat from electronic circuits. Power consumption has been cited as a key hurdle in achieving exascale performance in supercomputers. The U.S. Department of Energy has set a goal of 20 megawatts as the maximum power consumed by an exascale supercomputer [70]. The Thermal Design Power (TDP) is the amount of heat generated by a computer chip or component during normal operation for which the cooling system has been designed [71]. Power capping used in some systems places a maximum amount of power that can be consumed by a computer server. A typical goal for an HPC application might be to minimize execution time subject to staying within the specified power cap constraint. Further, power consumption in data centers used for cloud computing applications is a major operating expense; electricity is a major cost in operating data centers. It is estimated that in total, data centers consumed approximately 70 billion kW-hours or about 1.8% of the total electricity consumption in the U.S. in 2014 [72]. As such, reducing power consumption is increasing in importance for high performance and cloud computing applications.

In order to effectively run discrete event simulation systems on these types of systems an understanding of the energy consumed by all components of the simulation is needed. The priority queue data structure used for the future event list in discrete event simulation drives the overall execution of many simulation systems. Understanding the energy consuming components of this data structure is important for understanding the overall energy consumption of executing a discrete event simulation. This chapter examines and compares the energy consumption of four different implementations of the future event list in order to gain an understanding of where the energy is consumed and which implementation is effective for energy constrained environment when running a discrete event simulation.

Priority queues must implement enqueue and dequeue operations of events in discrete event simulations. An enqueue involves inserting a future event into the list. The dequeue operation is responsible for deleting the event from the list with the highest priority, or the smallest timestamp in a discrete event simulation. Discrete event simulations also sometimes require an operation to delete an arbitrary event to implement certain operations, however, this aspect is not considered here. The selected implementation determines how these operations are completed and plays a vital role in the overall execution of the discrete event simulation. Because many simulations contain only a modest amount of computation per event, the time to access the priority queue can have a large impact on the execution time of the simulation program.

The hold model is the standard method used to evaluate priority queue implementations. A hold operation involves a dequeue operation followed by an enqueue operation [73, 74]. The latter involves enqueueing a new event with timestamp T units into the simulated future where T is determined by drawing a random number from some probability distribution. Standard techniques can be used to measure the average time per hold operation performed on a queue of constant size. Although the hold model has certain deficiencies, e.g., the size of the priority queue remains fixed (plus/minus one), it provides a useful starting point for the present study.

3.1 Power Model for Priority Queue Implementations

Priority queues for the future event list drive the execution of a discrete event simulation and running such a simulation on an energy constrained device requires an understanding of where the most power is drawn. A power model is essential to characterize such power consumption. The main hardware components of interest here are the processor and memory system. Therefore, we developed a power model that includes the power consumed for computations and memory access of priority queue implementations. This relationship between computation power and memory access power are captured in the equations below.

PQPower = **computation_power + memory_access_power**

computation_power = computation_current* computation_voltage

memory_access_power = (cache_access_current*cache_access_voltage) + (extmem_access_current *extmem_access_voltage)

The elements of this power model include the power consumed for computation and for memory accesses. The computation power can be characterized by two major operations performed on the priority queue: comparison and data movement (e.g., swap) operations. Comparisons are made to find the correct location to place events within the priority queue. Depending on the implementation the complexity can range from O(N) to O(log N) for the data structures examined here. Data movement involves swapping positions between elements within the priority queue in order to maintain priority order. The power consumed to perform memory access includes the operations of accessing memory to swap events in the priority queue to maintain properties of the priority queue.

The overall power consumption depends on the frequency of occurrence of computations and memory accesses to implement these operations.

3.2 Priority Queue Implementations

The priority queue implementations evaluated in our study include the linear list, implicit heap, explicit heap, and splay tree. These are described next.

3.2.1 Linear List

The linear list implementation for the future event list keeps events stored in order using a linear singularly linked list data structure. The event with the highest priority is stored at the head of the list. Events are inserted or enqueued into the list using a linear search, searching by starting at the head of list first, in order to find the correct location to place the event. The enqueue operation on the linear list has a computational complexity of O(N) with N being the number of events. The dequeue operation consist of deleting the event at the head of the list with the highest priority and has complexity O(1).

3.2.2 Implicit Heap

The implicit heap implementation is a complete binary tree with nodes maintaining the heap property, i.e., every node’s timestamp is smaller than that of their children. The root node holds the smallest time stamped event overall. Implicit heaps can perform enqueue and dequeue operations with a complexity of O (log N) time. Implicit heaps use an array in order to store events. The children of node i are stored in positions 2i and 2i+1 in the array. Use of an array allows implicit heaps to have good spatial locality, which is advantageous in cache memory systems, especially when the data structure fits within the cache. By storing the data into an array the elements are able to be stored in contiguous memory locations. This property is important for energy consumption because assesses to main memory require a significant amount of energy to drive address and data lines and to access DRAM.

The enqueue operation for an implicit heap operates by inserting a value to the end of the heap and placing the node in the proper place within the heap by traversing up the heap comparing the priority of the node to its parent to make sure it does not violate the heap property. The dequeue operation for an implicit heap operates by deleting the head node of the priority queue, moving the last item in the heap to the root position, and traversing and swapping elements down the tree in order to maintain the heap property. If the number of elements in the heap exceeds the size of the array that has been allocated, a resize operation is required to either extend the size of the array, or if this is not possible, to allocate a new array and copy the existing heap elements to the new array. Similarly, if the size of the heap is much smaller than the size of the array, it may be desirable to resize the heap downward to allow the memory to be used for other purposes. Heap resizing can be a time consuming operation if the heap contains many elements.

3.2.3 Explicit Heap

Like the implicit heap, the explicit heap uses a complete binary tree with nodes maintaining the heap property. The explicit heap also performs enqueue and dequeque operations with a worst case complexity of O (log N). Explicit heaps use pointers to store heap data elements rather than through the use of array indices. The enqueue and dequeue operate in the same manner as the implicit heap except an additional operation is required to locate the last element in the heap. In the implementation used here memory for heap nodes are allocated by separate dynamic memory allocation operations, e.g., using malloc() in C. The explicit heap therefore eliminates the need for resizing operations. However, the explicit heap loses some of the spatial locality properties realized through the array in the implicit heap because memory obtained via malloc() may not be contiguous with memory previously allocated for other nodes in the heap. Further, use of explicit pointers increases the amount of memory required for each heap node, increasing the memory footprint of heap nodes and reducing the efficiency of the memory hierarchy. With regard to energy this can lead to less effective use of cache memory in the explicit heap implementation, possibly leading to greater energy consumption. In addition, the increased time for memory operations will lead to greater static energy consumption.

3.2.4 Splay Tree

The splay tree is a self-balancing binary search tree. Using explicit pointers for links, it has O(log N) average amortized time complexity of enqueue and dequeue operations [75]. The splay tree uses a heuristic called a splay operation to help rebalance the tree after insertion and deletion operations are performed and bring the last accessed key to the root of the tree. Splay tree enqueue operations consider four cases. If the tree is empty a node is allocated and stored in the root of the tree. The second case uses the splay function which is the operation of moving a node to the root of the tree. The splay tree performs standard binary search tree search operation in order to find a given item (with a specified key) within the tree. If the key for the given value being entered into the tree is already there then that key is splayed and becomes the new root of the tree, else if the key isn't present then the last accessed node is splayed and becomes the root of the tree. Otherwise, if the key of the node being entered is the same as the key of the root node then nothing has to be done since the key is already present. If third case is not true then we allocate memory for a new node and compare the root's key with the given key to be inserted. If the given key is smaller than the root's key then the root node becomes the right child of the new node, and the left child of the root node becomes the left node of the new root node. Otherwise if the given key is greater than the root the root becomes the left child of the new node.

The dequeue operation performs in a similar matter. It has four cases as well. If the root is null it simply returns the root. The second case checks if the given key is in the tree and makes it the new root. If not then the last accessed leaf node now becomes the new root. If the given root is not present then delete the root of the tree. The last case is if the given key is present then split the tree into two trees, one for the left sub tree and the other for the right sub tree. If the root of the left sub tree is null then return the root of the right sub tree. Else splay the node with the maximum value of the left sub tree, then make the root of the right sub tree the right child of the root of the left sub tree and return the root of the left sub tree.

3.3 Power and Energy Experiments

To evaluate the power and energy consumption of the priority queue implementations we utilized the Jetson TK1 development boards with ARM 32-bit CPU with 4 cores operating at 2.3 GHz and 2 GB memory running Linux 3.10.40 operating system. The L1 cache energy and power measurements were completed using the PowerMon2 power measurement system. All implementations were created using the C programing language and compiled and ran using the gcc 4.8.4 compiler. For all evaluations we ran our implementations with one cpu core powered on and the other three cores powered down and we also performed the experiments with only the low power core on.

To evaluate the performance of the priority queue implementations we used the hold model. The simulation engine drives discrete event simulations, which hold the future event list. The future event list is responsible for holding the events that represent the system. Each event is responsible for holding a sequence of operations that reflect a process that occurs in the real system. Each event is executed once it is retrieved from the future event list, an execution of an event can lead to a creation of a new event that is then placed in the future event list. Executing and inserting events drive the execution of the discrete event simulation until all events or processed or the simulation time has completed, which ever comes first. The hold model is a widely used to measure priority queue performance [76]. The priority queue is initialized to contain N events. Sequences of hold operations are then performed where each operation involves removing the highest priority (smallest timestamp) event, and inserting a new event known as a dequeue operation. The time stamp of the new event is obtained by adding the current simulation time, i.e., the timestamp of the last event that was removed, with a random value drawn from a fixed exponential probability distribution, known as an enqueue operation. In these experiments the size of the priority queue was varied while keeping the number of hold operations constant.

3.4 Cache Hit Benchmark

To determine the power consumption for a cache hit we implemented a simple two-dimensional array and performed memory access operations. An integer array of size 100 x 100 was implemented in C and pre filled with zeros. In order to determine the power consumed when cache hits dominate we accessed the first four hundred and fifty values in contiguous memory one thousand times and computed the average power drawn by one cache hit. We repeated the same experiment accessing locations within the array that are contiguous we instead accessed locations within the array that were not stored in the L1 cache. To determine the power consumption for a cache miss we implemented a two dimensional array that could not completely fit in the cache. An integer array of size 500 x 500 was implemented in C and filled with zeros. We staggered the locations that were accessed by accessing locations at the end of the array to ensure that we obtained a cache hit and did this for four hundred fifty locations within the array one thousand times. This allowed us to obtain the average power drawn for a cache miss. These benchmark values give us a baseline to use for constants for our power model to determine the power consumption for future event list implementations.

Table 3-1:Memory Benchmark Value

Cache hit power	3.893331 watts
Cache miss power	4.348888 watts

3.5 Priority Queue Power Consumption

An evaluation for small priority queues that fit in the L1 cache was first performed using the hold model to provide baseline data. This configuration was designed to keep the number of cache misses and references to main memory to a minimal level. Figure 3-1 shows the average power drawn in performing ten million hold operations for the four priority queue implementations using different queue sizes. The same evaluation was also performed for large priority queues that do not fit in the cache in order to assess the impact of cache misses on power which is also display in the same graph to the right of the vertical black line. The black line indicates the point at which the queue no longer fits within the L1 cache. Figures 3-2 shows the average energy drawn for the same experiment amongst the four implementations. Figures 3-4 show the execution time for completing one hold operation for the four implementations evaluated when performing this experiment.

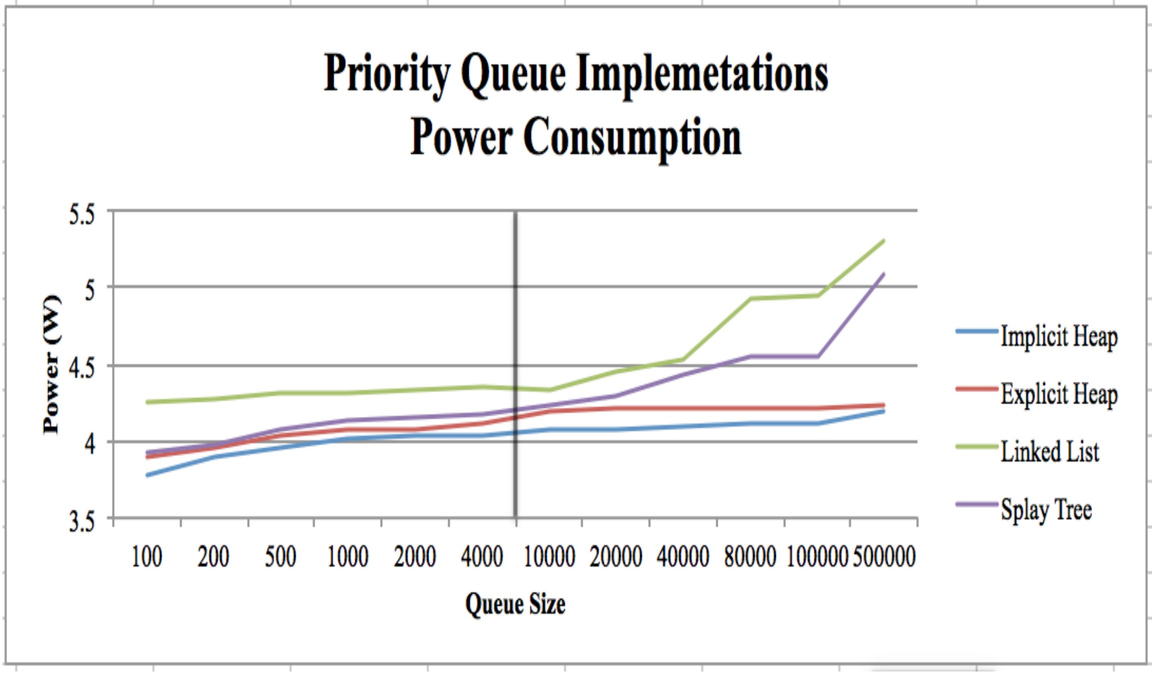

Figure 3-1: Priority Queue Implementations Power

The graphs indicate that the implicit heap yields the least power drawn, for both large and small queues. Figure 3-1 indicates that as the size of the queue increases but stays within the cache that the power consumed by the implicit heap and linear list remains about the same, but increases significantly for the implicit heap and splay tree. The linked list draws the most power at a queue size of 1000 but consumes less power than the explicit heap and splay tree when the queue size reaches 4000. For large queues that do not fit in the cache, shown to the right of the black vertical line, the implicit heap again consumes the least amount of power, and this amount does not vary significantly as the queue size is increased. Power in the other implementations does increase with queue size. We attribute the power draw increases for the explicit and splay tree to an increase in the rate of the number of swaps performed per second in order to maintain the data structure. We analyze this in section 3.6.

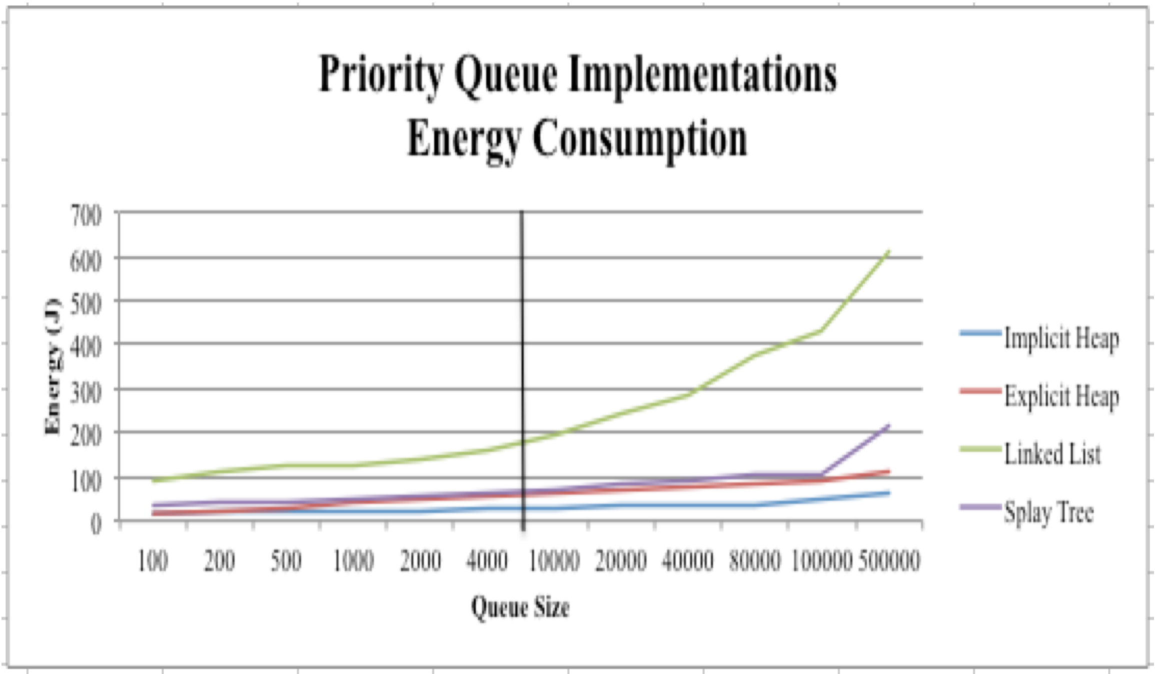

Figure 3-2: Priority Queue Implementations Energy Consumption

While the power measurements indicate the rate of energy consumption, the next set of curves show the overall energy consumption, which is affected by the time required to perform enqueue and dequeue operations. The energy consumption for small and large queue sizes shown in figure 3-2, the figure indicates similar trends in energy consumption for small and large queues. The linked list consumes the most energy due to the time required to perform hold operations; this is shown in figures 3-3. The implicit heap consumes the least amount of energy. This result is a direct consequence of the small amount of power drawn and time needed to execute hold operations in this implementation. From these results one would conclude that where power, energy, and time are a concern the implicit heap is the best implementation among those examined. The explicit heap is second; this is also a consequence of the amount of time needed to execute hold operations.

3.6 Priority Queue Swaps, Linear Comparisons, and Pointer Swaps

The average number of swaps and comparisons per second were compared for the four implementations. The implicit heap, explicit heap, and splay tree swap elements in order to maintain the heap property or to rebalance of the tree. The linked list implementation performs comparisons amongst the elements inserted into the list but does not require swaps. These experiments performed ten million hold operations. The size of the priority queue was varied from one thousand to four thousand elements for small queues that fit in the cache and large queues varying in size from ten thousand to forty thousand elements.

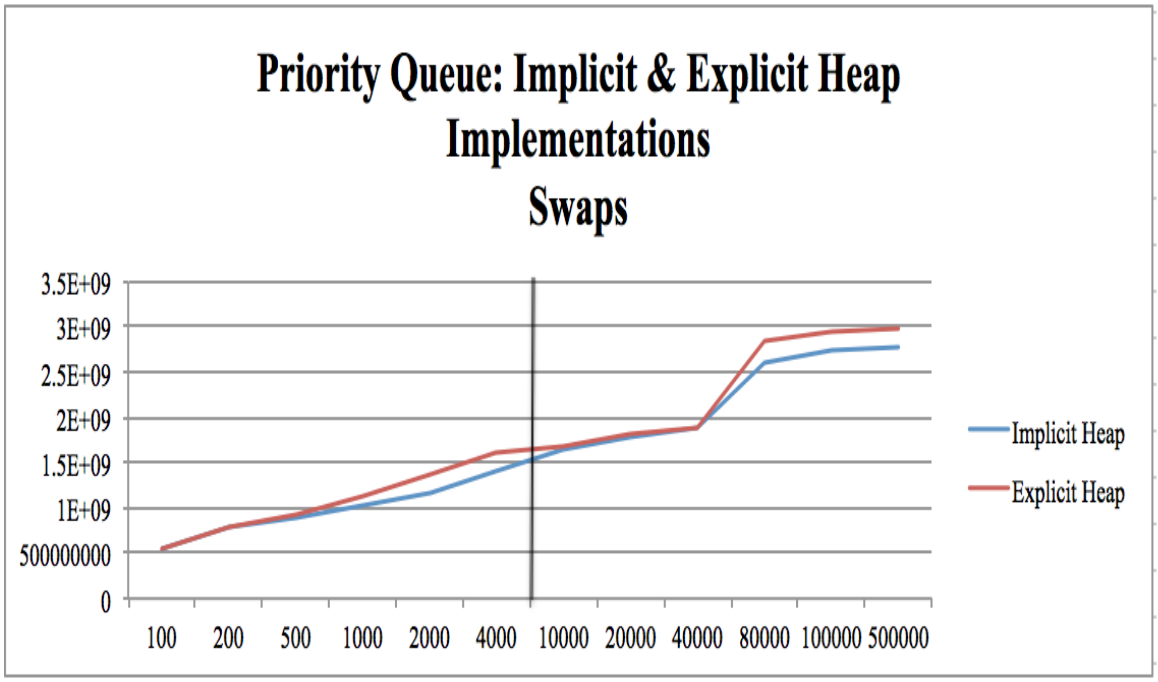

Figure 3-3:Priority Queue Implicit & Explicit Heap Swaps

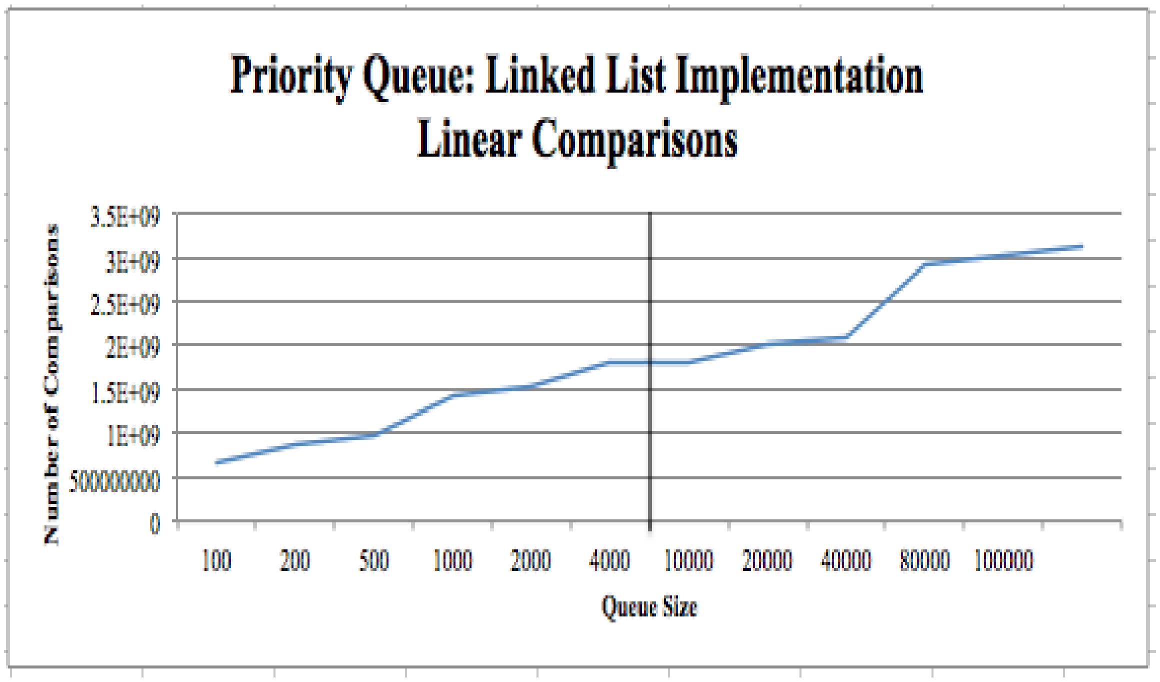

Figure 3-4:Priority Linked List Implementation Linear Comparisons

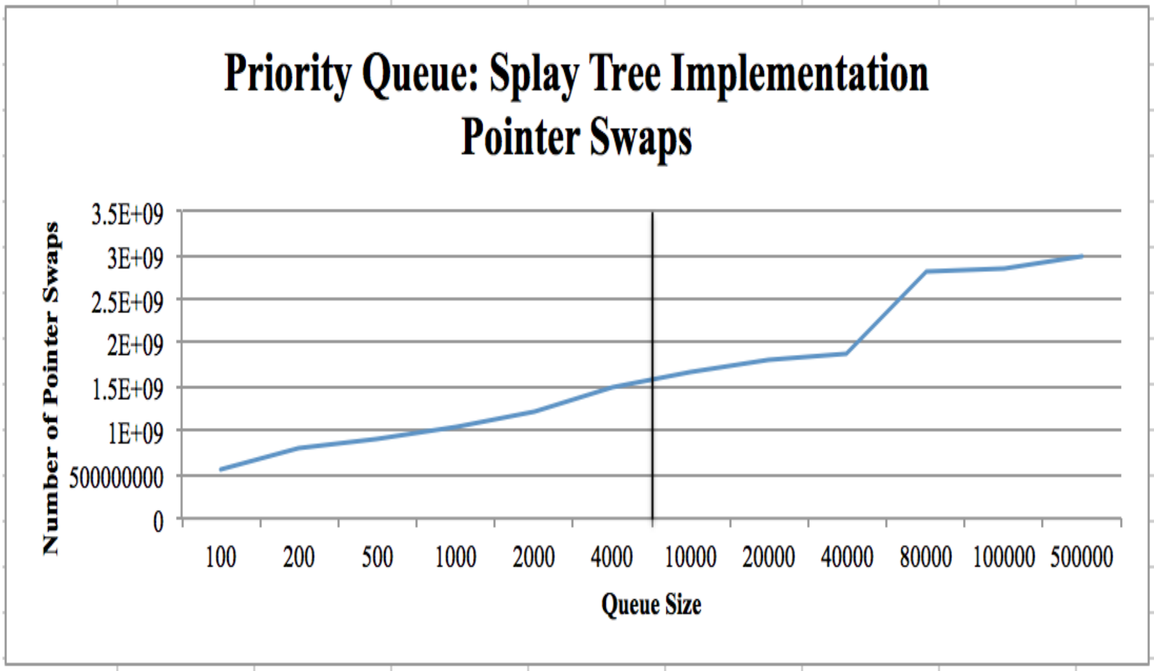

Figure 3-5: Priority Queue Splay Tree Pointer Swaps

The results indicated in figure 3-4 show that the linked list implementation produces the highest rate of comparisons per second for small queues and the smallest rate for large queues. This is consistent with the nature of the data structures. For the tree based implementations, as the size of the tree grows more swaps are required for each priority queue operation in order to not only place events in the correct location but also to maintain the structure of the tree. The linked list structure is not changed when events are inserted.

The average time per hold operation in executing the priority queue implementations using the hold model for evaluation is shown in the graphs below. We compare the average time per hold operation for both small (Figure 3-6) and large queues sizes indicated to the right of vertical black line.

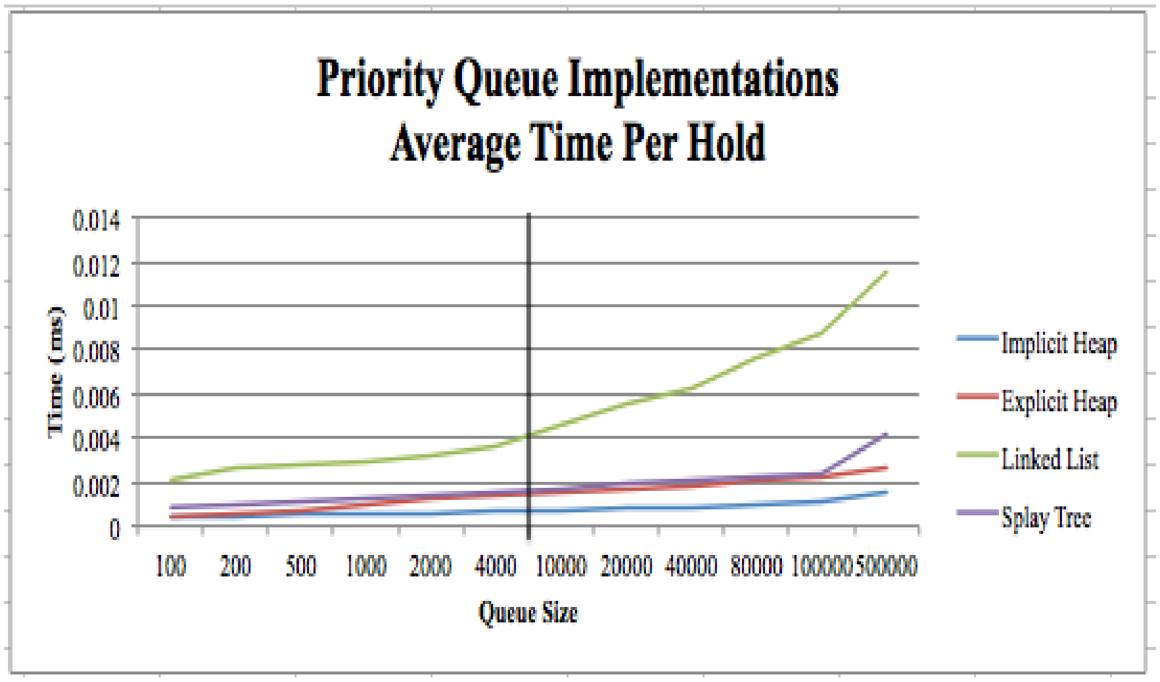

Figure 3-6:Priority Queue Implementations Average Time Per Hold

Figure 3-6 shows that for both large and small queues the average time per hold operation increases with the size of the queue. We also see that the pattern is the same amongst the four implementations for both the small and large queues, the implicit heap performs its hold operations with the quickest time with the linked list implementation performing hold operations with the longest time per hold. The linked list time per hold is attributed to fact that the enqueue operation performs a linear search in order to insert events into the queue which requires and O(N) complexity whereas in the other three implementations enqueue operations have O(log N) complexity. All implementations have their time per hold rate increase with the size of the queue regardless if the queue fits within the cache as shown in Figure 3-6 or does not fit within the cache as shown to the right of the vertical black line in figure 3-6.

3.7 Priority Queue Energy Model for Hold Operation

The preceding results show the overall energy and power consumption of four different priority queue implementations: implicit heap, explicit heap, linked list, and splay tree data structures. We examine how the size of the priority queue affects the overall energy, power, time per hold, and number swaps and comparisons as we increase the size of the priority queue. Examining the data from these experiments provides insight toward understanding and creating a model for each implementation that can predict the amount of energy expended for performing one hold operation. We evaluate how each prediction model changes as the size of the priority queue fits within the cache (32 KB), and exceeds the size of the cache of the development board.

3.7.1 Implicit Heap

The implicit heap is an array implementation of the priority queue. To enqueue into the implicit heap requires that a node be inserted at the end of the heap. Since the implicit heap is an array implementation we can access the end of the heap in O(1) time. Once a node is inserted into the heap we must ensure that heap order is still intact. To ensure heap order a heapify operation is performed. We heapify the newly inserted node up the heap by performing swaps between nodes in the heap until the order is restored. To dequeue from the heap we swap the last element in the heap with root. We then delete the last node from the heap, which is now the root. We then perform the heapify operation starting at the root heapfying down the heap to restore heap order by swapping nodes. The heapify operation performs in O(log n) time with n being the number of elements in the heap. Our energy model below shows the components of the implicit heap data structure that affects the energy needed to perform one hold operation on an implicit heap.

$$\text{Energy} = C_1 + C_2 S$$
$$C_2 = \text{time\_swap} * \text{power\_swap}$$
$$S = \text{number of swaps}$$

Table 3-2: Implicit Heap Energy Constants

	C_1	C_2
In Cache	4.00E-07J	1.44E-08J
Out Cache	5.00E-07J	1.60E-08J

C_1 reflects the constant base energy needed for the implicit heap on the development board. C_2 represents the energy needed to perform one swap operation on an implicit heap. This constant is based on the time and power needed to perform a swap operation. We evaluate the energy per hold as we increase the number of swaps. Our model compared to the experimental results is shown in the graphs below for implicit heaps whose size fit within the cache and those whose size cannot fit completely within the cache. Table 3-2 shows the constant values used for our prediction model for implicit heaps within and not within the cache.

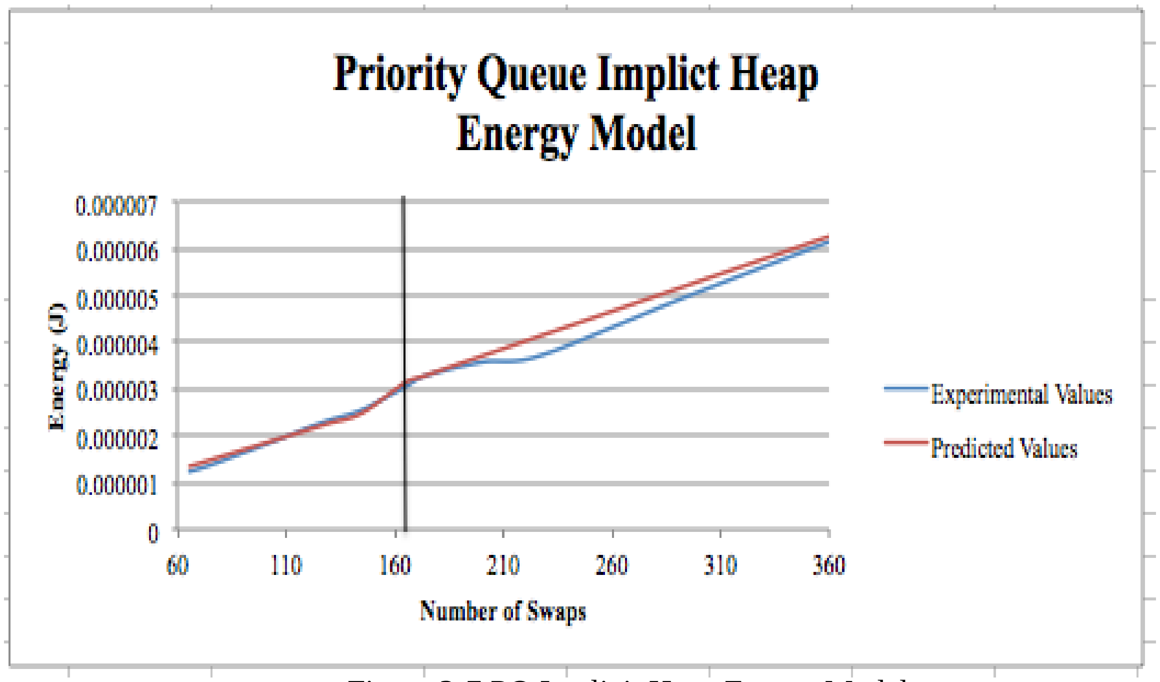

Figure 3-7:PQ Implicit Heap Energy Model

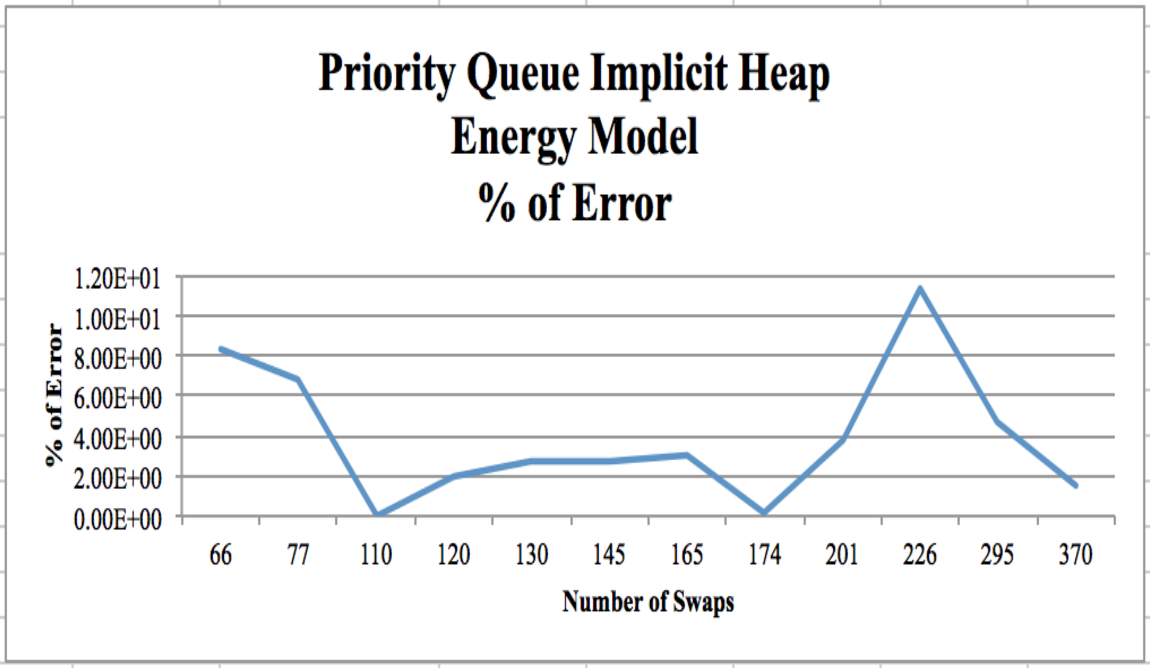

Figure 3-8: PQ Implicit Heap Energy Model % of Error

Figure 3-7 displays our experimental results in blue and our predicted results in red. The data suggests that the relationship between energy consumption and performing one hold operation is linearly dependent on the number of swaps that are need to complete a dequeue followed by an enqueue when using an implicit heap. Figure 3-7 also shows that more swaps are needed when using a larger heap and the effects of accessing memory that is not located in the cache. Figure 3-7 compares the measurements with predictions from the model when we use heaps who size fit within the cache cannot completely fit within the cache indicated by the data to right of the vertical black line. The constant values using implicit heaps with sizes larger than the cache and accessing locations within the heap that may not be located in the graph are shown in column 2 of table 3-2. It is seen that there is good agreement between the values predicted by the model and those observed in the measurements.

3.7.2 Explicit Heap

The explicit heap is a pointer implementation of the priority queue. It performs the enqueue and dequeue operations in the exact same manner as the implicit heap. The difference between the implicit and explicit heap is that since it is implemented using pointers we can not guarantee that memory allocated for nodes within the heap are contiguous in memory. We also do not have a O(1) access to the last node within the heap; rather, O(log n) time to find the last element within the heap. This requires more energy to perform a hold operation when using an explicit heap. Our energy model showing the energy needed to perform one dequeue followed by an equeue are displayed below.

$$\text{Energy} = C_1 + C_2S + C_3D$$
$$C_2 \text{ (swap energy)} = \text{time\_swap} * \text{power\_swap}$$
$$C_3 = \text{energy to search}$$
$$D \text{ (depth)} = \log \text{(Num elements in heap}$$

Table 3-3: Explicit Heap Energy Constants

	C_1	C_2	C_3
In Cache	5.50E-07J	2.00E-08J	3.00E-07J
Out Cache	6.60E-07J	2.50E-08J	3.50E-07J

The prediction model for the explicit heap includes the energy consuming components of the hold operation. C_1 represents the base energy needed to store an explicit heap on the development board. C_2 represents the energy needed to swap between elements within the heap when heapify operations are performed. C_3 represents the energy needed to search for the last element in the heap when performing a dequeue operation; this value is dependent on the depth of the heap which depends on the size of the heap. Table 3-3 shows the constant values used in our prediction model for values C_1, C_2, and C_3 for heaps that fit within the cache and heaps that cannot completely fit within the cache.

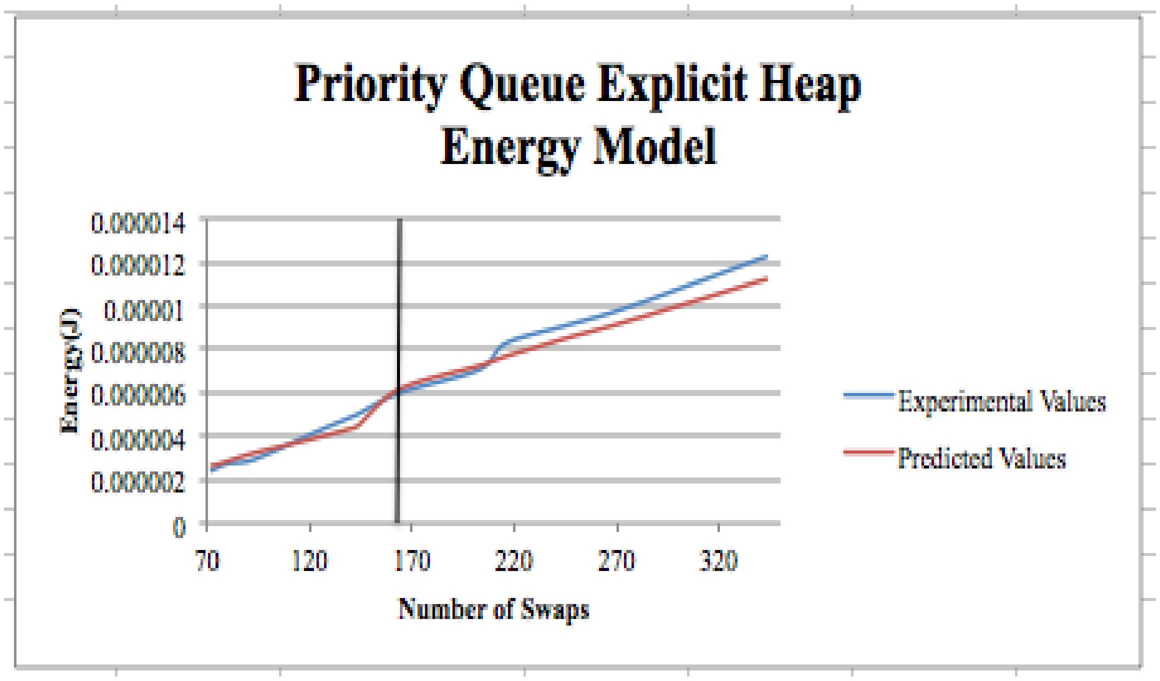

Figure 3-9: PQ Explicit Heap Energy Model

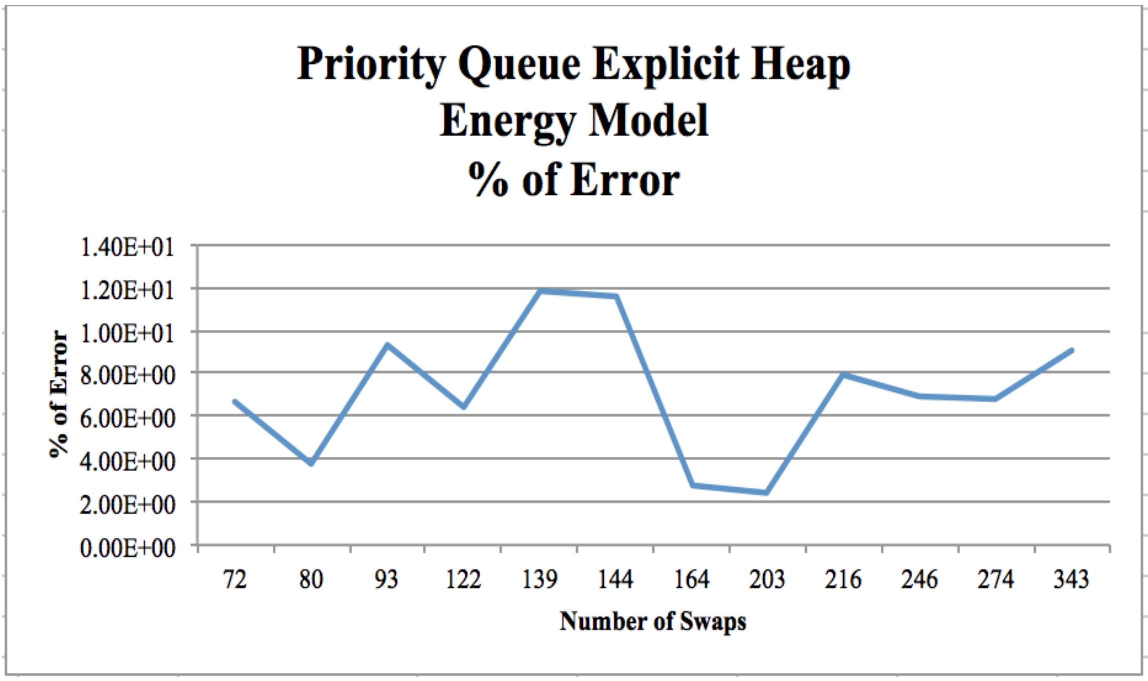

Figure 3-10: PQ Explicit Heap Energy Model % of Error

Figure 3-9 compares the results from the prediction model to experimental measurements as we increase the number of swaps. Like the implicit heap we see that as more swaps are needed to complete a hold operation data accessed through pointer references may not reside within the cache memory. This access operation may require the need for more energy depending on the memory access pattern. Figure 3-9 shows the prediction model that takes into account these out of cache memory accesses that may be occurring once we perform hold operations on heaps who size cannot completely fit within the cache. More swap operations are required and more energy is consumed to reach the last element within the heap.

3.7.3 Linked List

The linked list implementation is also a pointer implementation of the priority queue where the underlying structure is a linear list of nodes that are in order based on priority. From a practical perspective this implementation is only useful for priority queues of small sizes. To perform an enqueue operation on a linked list requires that we compare the inserted value to the values already in the list to find the appropriate location to store the inserted value. The enqueue operation in a linked list takes O(n) time, with n being the number of elements in the list. The dequeue operation on a linked list priority queue implementation requires that we remove the root node from the list and make the next node within the list the new root. This operation requires O(1) time. Our focus for the prediction model for the linear list implementation of a priority queue is on the amount of energy needed to perform a linear comparison between nodes. Since a link list is a pointer implementation we are typically dealing with non-contiguous memory locations in order to access the pointers within the list. This property can affect the search energy. The prediction model for the linked list implementation is as follows:

$$\text{Energy} = C_1 + C_2\text{Comps}$$
$$C_2 = \text{time\_compare} * \text{power\_compare}$$
$$\text{Comps} = \text{number of comparisons}$$

The prediction model uses C_1 to represent the base energy needed to maintain a linked list data structure. The energy to compare nodes within the link list is represented by the value C_2 shown in table 3-4. The value of C_2 is determined by the product of the time required to compare elements with the power required to make a comparison between two nodes.

Table 3-4: Linked List Energy Constants

	C_1	C_2
In Cache	1.00E-08J	9.00E-08J
Out Cache	1.70E-08J	9.30E-08J

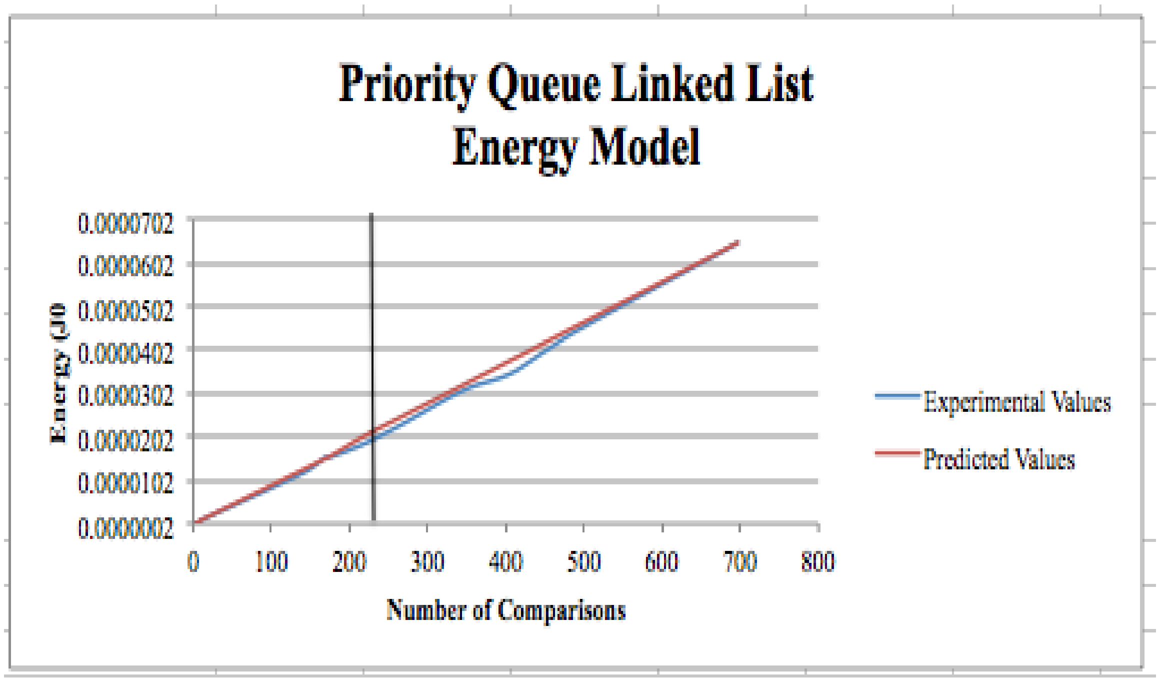

Figure 3-11: Priority Queue Linked List Energy Model

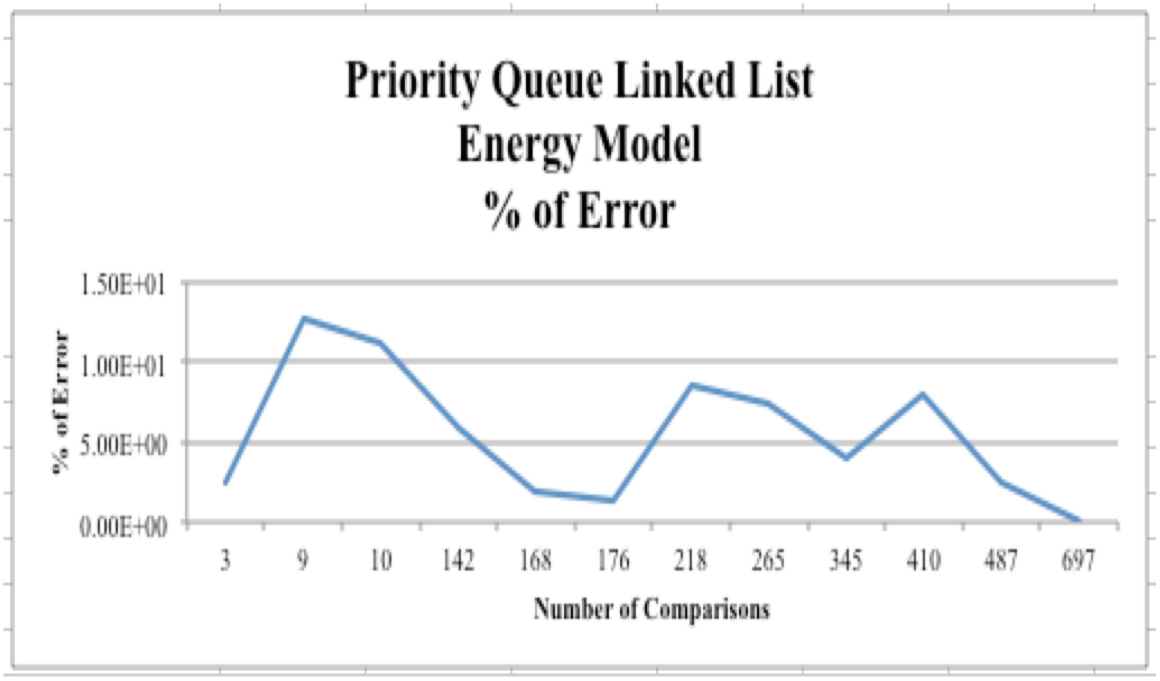

Figure 3-12: PQ Linked List In Cache Memory % of Error

The linked list model shows a linear relationship between the size of the list and the numbers of comparisons needed to perform a hold operation on the linked list implementation. These results demonstrate that as the number of comparisons needed to complete a hold operation increases that the overall energy consumption increases linearly. This trend is seen in Figure 3-12 when the linked list fits within the cache and when it does not, respectively shown to right of the vertical black line in the figure. Our in-cache model when compared to the experimental values show that as we get closer to linked list sizes near the size of the L1 cache that our prediction model doesn't begin to hold as well. This maybe a reflection that we are now accessing values outside of the cache requiring in a need for more energy to be exerted to perform enqueue and dequeue operations. This requires an adjustment to the constant values for prediction energy consumption for linked list whose total size cannot fit within the cache.

3.7.4 Splay Tree

The splay tree implementation, like the implicit and explicit heap, is another variation of the binary tree data structure. The splay tree uses splay operations to keep commonly accessed nodes near the root to take advantage of locality. Like the explicit heap the splay tree implementation uses pointers to connect nodes within the data structure. Splay trees maintain balance by using the splay operation to perform rotations on the tree to bring needed nodes to the root of the tree. An enqueue operation requires O(log n) amortized time complexity, but has a worst case time of O(n) for individual operations. Predicting the amount of energy needed to perform a hold operation, i.e., a dequeue followed by an enqueue operation, requires determining the energy needed to perform a splay operation. The enqueue operation requires a search to find the next empty location within the tree. Once the inserted element is placed in the tree, a splay operation is performed to (approximately) restore balance to the tree. The splay operation requires that rotations be performed and within those rotations nodes swap pointers to their right and left children in each sub tree. Our prediction model for one hold operation reflects the relationship between these three operations and is shown in the equation below.

$$\text{Energy} = C_1 + C_2R + C_3D$$
$$C_2\ (\text{rotation energy}) = \text{time\_swap} * \text{power\_swap}$$
$$C_3 = \text{rotation energy}$$
$$D\ (\text{depth}) = \log(\text{num elements})$$
$$R = \text{number of rotations}$$

Table 3-5: Splay Tree Energy Constants

	C_1	C_2	C_3
In Cache	6.00E-07J	1.70E-07J	1.64E-07J
Out Cache	6.50E-08J	2.00E-07J	1.67E-07J

This model associates the base energy needed for a splay tree with the number of swaps required and the number of rotations needed to complete an enqueue and dequeue operation using the splay tree data structure. C_1 represents the base energy, C_2 represents the energy need to complete a rotation within the tree during a splay operation, and C_3 represents the energy needed to perform a search for the last element within the tree during an enqueue operation. D represents the depth of the tree.

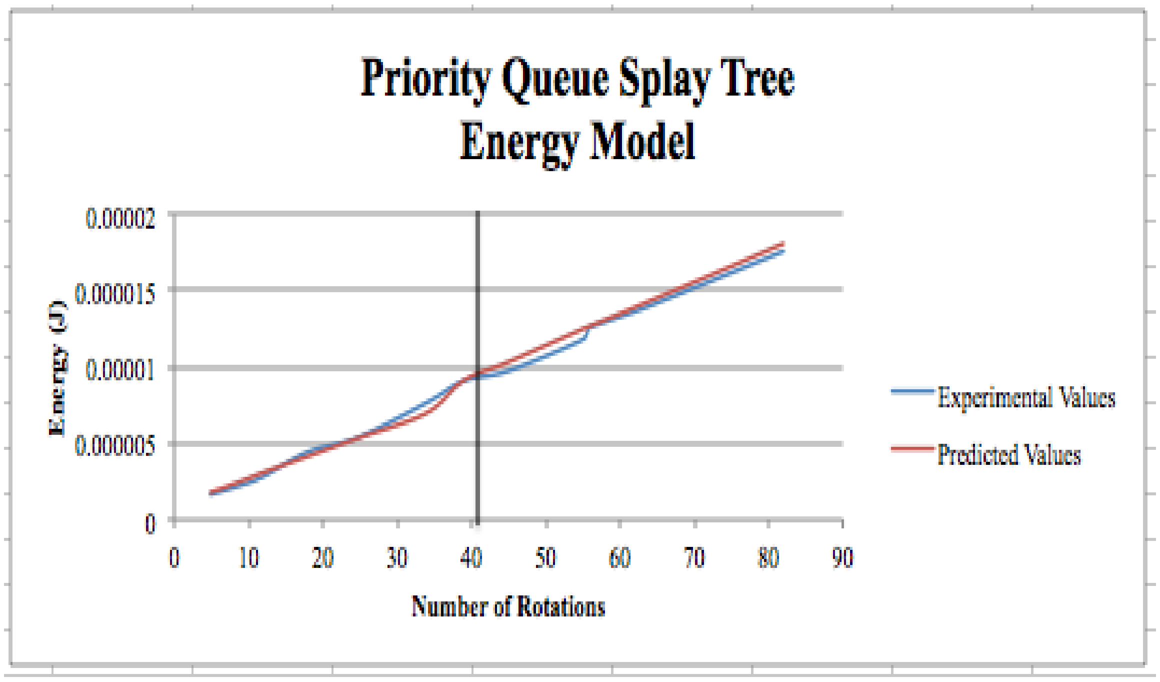

Figure 3-13: PQ Splay Tree In Cache Memory

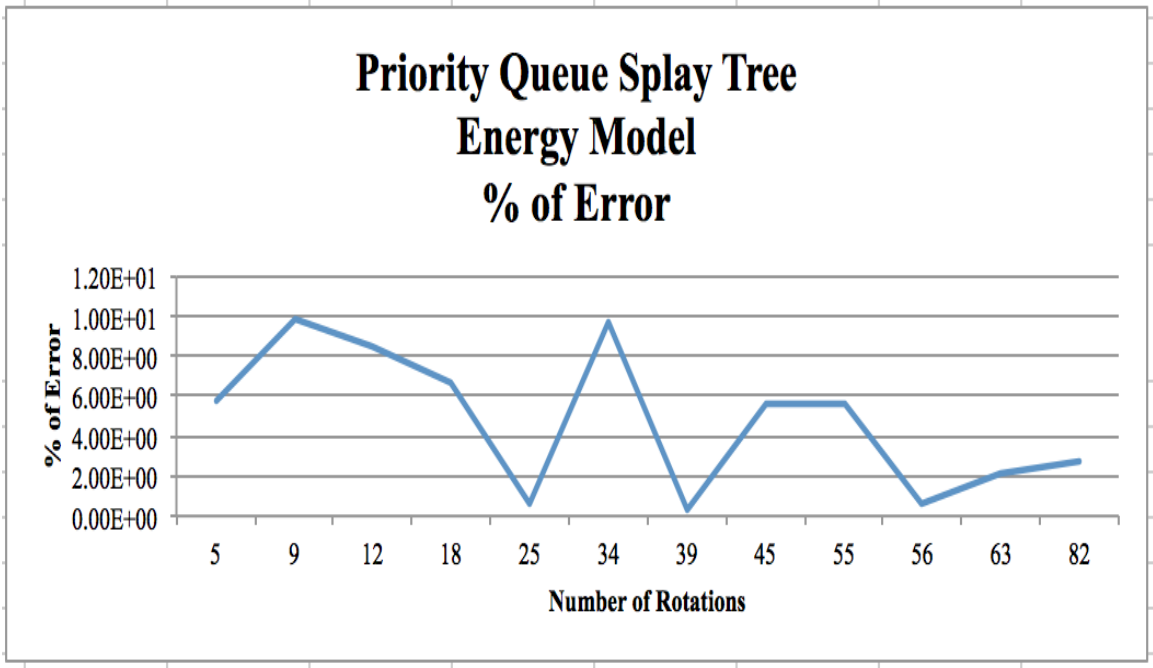

Figure 3-14: PQ Splay Tree In Cache Memory % of Error

Figure 3-13 shows that as we increase the size of the tree to no longer fit within the cache more energy is needed to search for the last node and rotate the nodes within the tree to maintain balance.

3.8 Discussion

Our study gives an overview of how the implicit heap, explicit heap, linked list, and splay tree priority queue implementations perform under the hold operation. As the size of the priority queues increase we see a change in the behavior of the energy consumption and attribute that to memory access. The implicit heap performs with the least amount of energy consumption amongst the implementations evaluated. We attribute this behavior to the nature of the underlying data structure of the implicit heap, an array. Since arrays take advantage of using local locality we can guarantee that for implicit heaps that fit within the cache that they are contiguous in memory, which is advantageous for energy consumption. To further evaluate our assumption that memory access in the cache versus outside of the cache affects overall energy consumption under the hold operation we dissect each data structures operations performed during the hold operation to illustrate the energy consumption behavior. Our investigation attributes the relationship of the implicit heap to the number of swaps required to perform one hold operation for queues who's size can fit within the cache and sizes which cannot completely fit within the cache. Our prediction model shows for all priority queue implementations that as we increase the number of swaps needed and the size of the priority queue we get closer to maximum L1 cache size of the board and our prediction percent of error begins to widen. We then reformulate the constants for our prediction model in order to gain a closer predication for priority queue sizes that cannot completely fit within the cache. This is done for all data structures. Our explicit heap implementation is also dependent upon the number of swaps that are needed to complete a hold operation where as the linked list is dependent on the number of comparisons between nodes within the list. The splay tree implementation is dependent upon the rotations needed to be performed during the splay operation that occur when a hold operation is executed. This gives us further validation that accessing memory outside of the cache requires more energy than accessing values within the cache despite what underlying data structure is used amongst the data structures of our investigation. The energy constants required to perform the operations (implicit swap, explicit swap, linear comparison, and tree rotation) that drive the energy consumption for the four implementations are displayed in Table 3-6 below.

Table 3-6: Swaps, Linear Comparisons, Tree Rotation Energy Constants

	Implicit Swap Energy	Explicit Swap Energy	Linear Comparison Energy	Tree Rotation Energy
In Cache	1.44E-08J	2.00E-08J	9.00E-08J	1.70E-07J
Out Cache	1.60E-08J	2.50E-08J	9.30E-08J	2.000E-07J

When comparing the energy driving operations between each implementation the constants validate that the implicit heap is the most energy efficient implementation. Overall energy efficiency is gained by the amount of energy that is needed to perform an implicit swap in an implicit heap. This behavior is a reflection of the amount of time that it takes to perform an implicit swap, the time her is a consequence of the implicit heap taking advantage of spatial locality that is gained when the underlying data structure is an array. The explicit heap performs the same swap operation between elements but does not use spatial locality, under this implementation pointers are used and do not guarantee spatial locality, finding a pointer in memory to retrieve an element requires more time than an implicit heap implementation leading to more energy consumption which is reflected by the constant values displayed in Table 3-6. The constant energy value to perform a linear comparison also is a reflection of the amount of time it takes to search through the linked list to determine where an inserted value fits within the list. This constant value could be different depending on the distribution used in order to determine the priority of the future event inserted in the list. The linked list implementation could be useful for simulation applications that require very little events required to be store in the future event list. The user may gain overall energy consumption when the list has less than fifty items in the tree. The energy required to perform a hold operation in a linked list requires only 1.00E-08J for queues that fit within the cache and 1.70E-08J for queues that do not completely fit within the cache. The implicit heap, explicit heap, and splay tree implementations require 4.00E-07J or greater to perform a hold operation on their data structures. They are more energy efficient for larger applications because they save more overall energy consumption when performing swaps in the heap implementations and rotation on the splay tree implementation. The table also reflects that when comparing all four operations that performing a rotation in a splay tree requires the most energy consumption. This raises the question of why is a splay tree implementation faster than a linked list implementation. The splay is a hierarchal implementation who performs its insert and delete in O(log n), so although performing a rotation takes longer than a linear comparison the number of rotations that need to be performed to complete an insertion into a splay versus inserting into a linked list requires less overall time leading to significantly less energy consumption overall. Another behavior that is apparent when comparing energy constants of the data structures evaluated is the energy required to search for the last element in an explicit heap in comparison to a splay tree. These constant values are reflected in table 3-7 below.

Table 3-7: Explicit Heap and Splay Tree Search Energy Constants

	Explicit Heap Search Energy	Splay Tree Search Energy
In Cache	3.00E-07J	1.64E-07J

Out Cache	3.50E-07J	1.67E-07J

Table 3-7 shows that searching for the last element within a splay tree requires less energy than in an explicit heap. The splay tree uses rotations in order to keep the height of the tree balance, it is a self optimizing data structure that uses the splay operation to keep frequently accessed items close to the root. These would be an advantage for someone who needs to use a priority queue in an application where they know the height of the tree where be small. They could gain energy efficiency overall from taking advantage of the small amount of energy that is needed to search for the last element in the tree when performing an enqueue or dequeue operation versus using an explicit heap implementation. This gives us insight under which situations that each implementation may advantageous to use over others.

3.9 Conclusion

Utilizing mobile devices to execute embedded discrete event simulations requires an understanding of how each component of the simulation affects overall energy consumption. We have conducted a study focusing on characterizing and understanding where power is consumed for four different priority queue implementations of the future event list in discrete event simulations. Our study compares the linked list, implicit heap, explicit heap, and splay tree implementations of the priority queue. We evaluate the average power used to execute a hold operation as we vary the queue size. Our results indicate that the implicit heap consumes the least power, energy, and time as queue size increases. The implicit heap can exploit local locality of memory references because its underlying data structure is an array, which utilizes contiguous memory locations. This helps ensure higher cache hit rates resulting in fast access and lower power consumption.

The linked list implementation does not exhibit this degree of memory locality and requires a search of time complexity O(N) for enqueue operations, leading to longer search time and overall more energy consumption than the tree implementations as the size of the queue increases.

The explicit heap implementation, like the linked list uses pointers to link nodes. This implementation's underlying data structure is a tree which performs enqueue and dequeue operations with O(log N) time complexity. This allows the time needed to search during the enqueue operation to be minimal but the access time required to access nodes is larger than the implicit heap because events may not be placed in contiguous memory.

We implemented models to predict energy consumption in performing enqueuer and dequeue operations for all four implementation to highlight the causes of energy consumption in executing enqueuer and dequeue operations. These prediction models show the relationship among the operations needed for each implementation to execute a priority queue operations on an embedded system. Our models show that as we perform a hold operation on priority queue sizes that cannot fit within the cache that the energy needed to perform a hold operation increases, but this effect can be captured by modifying constant values used by the model. Empirical measurements show excellent agreement between the model and measurements. We conclude that among the priority queues considered in this study the implicit heap is the best data structure to use in so far as the hold model represents typical behavior for the simulation program being used. We also conclude that exploiting locality in memory offers a significant benefit to reducing overall energy consumption.

4 ENERGY FOR HLA DDM APPROACHES

Data distribution management (DDM) is a set of services defined in the High Level Architecture to distribute information in distributed simulation environments [7]. DDM services are implemented by Run-Time Infrastructure (RTI) software. Several different approaches to implementing the DDM services have been proposed including grid-based implementations, region-based implementations, and hybrid approaches that utilize a combination of ideas from the grid and region-based approaches [30]. These approaches have certain computation and communication requirements that are necessary to perform DDM operations. To our knowledge no work has been conducted to date examining DDM services from the standpoint of energy consumption. This is the focus of the work described here. We examine the well-known region based and grid based approach along with two other DDM approaches to gain an understanding of the energy consumption properties of using DDM in energy constrained environments.

Dynamic Data Driven Application Systems (DDDAS) are applications that continuously monitor, analyze, and adapt operational systems in order to better assess and/or optimize their behavior [1]. Applications arise in many areas such as natural disaster management, transportation and manufacturing, among others [4, 77, 78]. Many DDDAS applications involve sensing and computation on mobile devices, utilizing communications through wireless networks. Energy consumption in these applications is a major concern because battery life often limits the effectiveness of DDDAS applications utilizing mobile platforms.

With the growing use of mobile devices research in the area of mobile computing has increased. Mobile devices have the ability to provide real-time information that can be used as input to real-time applications such as DDDAS. Sensors such as GPS, cameras, accelerometers, and environmental sensors are becoming more widely deployed. The dependency on battery power for mobile devices makes it vital to understand the energy consuming properties of running applications on mobile systems.

We are concerned with understanding how to implement and use DDM to provide real-time information to run embedded DDDAS applications within mobile devices. These DDDAS applications are used to make predictions and analyze systems such as traffic networks.

4.1 Data Distribution Management

Data Distribution Management services are used to reduce traffic flow over the network. DDM services are defined in the High Level Architecture Interface Specification. The services are implemented by Run-Time Infrastructure software. DDM utilizes an N-dimensional coordinate system called a routing space to represent, for example, a geographical area. Federates express their interest by defining subscription regions that characterize the information they are interested in receiving. Each message is associated with a publication region to characterize the content of the message. If an overlap is detected between a message publication region and a federate’s subscription region, the message is sent to that subscribing federate. The region based and grid based DDM approaches are the most well known approaches to implementing DDM services. Several other approaches have been proposed and implemented to overcome the drawbacks of these approaches.

4.1.1 Region Based Approach

The region based approach manages interests by performing a matching computation between all publication and subscription regions defined within the routing space [79]. This approach incurs a $O(N^2)$ computation cost where N is the number of subscription / publication regions. Although costly this approach is efficient in communicating messages. A multicast group is defined for each publication region within the routing space. Once an overlap is detected between a publication region and a subscription region, the subscribing federate joins the multicast group associated with that publication region. If a subscription region changes, the new subscription region must be compared against each other publication region to determine which groups it should leave or join. Similarly, if a publication region is changed, it must be compared against every other subscription region to determine the new composition of its multicast group. This approach is sometimes called the brute force approach because it is a direct implementation of the DDM services.

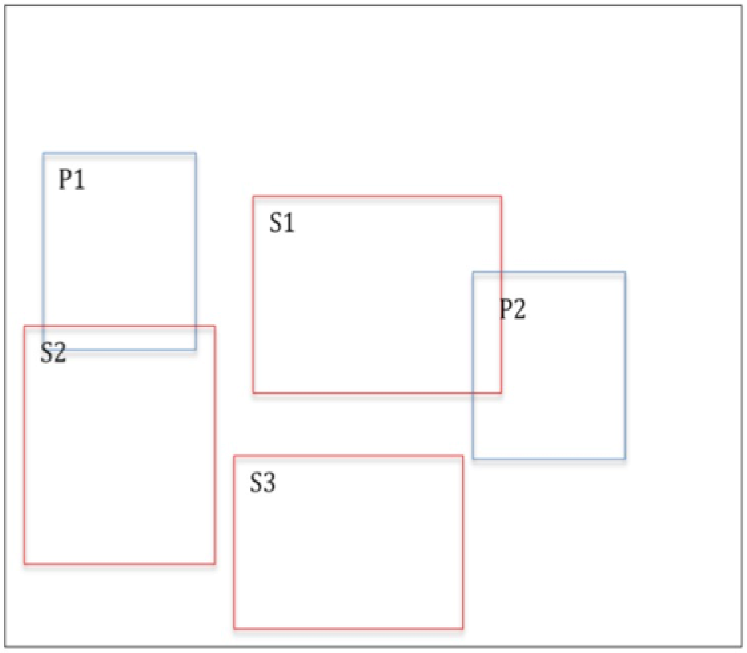

Figure 4-1: Region Based Approach

An example illustrating this approach is shown in Figure 4-1. Two multicast groups are created, one for P1 and one for P2. Federates subscribing the region S2 join the group for P1 and those subscribing to S1 subscribe to the group for P2. Messages associated with publication region P1 are thus routed to federates subscribed using region S2.

4.1.2 Grid Based Approach

Grid based DDM aims to eliminate the need to directly match regions against each other to determine region overlap between publication and subscription regions. The grid based approach implements DDM by partitioning the routing space into fixed size grid cells [80]. A multicast group is defined for each grid cell within the grid structure. The grid cells are used to determine overlap between publication regions and subscription regions within the routing space. Federates determine those grid cells for which their publication regions overlap and send each message to the associated multicast group(s). Similarly, federates subscribe to groups associated with cells that overlap with their subscription regions. The grid based approach is considered to be more computationally efficient and scalable than the region based approach.

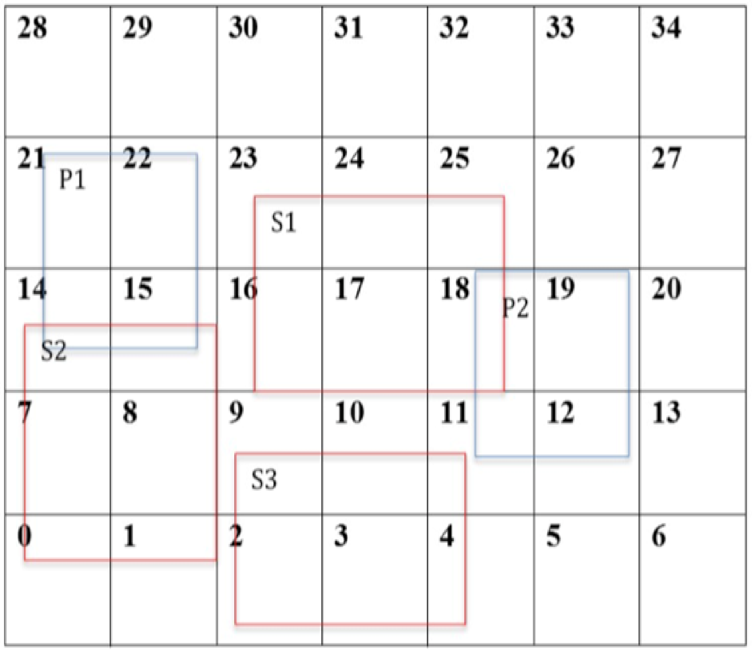

Figure 4-2: Grid Based Approach

The drawback to the grid-based approach is it lacks communication efficiency. Subscribing federates can receive irrelevant and duplicate messages under this approach. The example in Figure 4-2 illustrates the grid-based approach. A message sent with publication region P1 is sent to groups 21, 22, 14, and 15. Two of these messages are sent to federates subscribed to region S2, resulting in duplicate messages. Similarly, messages sent with publication region P2 are sent to federates subscribed to S3 through multicast group 11, even though P2 and S3 do not overlap.

4.2 Energy tradeoffs

Tradeoffs arise concerning energy consumption for different data distribution management approaches. Every DDM approach must perform operations such as: interest matching, processing changes to regions, and communicating distributed simulation messages using the DDM services. The difference among DDM approaches concerns how they perform and handle these operations and the effects they have on energy consumption.

4.2.1 Region Based Approach

The region-based approach typically uses a central controller to handle DDM operations. The central controller is responsible for comparing all publishing regions against all subscription regions in order to determine which multicast groups subscribing federates should join. When a region changes publishing federates broadcast an update messages to all subscribing federates. The initial establishment of multicast groups using this approach is computation intensive. The matching computation is $O(N^2)$ where N is the number of publication or subscription regions. Each publication region must be compared against all subscription regions in the routing space to determine which regions overlap. Subsequent changes to a publication or subscription regions require computation of complexity O(N). Therefore, the frequency at which a federate changes its subscription regions has an significant impact on energy consumption.

Although the region-based approach is computationally intensive it is efficient in terms of communications. The region-based approach does not create irrelevant or duplicate messages that occur in grid based approaches, as discussed later. Subscribing federates only receive messages for publication regions that overlap with their subscription regions. The use of a central controller can limit scalability, however.

4.2.2 Distributed Region Based Approach

The use of a central controller can limit scalability. To address this issue we propose a distributed region based approach. This approach uses R controllers. Each controller is responsible for a fixed area of the routing space and manages publication and subscription regions that lie within their assigned area. Subscription regions that overlap more than one area require communication among the corresponding controllers responsible for those regions to determine the multicast groups a subscribing federate must join and which controllers must be notified when a region changes. This alleviates the scalability issue in the original region based approach. Since direct matching still occurs under this approach the communication of distributed simulation messages through the DDM system is still just as efficient as the original region based approach; federates do not send or receive duplicate or irrelevant messages thereby improving energy consumption relative to the grid-based approaches described later.

4.2.3 Fixed Grid Based Approach

The use of a grid structure greatly reduces the computation needed for DDM operations by avoiding the matching computations required by the region-based approaches. But, it leads to irrelevant and duplicate messages, as discussed earlier. The number of irrelevant and duplicate messages depends on the size of the grid cell. Thus, grid cell size plays an important role on grid based DDM approaches. Small grid cells can lead to many duplicate messages. While large grid cells reduce the number of duplicate messages, they increase the number of irrelevant messages.

4.2.4 Dynamic Sort Based Approach

The dynamic sort based approach aims to reduce the expense of executing region modification by using a time stepped approach. Traditionally committing region modifications every time step leads to costly computation and a high communication overhead. To overcome this drawback region modifications are usually committed periodically. In each time step of the simulation only a small subset of regions used in the federation are modified. The dynamic sort based approach was created by Pe in order to dynamically match region modifications [37]. The algorithm works based on the condition that the regions new upper (lower) bound is greater (less) than its previous upper (lower) bound limit. Once a change in the region's bound has been detected then that triggers the algorithm to perform re-matching between the two regions. The dynamic sort based algorithm maintains four-sorted list per dimension in order to not have wasteful processing during selective region modifications. The algorithm maintains a bounds list for each of the upper and lower bound of each dimension, which is generally two. The shift of a federate's extents in each bound does not tend to change by a large margin during each time step of the simulation. This small margin change is because the federate is bounded by a maximum speed. This means that it is not necessary to check against all update regions but only those within bounds of the new extents of the federate's subscribing region which is a small subset of all subscription regions apart of the federation.

4.2.5 Grid Filtered Region Based Approach

The grid filtered region based approach increases scalability and reduces the number of irrelevant and duplicate messages. Like the grid-based approach, this approach utilizes a grid structure to perform interest management operations. Boukereche created this approach in order to reduce the irrelevant message problem that incurs in the grid based approach [32]. This method uses a threshold parameter that indicates the percentage of area a region must cover within the grid cell before it joins the grid cell's multicast group. This threshold triggers matching computations between regions within grid cells. Those publication regions whose area of coverage of the grid exceeds the threshold are placed in a full coverage list for the cell. The same is done for publication regions that are placed in a full coverage list for subscription regions. These regions need not perform matching within the cell. Those regions whose area of coverage is below the threshold are placed in partial coverage lists for publishers and subscribers. All publication regions within the partial coverage list are matched against all subscription regions within the partial coverage subscriber's list. This reduces the number of irrelevant message by using direct matching computations.

4.3 Scenario

These experiments focus on a vehicle traffic application. The envisioned system is a Dynamic Data Driven Application System (DDDAS) using mobile devices. These devices, e.g., smart phones, utilize traffic volume data from sensors embedded within the arterial traffic network. A two-dimensional routing space is used that corresponds to a traffic network measuring 50 by 50 kilometers. The embedded sensors are placed at each intersection of the traffic network and detect vehicles traveling in the North and South direction. Sensor devices publish data such as measurements of traffic volume; thus, their publication region corresponds to an area surrounding the location of the sensor. Subscription regions represent areas of interest for a mobile vehicle and are typically distant from the location of the vehicle. Specifically, vehicles express interest in receiving data from sensors that are located 805 meters (0.5 miles) away from their current location in their current direction of travel, which is either North or South bound in the arterial traffic network. The embedded sensors have a sensor range of approximately 400 meters. Each moving vehicle is a subscribing federate and the embedded traffic sensors are publishing federates. The arterial road network contains vehicles that are assigned a random position within the arterial road network and travel either north or south, moving with a constant velocity of 20 meters per second.

4.4 Experimental Setup

These experiments utilized the LG Nexus 5x cellular phone with a Qualcomm Snapdragon 808 processor, 2GB memory, and 16GB storage as the mobile computing platform. The phone runs the Android version 6.0.1 (Android Marshmallow) operating system. Computation experiments are explicitly performed on the Jetson TK1 development boards with ARM A15 32-bit CPU with 4 cores operating at 2.3 GHz and 2 GB memory. Energy and power measurements were performed using a PowerMon2 power measurement system for measurements that were evaluated on the board [81]. The Trepn profiler application was used to perform measurements on the Android phone [57].

To evaluate the computation energy of the four DDM approaches – region based, distributed region based, fixed grid based, grid filtered region based, and the dynamic sort based – we perform all computations on the Jetson development board. Each approach's matching component was run on the board and the energy consumption was measured to determine the average amount of energy consumed.

To evaluate the energy needed for communication we assumed that one RTI client communicates with the development board through TCP sockets over a WLAN network. The client for this experiment was the Nexus 5x mobile device that performed all DDM operations. This was implemented by creating an Android application using Android Studio. All communication was performed between the mobile device, which acted as the central controller in the case of region-based approaches, and the publishing federate in the case of grid based approaches. Wireless communication was performed between the mobile device and the board with the device receiving 1000 byte messages and the board sending 1000 byte messages. The mobile device mimics the full operation of what would happen during communication under the traffic scenario described earlier. In the case of the region-based approaches the device sends a message to the server indicating its subscription region. The server (board) then performs matching between the client's region and the known other regions within the routing space to initially set up the multicast groups. The server then communicates with the client to indicate what multicast groups it should join. In the case of the grid based approach the client performs its own interest matching computation by determining which grid cells its subscription region overlaps, and then joins the designated multicast groups. Energy consumed by the client for messages sent and received under each approach was then measured.

All experiments assume that two central controllers are used for the distributed region based approach and the threshold for the grid filtered region based approach is set to 0.6.

The grid cell size is set to 20m x 20m, publication regions are 40m x 40m and subscription regions are 80m x 80m.

4.5 Energy for Computations: Initial Multicast Group Association

We define computation energy to be the energy used to compute the matching component of the DDM approaches. An evaluation of the energy used to determine the matching component of each DDM approach as the number of subscribing federates increase are shown in Figure 4-3.

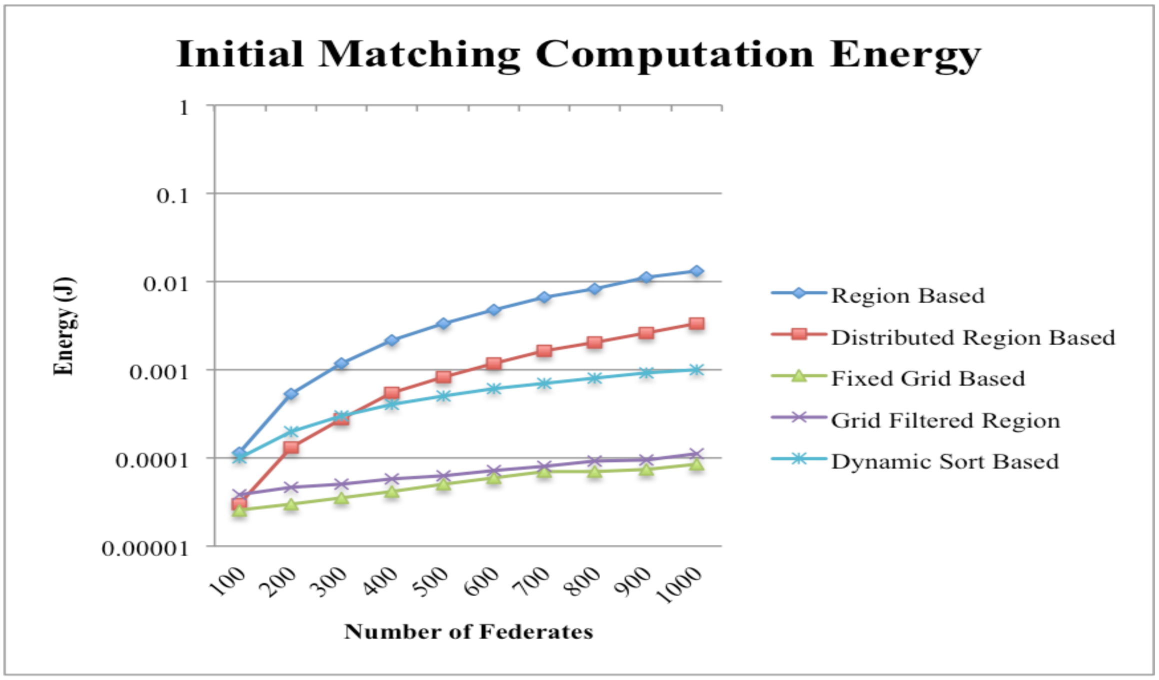

Figure 4-3: Computation Energy as federates increase

The results show the energy cost of using a purely region based DDM approach in comparison to grid based DDM. The region based matching cost is $O(N^2)$ to initialize DDM operations since every publication region is compared against every subscription region.

The distributed region based approach performs matching in the same manner as the region based approach. It designates a subset of regions to each central controller in the routing space based on the coordinates of the region. Each controller performs matching on the regions that are in their domain as well as those that cross multiple domains. This reduces the amount of energy consumption used for matching because the cost is divided among multiple controllers. Utilizing a subset of central controllers allows the number of regions that must be compared to be reduced which in turn reduces the energy consumption cost compared to the centralized region based approach.

The fixed grid based approach consumes the least amount of energy out of all the approaches compared in this study. The matching operation in this approach involves federates determining the grid cells with which their regions overlap. The increase in energy is due to each federate determining the overlapping grid cells and adding themselves to the list of federates in the multicast group.

In the grid filtered region based approach computation energy is similar to that of the fixed grid based approach. Federates again match themselves to grid cells by determining those cells with which their regions overlap. This approach performs matching by utilizing a threshold parameter that allows region matching to be conducted between publication and subscription regions within the grid cell whose area of coverage falls below the threshold. Those regions within each grid cell whose coverage is above the threshold automatically join the multicast group for that grid cell.

4.6 Energy For Computation: Update Multicast Join/Leave

Updating multicast group association is defined as the action of federates joining and leaving the current multicast to join the multicast group reflecting their current interest. Depending on the DDM approach in execution the operations of joining and leaving is performed differently. In order to implement a federate’s updated interest, the matching operation must be performed on a subset of the regions. This performance requires computation every time a modification request is made which will lead to some energy cost. An evaluation of what that energy cost is between different DDM approaches for this modification to be made under different update rates is reflected below. This experiment consisted of evaluating the energy consumption of five hundred federates modification their interest expression during a five minute time frame. For each update rate each approach evaluated the changes made to joining and leaving multicast groups to reflect each federate’s current interest.

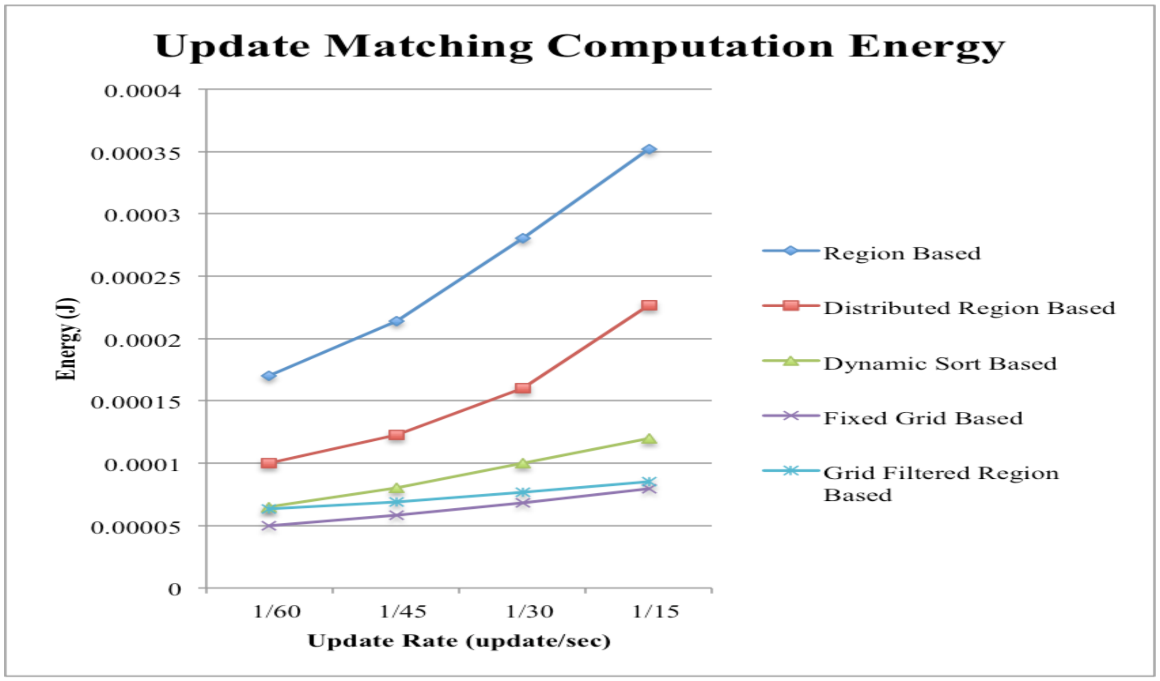

Figure 4-4: Energy of Join/Leaving Multicast Groups

Figure 4-4 shows the data collected from varying the rate at which modification are made to update multicast groups reflecting subscribing federate's interest. Over all the approaches evaluated updating frequently requires overall more computation causing an increase in energy consumption. The fixed grid based approach consumed the least amount of energy amongst the approaches evaluated because of its O(1) computation used to match regions to overlapping grid cells. The dynamic sort based approach keeps an updated sorted list of the extents of all regions in the federation eliminating the need to compare against all regions when an update region makes a modification to its extent. It decreases the computation that is needed in a strictly region based approach leading to less energy consumption compared to region based approaches.

4.7 Energy For Communications: Update Messages

Update messages are defined to be data messages sent by publishers to subscribers to transmit simulation data. Figure 4-5 shows the average amount of energy consumed by one RTI client (federate) in receiving messages for each update message sent by a publisher, including irrelevant and duplicate messages.

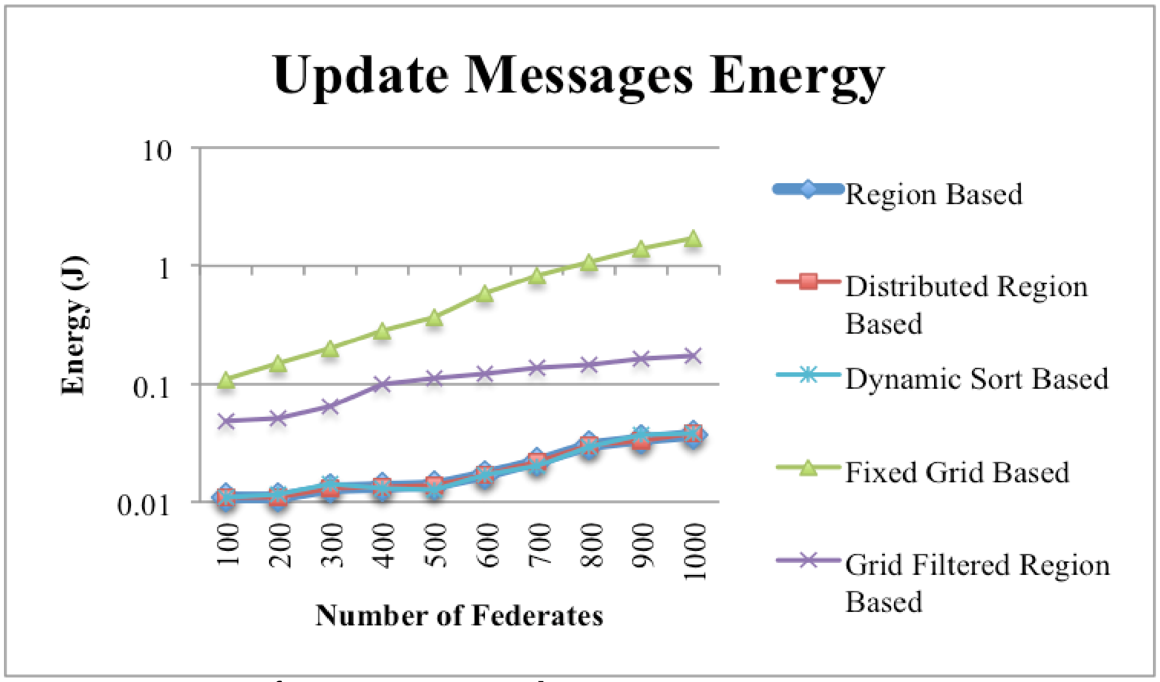

Figure 4-5: Energy of communicating update messages

The results show the communication tradeoffs that occur when utilizing region-based approaches compared to grid based approaches. The fixed grid based approach consumes the most energy among the five approaches while the region based and distributed region based approaches consume the least amount of energy.

The region based and distributed region based approaches consume more energy for matching computation but use less energy for communications. The matching computation ensures that no irrelevant or duplicate messages occur in the region-based approaches. This reduces energy consumption. Sending fewer messages reduces the time needed to send and receive messages.

The grid region based approach consumes less energy than the fixed grid approach but still uses more energy than the region based approaches and the sort-based approach. The use of the grid structure results in duplicate messages, but adding the threshold parameter eliminates the irrelevant message problem. The use of direct matching among those regions that do not exceed the threshold avoids irrelevant messages. This in turn eliminates irrelevant messages leading to a reduction in the number of messages received during an update and leads to a reduction in energy consumption. This experiment suggests that this approach achieves a balance between computation and communication with respect to energy consumption. It consumes little energy for computation and a small amount of energy for communication.

4.8 Energy for Communications: Constraining Publication Regions

Constraining the publication region to one grid cell will allow grid based DDM approaches to gain efficiency in federate interest matching. By constraining the publication regions irrelevant and duplicate messages are eliminated. Under this constraint subscription regions cannot overlap grid cells that partially contain a publication region due to the fact the publication region occupies the entire grid cell. This eliminates the irrelevant message problem. Duplicate messages also will not occur under this constraint because publication regions only occupy one grid cell so update messages are sent to only one multicast group. In the current grid based DDM approach publication regions that overlap multiple grid cells join multiple multicast groups and any subscription region that is part of those groups will receive the update messages produced by the publishing federate. If a subscribing federate joins multiple multicast groups that a publication region has also joined they will receive the same message multiple times leading to duplicate messages.

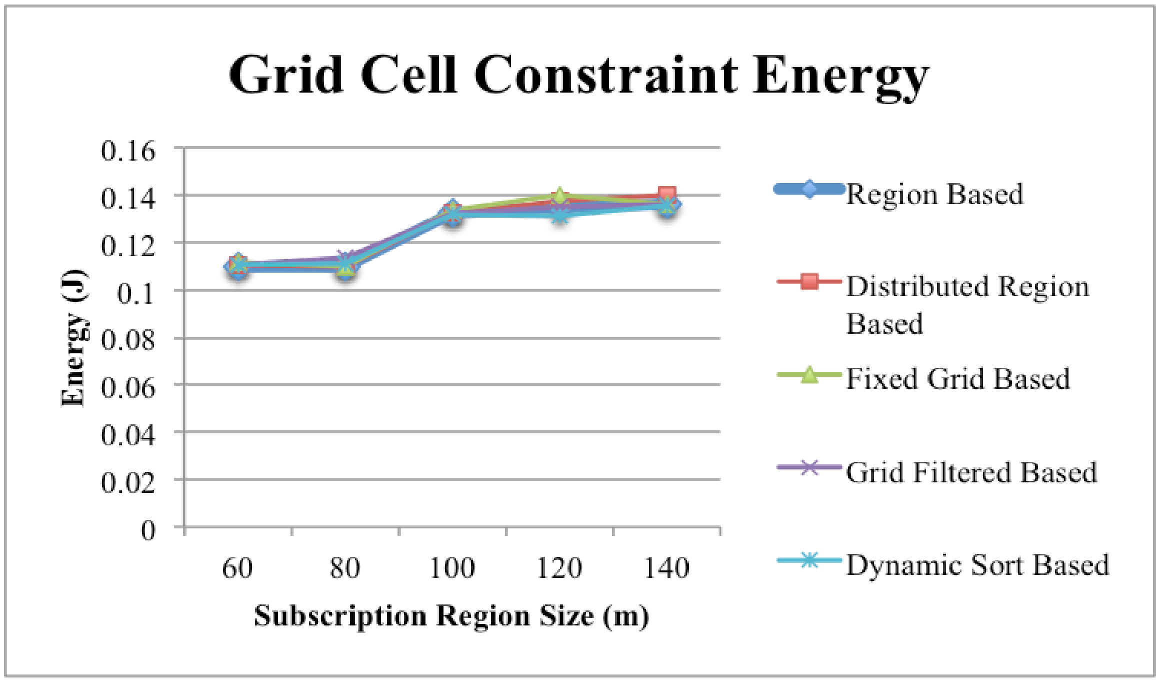

Figure 4-6: Publication region constrained to one grid cell

Figure 4-6 shows the energy consumption when constraining publication regions to one grid cell. We see that as we vary the size of the subscription region we increase the number of update messages; but, the effect is the same regardless of using the region based or grid based approach. This result is due to the fact that there are no irrelevant messages. The results also show that duplicate messages are eliminated under this constraint since the publication region only matches to one grid cell it only send update messages to one multicast group. Subscription regions under grid-based approaches can only overlap any publication region once since it is contained to one grid cell. This experiment illustrates that placing constraints on DDM regions enables one to gain the communication efficiency of region-based approaches and the computation efficiency of grid based approaches. Constraining publication regions to one grid cell allows the region based and grid-based approaches evaluated in this study to consume relatively the same amount of energy. This constraint may not always be suitable for all scenarios, especially when the publication regions are not all the same size.

4.8.1.1 <u>Energy for Communications: Constraining Publication Region to N Grid Cells</u>

Varying the number of grid cells that the publication region encompasses allows us to understand the effects that grid cell size on the number of update messages a federate receives under different DDM approaches. We evaluate this effect by varying the number of grid cells covered by a publication region.

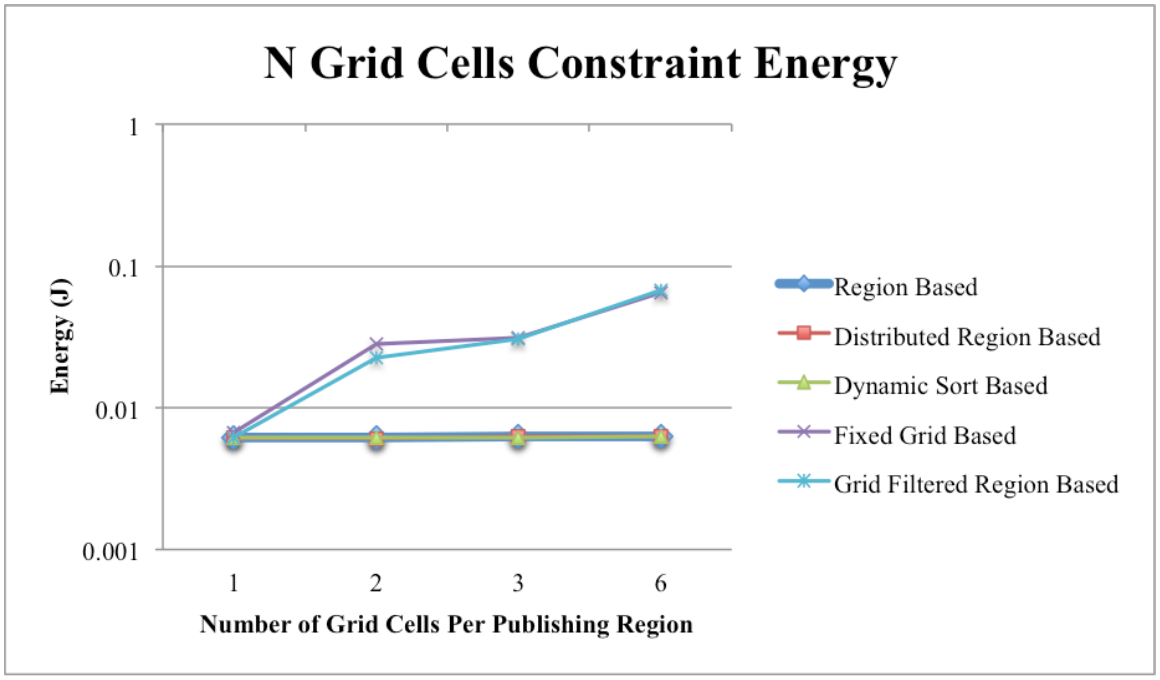

Figure 4-7: Publication region constrained to N grid cells

Figure 4-7 shows that as we increase the number of grid cells encompassed by the publication region we increase the number of duplicate messages that a subscribing federate will receive. Since the publication region in this experiment is bounded to the extents of the cells there are no irrelevant messages. Fixing the publication region to the bounds of the grid cells allows duplicate messages to occur under the grid filtered region based approach because every covered grid cell will automatically join a multicast group because they will always surpass the threshold boundary. Utilizing this constraint makes grid filtered region based use just as many messages as fixed grid based would because they both will not incur irrelevant messages but duplicate messages will still occur.

4.9 Varying Grid Cell Size

Grid based approaches are directly affected by the grid cell size. Smaller grid cells tend to lead to more duplicate messages and larger grid cells tend to lead to more irrelevant messages. In the following experiment we examine the effect that grid cell size has on energy consumption. We examine this effect while leaving the publication and subscription regions a static size for each grid cell size.

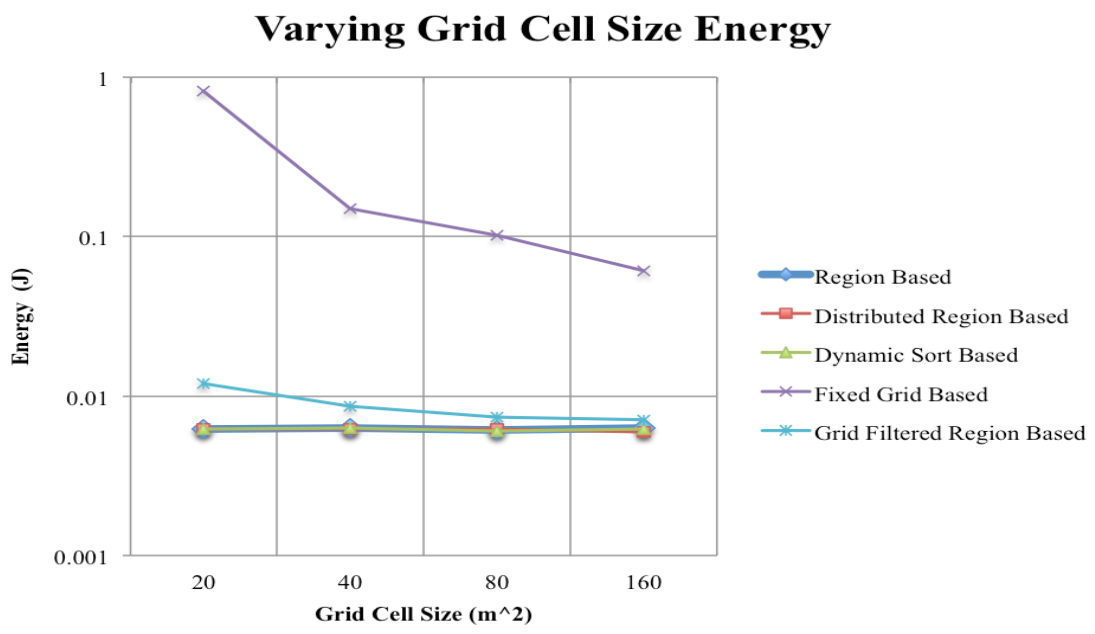

Figure 4-8: Varying grid cell size

Figure 4-8 shows the impact that grid cell size has on grid based DDM approaches. We see that larger grid cell size greatly impacts the energy consumption of the fixed grid based approach. An interesting phenomenon is the small effect that grid cell size has on the grid filtered region-based approach as the grid cell size increases. Here we see that a change in energy consumption for this approach does not occur until the grid cell size is at the largest size evaluated where we see a decrease in energy. This shows that utilizing a threshold of 0.6 to trigger direct matching between regions that fall below this threshold in the grid filtered region based approach gives a trend similar to a region based approach energy consumption when large grid cell sizes are used.

4.10 HLA DDM Energy Models

The work presented so far in this chapter examined on a broad level the energy and power consumption of using different High Level Architecture data distribution methods as a means to setup communication between federates in a distributed simulation. To gain a deeper understanding of how and which HLA DDM method is useful in different situations it is beneficial to have predictive models. Below we illustrate our prediction models for predicting computation and communication energy consumption of four different data distribution management methods.

4.10.1 HLA DDM Initial Computation Energy Models

The following section explores the energy dependent components for each data distribution approach that was evaluated to create an energy prediction model for initial computation of multicast group assignment for federates in the distributed simulation.

4.10.1.1 <u>Region Based: Initial Computation</u>

The region based DDM method performs computation by comparing every subscribing region and publication region in the federation to determine overlapping regions. This method is dependent upon a central controller that performs all the matching computations. Our prediction model shows the relationship between the number of matches that must be performed for each subscribing federate to determine those publishing federates from which they will receive updates.

$$\text{Energy} = C_{1+}C_2N_c$$

Table 4-1:Region Based Computation Model Constants

C_1	C_2
6.00E-10	.1184

The prediction model for the region based approach quantifies energy consumption as a function of the number of comparisons needed. The base power needed for each comparisons is indicated by the value C_1 and the rate of change is shown by the value of rate in Table 4-1.

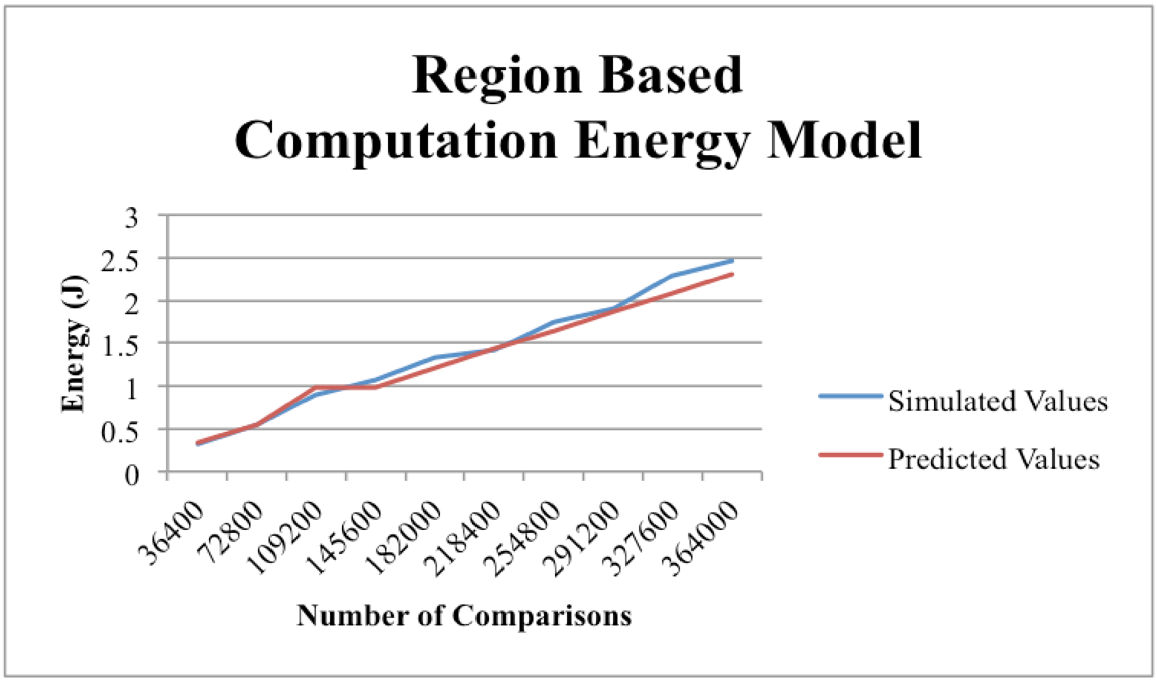

Figure 4-9: Region Based Initial Computation Energy Model

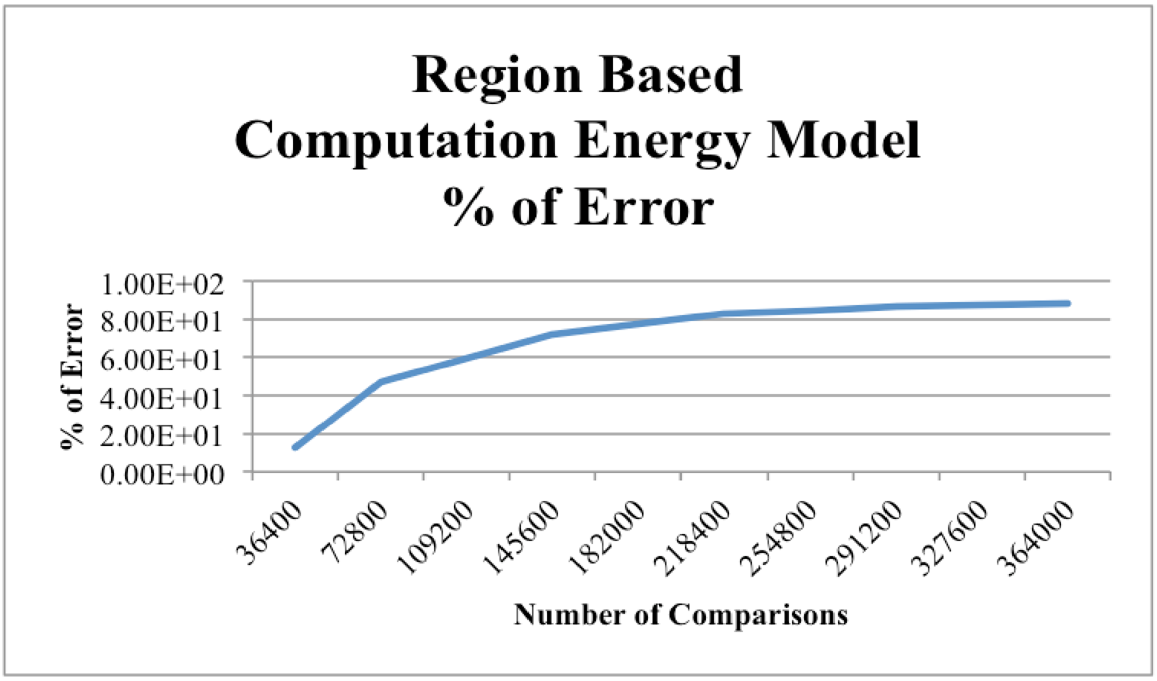

Figure 4-10: Region Based Computation Energy Energy Model % of Error

Figure 4-9 compares the experimental results with values computed by the model. The figure shows that predicted values are close to experimental measurements. Figure 4-9 verifies that the energy needed for the region based approach has a direct relationship to the number of computations that are performed between publishing and subscribing regions .

4.10.1.2 <u>Distributed Region Based: Initial Computation</u>

Distributed region based DDM is an approach designed to be more scalable than the region based DDM approach. The routing space is divided amongst a give number of regional controllers. Each controller is responsible for performing the matching operation between publishing and subscribing federates in their area of responsibility in the routing space. Those federate’s whose publication regions or subscription regions are within multiple sub areas of the divided routing space amongst regional controllers, are matched against publishers and subscribers in those sub regions as well by those regional controllers. Those regional controllers must communicate in order for the matching to take place. The model for the distributed region based approach follows the same formula for the region based computation, determines the amount of energy needed to compute the overlaps between federates to determine who communicates with whom.

Table 4-2: Distributed Region Based Computation Model Constants

C_1	C_2
3.00E-06	.0208

The distributed region based approach is similar to the region based approach. If we compare the constant values in table 4-2 and table 4-1 we see that the distributed region based constants are higher. The constant values here reflect the overall energy consumption needed to compute the distributed region initial computation for using four regional controllers. Even though the constants are higher the overall energy consumption is lower because we have to compare fewer regions under this approach.

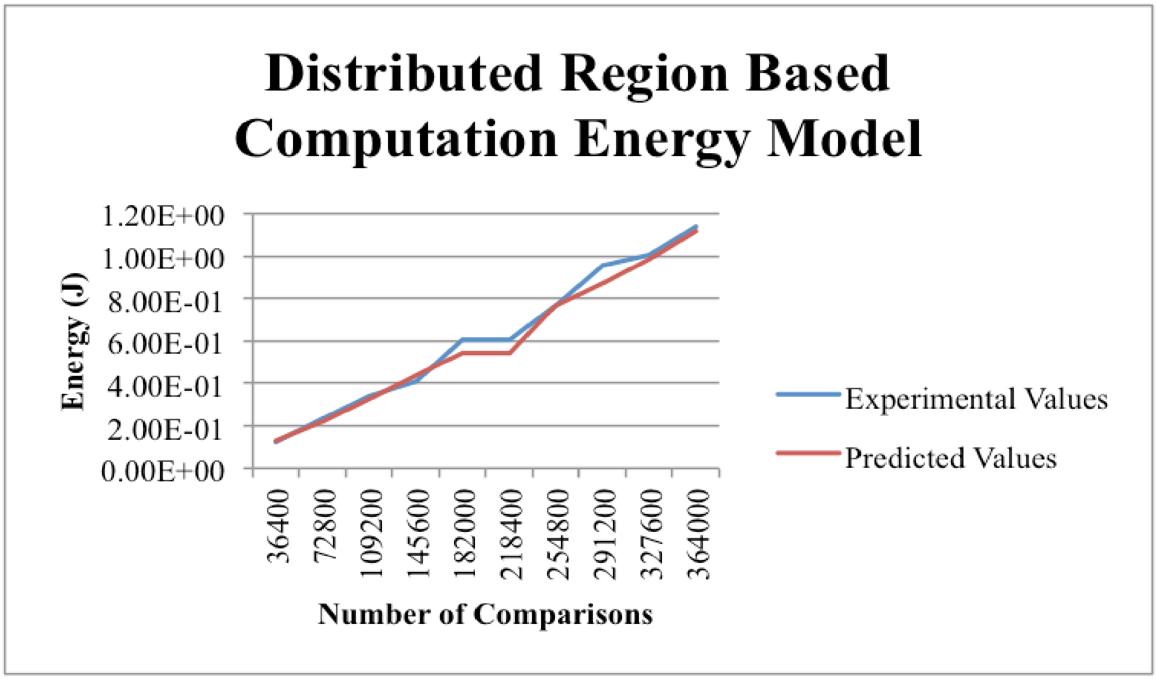

Figure 4-11: Distributed Region Based Initial Computation Energy Model Results

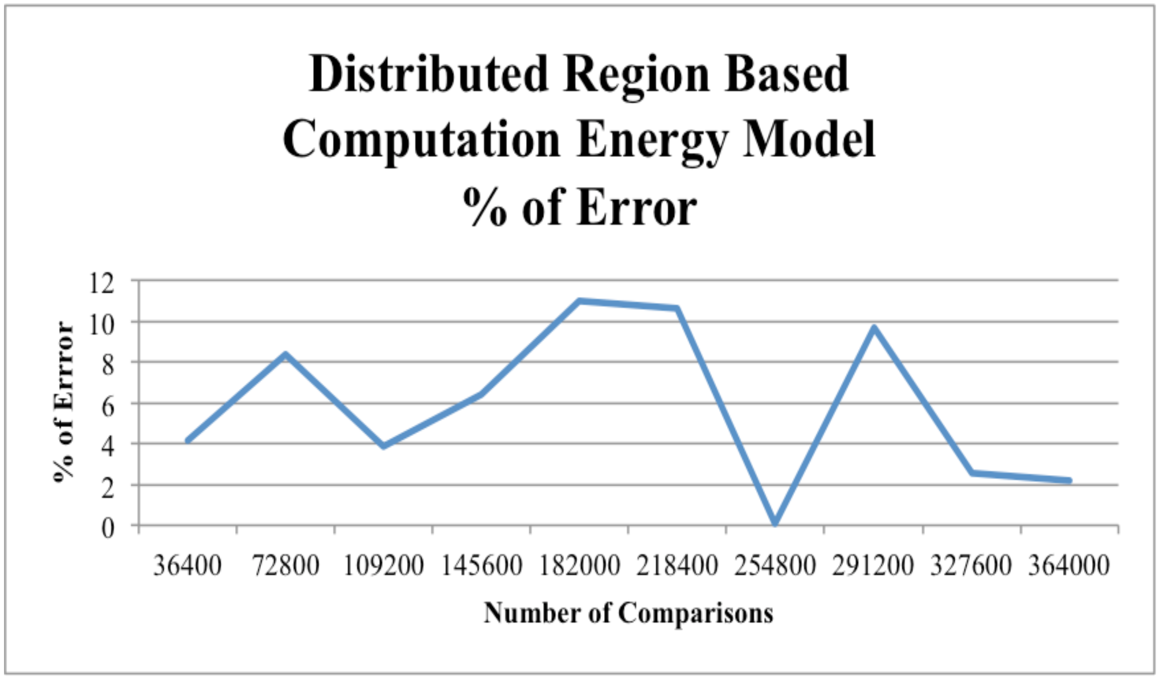

Figure 4-12: Distributed Region Based Computation Initial Computation Energy Model % of Error

Figure 4-11 compares model predictions with experimental measurements in computing the initial multicast group assignments. Comparing figure 4-9 and figure 4-11 we see that the distributed region based approach is able to compute the same initial multicast group assignments by using fewer comparisons amongst all of the controllers. The distributed region based approaches requires fewer comparisons and is more scalable because it does not depend on a central controller.

4.10.1.3 Fixed Grid Based: Initial Computation

The fixed grid based approach avoids the initial matching computation required by the region based approach. Each federate determines the grid cells that overlap with their subscription regions and joins the multicast group assigned to those grid cells. Federates send messages to the multicast groups assigned to the grid cells overlapping their publication region. The model for this approach predicts the amount of energy for each federate to determine and join the multicast groups of their overlapping grid cells:

$$\text{Energy} = C_1+C_2G$$

The energy needed for the fixed grid based approach is linear in the number of grid cells that each federate must join, which depends on the size of the federate's subscription region and the size of the grid cells. C_1 represents the base energy needed to perform the fixed grid based computation and C_2 is the amount of energy needed for each grid association. G represents the number of grid cells that federates associate themselves to during initial computation for determining multicast group assignment. The values in table 4-3 show the constant values for the experiments performed here. The grid cells used here are 20 meters by 20 meters and the subscription regions are 80 meters by 80 meters. Each subscription region covers sixteen grid cells.

Table 4-3: Fixed Grid Based Computation Model Constants

C_1	C_2
.0175	4.00E-06

Comparison of model predictions and measurements for the initial computation in the fixed grid based scheme are shown in figure 4-13 and figure 4-14.

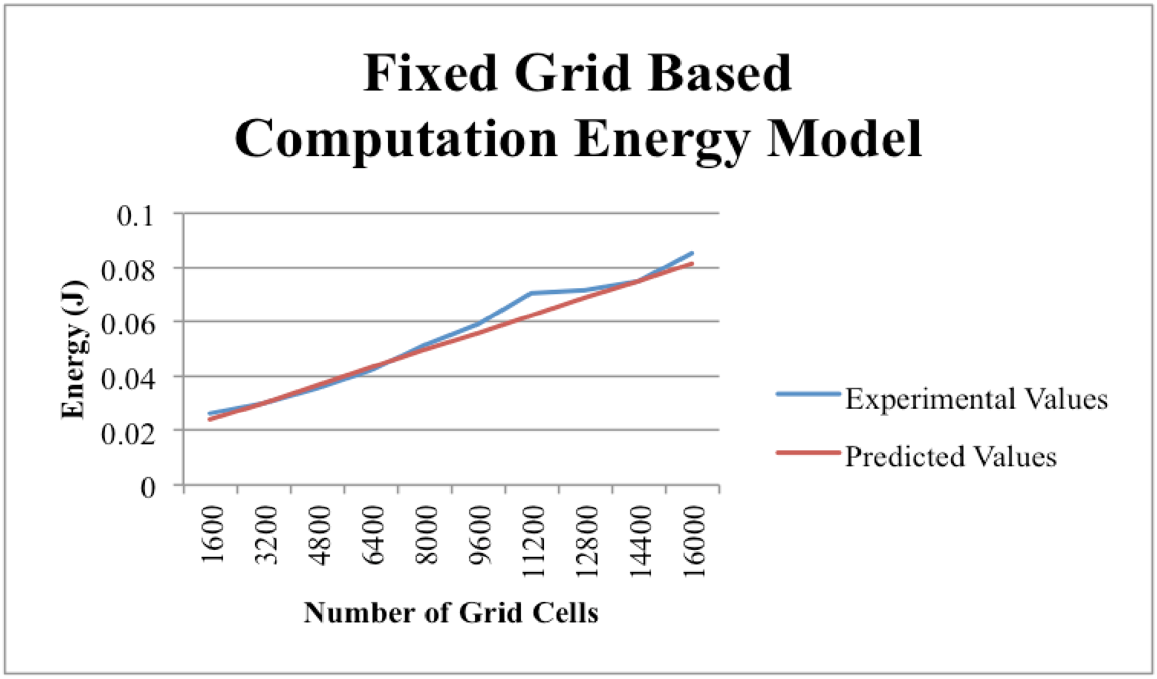

Figure 4-13: Fixed Grid Based Initial Computation Energy Model

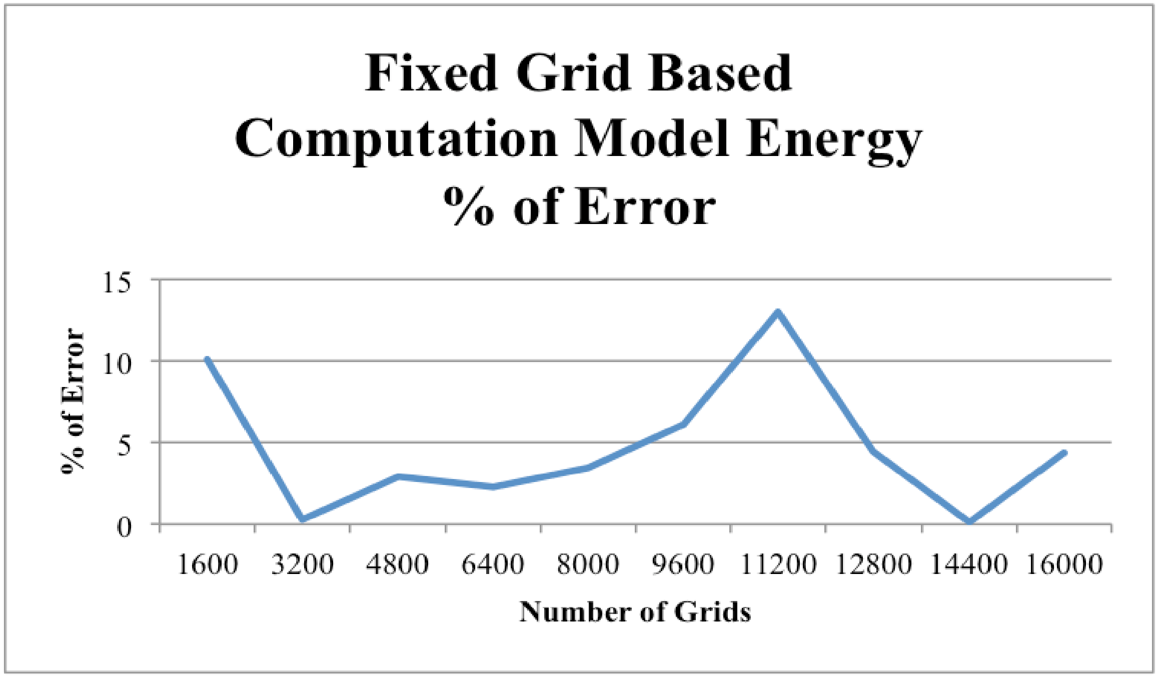

Figure 4-14: Fixed Grid Based Computation Energy Model % of Error

Model predictions agree well with measurements in computing multicast group assignments for grid cells as the number of grid cells increases, although a few measurements yielded differences ranging from 10% to 15%.

4.10.1.4 Grid Filtered Region Based: Initial Computation

The grid filtered region based method reduces the number of irrelevant messages in the fixed grid based approach. An irrelevant message occurs when a publication and subscription region both overlap with a common grid cell but the regions do not themselves overlap. Recall that this approach uses a threshold value is used to reduce the number of irrelevant messages. The threshold reflects the percentage of the grid cell area the federate's region must cover in order to be apart of the grid cells multicast group. Each grid cell maintains four list, publishing federate fully covered, subscribing federate fully covered, publishing federate partially covered, and subscribing federates partially covered. All publishing federates apart of the publishing federate fully covered list and publishing federates apart of the subscribing federate fully covered list automatically join the multicast group of that grid cell. All publishing federates apart of the publishing federate partially covered list are checked for direct overlap between all subscribing federates of the subscribing federate partially covered list, federates who have a direct overlap join the multicast group of the grid cell.

Table 4-4: Grid Filtered Region Based Computation Constants

C_1	C_2
.0283	5.00E-06

The grid filtered region based approach, like the fixed grid based approach, performs the initial computation operation by determining those grid cells with which that federate’s regions overlap. In this approach one additional step is needed to determine if direct comparisons between regions within the grid cell need to be performed. C_1 in our model indicates the base energy needed to perform computation; C_2 represents the energy needed to perform each grid association and threshold hold comparison. G represents the number of grid cells. We compare our model predictions and experimental measurements in Figures 4-15 and 4-16.

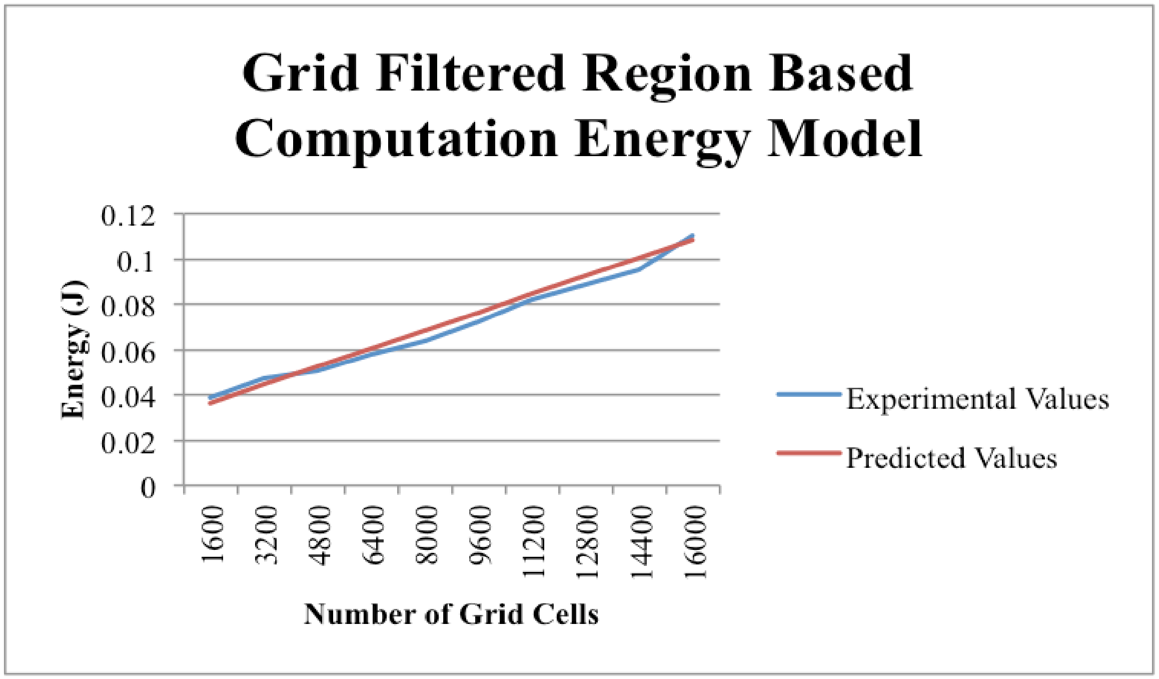

Figure 4-15: Grid Filtered Region Based Initial Computation Energy Model Results

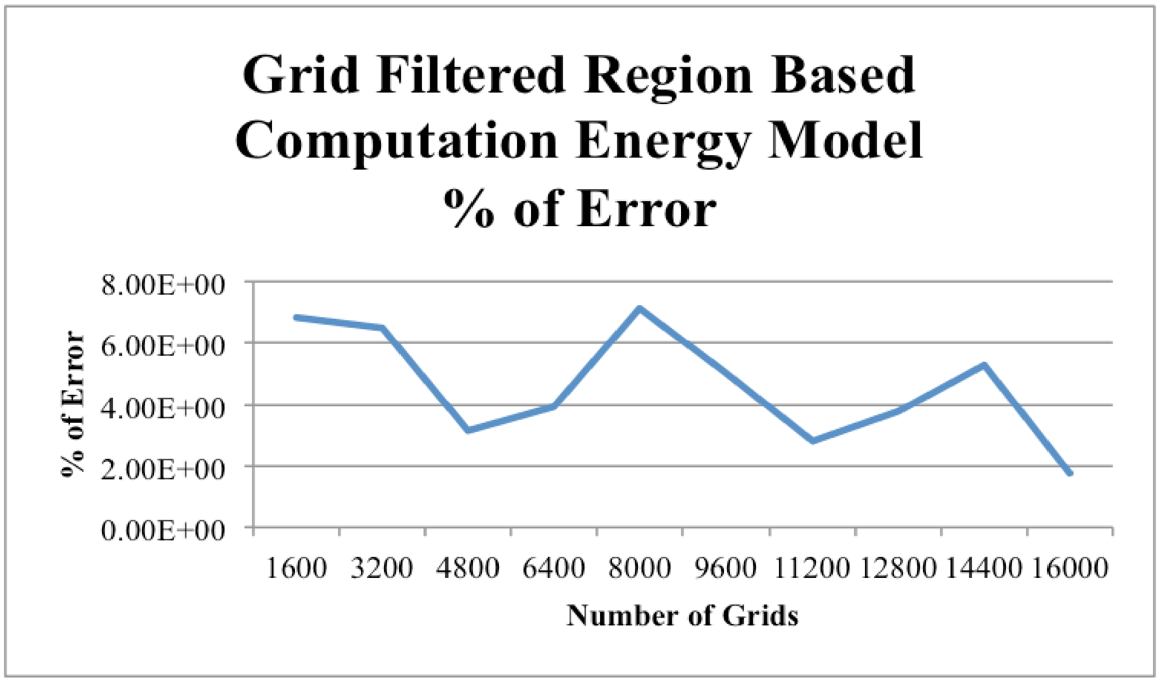

Figure 4-16: Grid Filtered Region Based Initial Computation Energy Model % of Error

The experimental results are based on the scenario presented in section 4.3. In comparing figure 4-12 and figure 4-13 we see that the overall energy consumption required to compute the initial multicast group association for grid based computation is less than the energy required for grid filtered region based. This is due to the extra computation needed to compare the threshold in each grid cell to help alleviate the irrelevant message problem in the fixed grid based approach.

4.10.2 Model for Update Messages

This section outlines the relationship between the number of messages needed under each approach for the first update that is sent by federates for the scenario presented in section 4.3. We present the energy models and compare predictions to experimental results for the region, distributed region, fixed grid based, and grid filtered region based data distribution management methods.

4.10.2.1 <u>Region and Distributed Region Based Update Message Model</u>

The region and distributed region based approaches produce the same number of messages. The differences in these two approaches concerns the determination of multicast group assignments and was presented in sections 4.1.10.1 and 4.1.10.2. Below we present the energy prediction model for receiving update messages.

$$\text{Energy} = C_1 + C_2 M$$

Table 4-5: Region and Distributed Region Based Constants

C_1	C_2
.0003J	.0049J

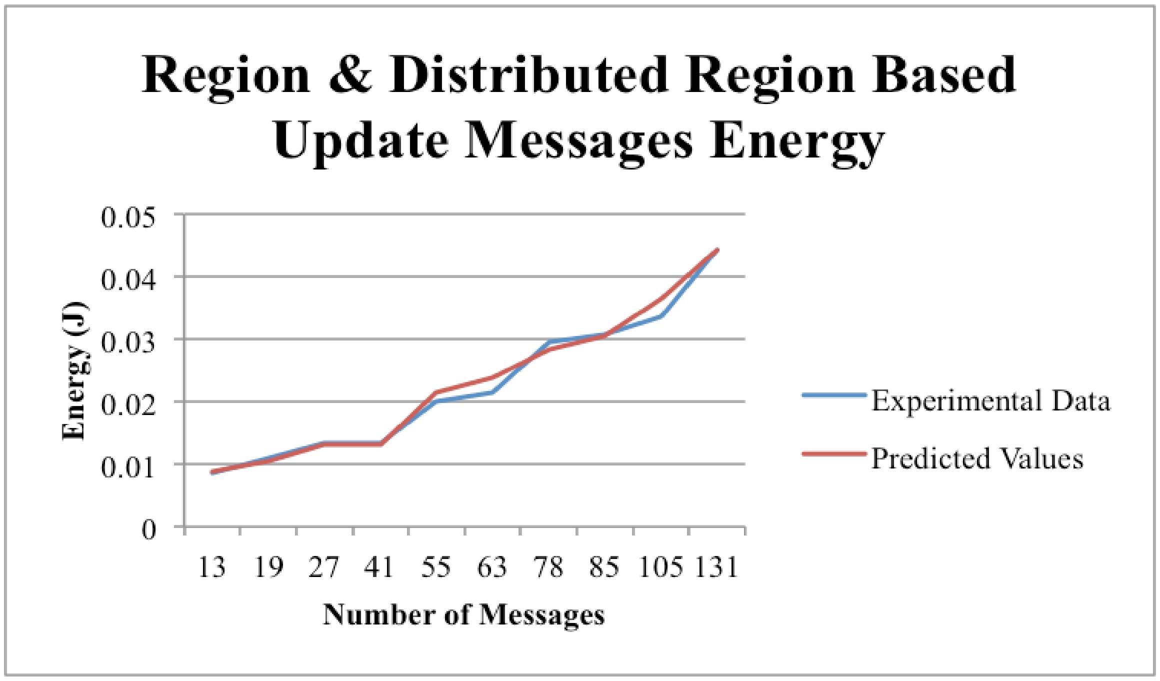

Figure 4-17: Region & Distributed Region Based Update Messages Energy

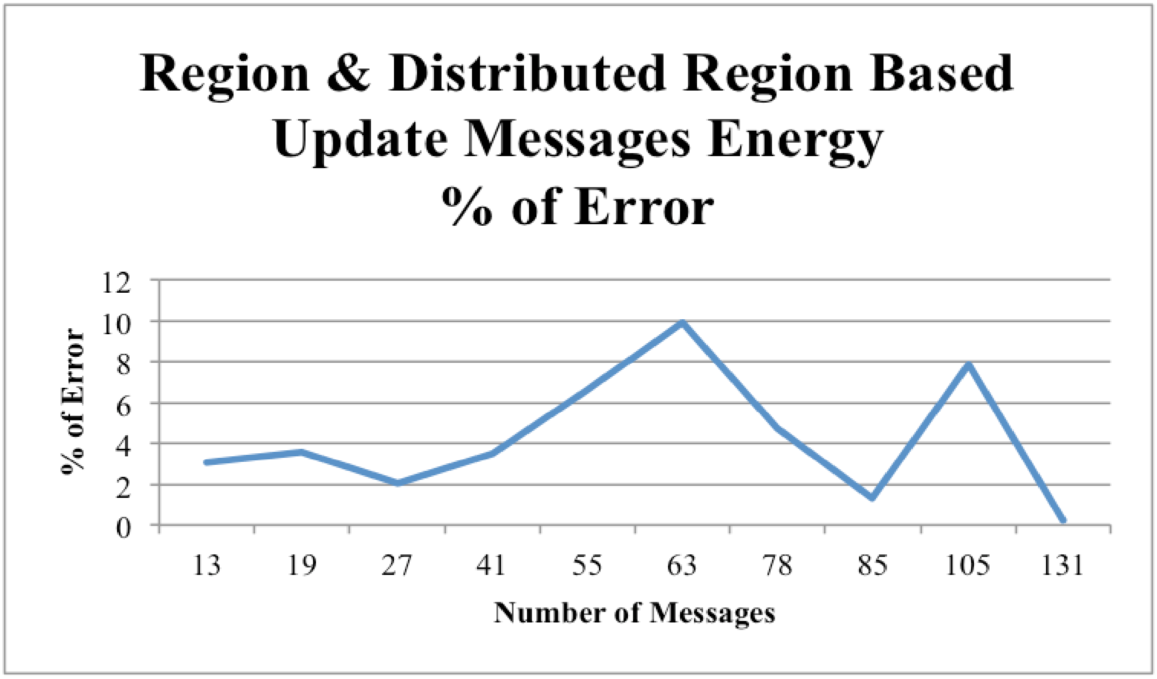

Figure 4-18:Region and Distributed Region Based Update Messages Energy Model % of Error

4.10.2.2 Fixed Grid Based Update Message Energy Model

Each grid cell in the routing space is assigned a multicast group federates whose subscription regions overlap that grid cell join the multicast group assigned to the cell. Energy consumption is a function of the number of messages produced by an update (M), i.e., the number of cells with which the publication region overlaps.

$$\text{Energy} = C_1 + C_2M$$

C_1 represents the baseline amount of energy needed to receive messages and C_2 represents the amount of energy expended to receive each message. The values for these constants are shown in Table 4-6 below.

Table 4-6: Fixed Grid Based Update Constants

C_1	C_2
0.0214J	.000269J

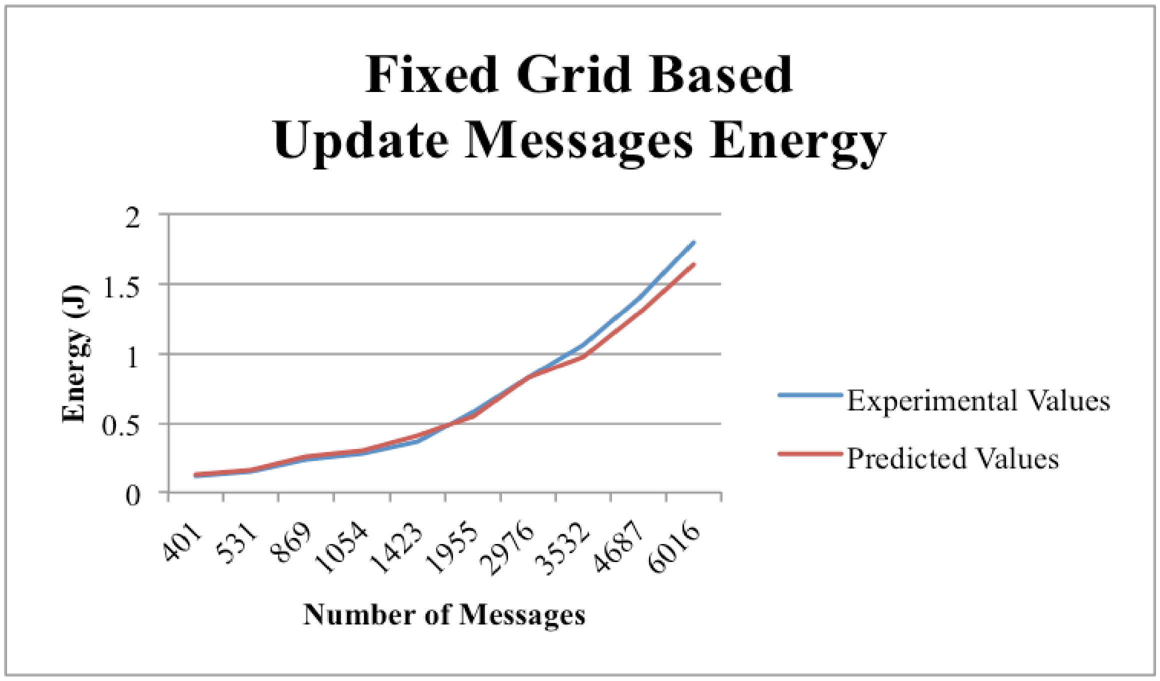

Figure 4-19: Fixed Grid Based Update Messages Energy

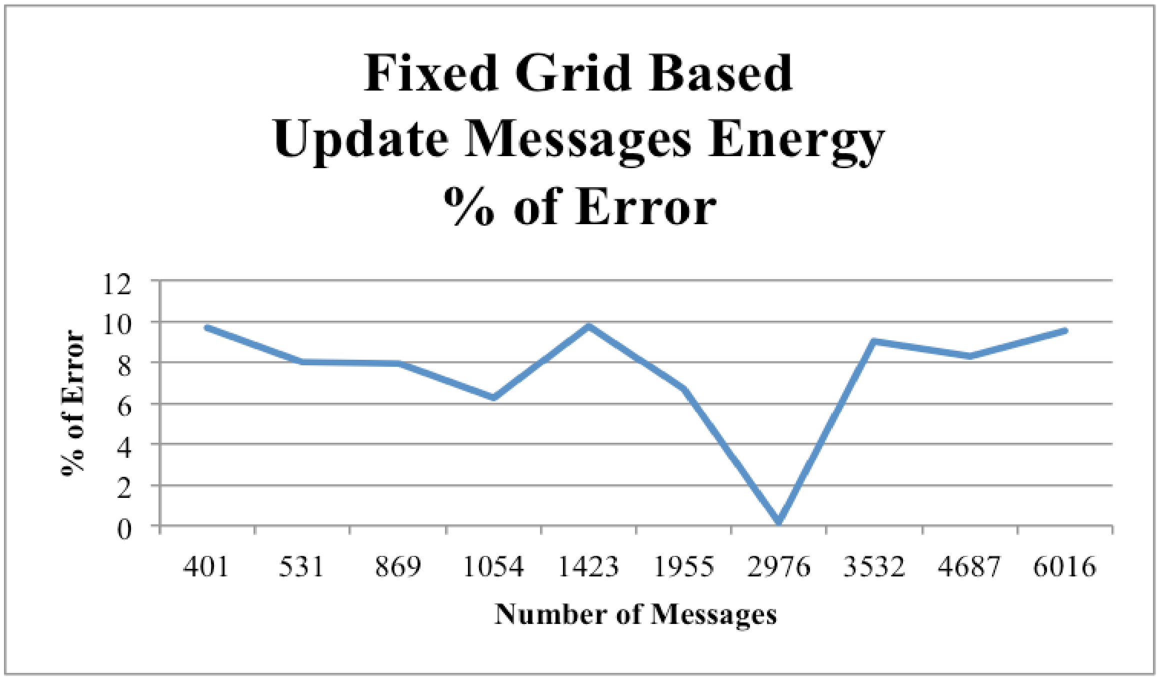

Figure 4-20: Fixed Grid Based Update Messages Energy Model % of Error

4.10.2.3 Grid Filtered Region Based Update Message Energy Model

The grid filtered region based approach uses a grid structure like the fixed grid based approach as well as an additional threshold parameter. We show below the correlation between the number of messages produced and the energy consumption expended to receive those messages below.

$$\text{Energy} = C_1 + C_2M$$

We see this same relationship for all methods evaluated. C_1 represents the baseline energy needed to receive messages. C_2 represents the amount of energy needed to receive each message. M represents the number of messages received. The constant values for these variables for grid filtered region based approach are reflected in table 4-7 below.

Table 4-7: Grid Filtered Region Based Update Constants

C_1	C_2
.0206J	.000251J

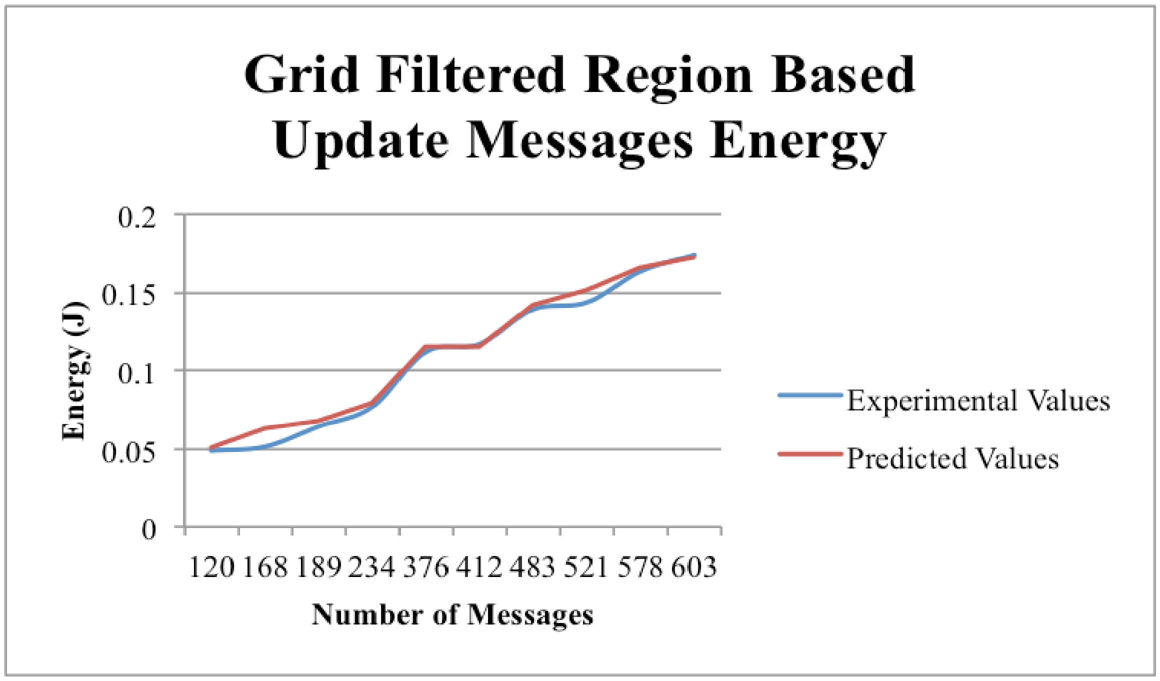

Figure 4-21: Grid Filtered Region Based Update Messages Energy Model

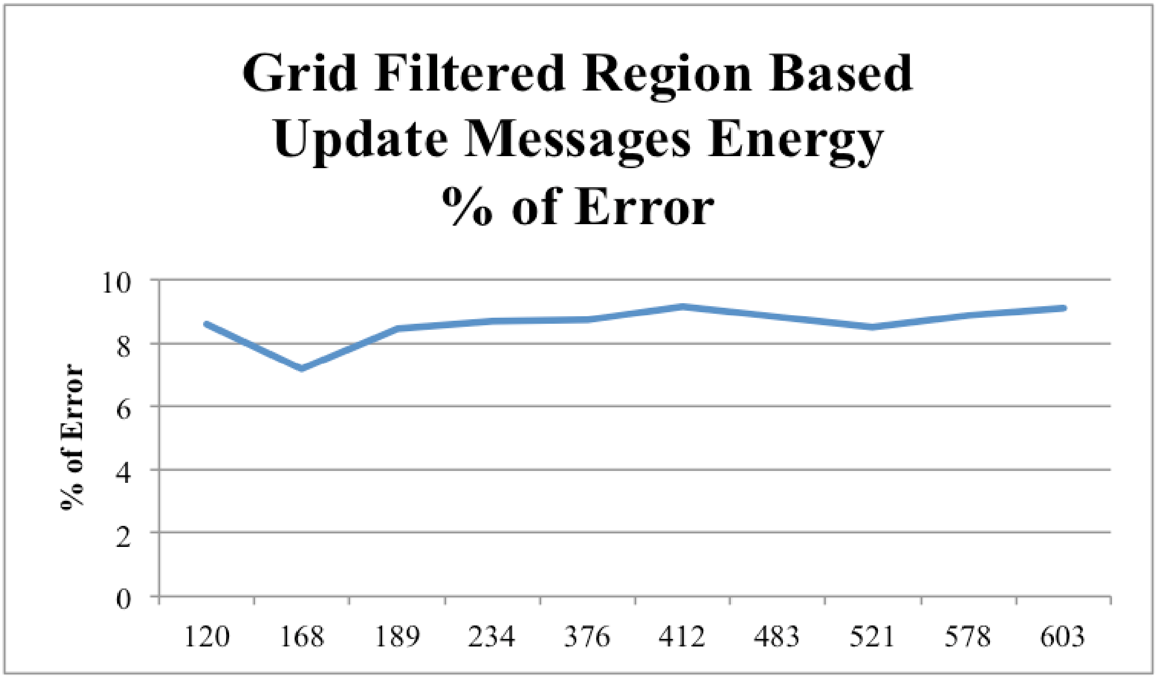

Figure 4-22: Grid Filtered Region Based Update Messages Energy Model % of Error

4.11 Conclusion

Using data distribution management in an energy-constrained environment requires an understanding of the energy consumed by the components of DDM operations. We analyze the energy required for computation and communication operations of five DDM approaches. Our experimental results quantify the high amount of energy consumption required in a centralized region based approach. We also conclude that in the area of communication that grid based DDM approaches can be costly in energy consumption when sending update messages due to the occurrence of irrelevant and duplicate messages. To overcome this problem we examined the effects of adding constraints on overall energy consumption when a DDM update is performed. We conclude that restricting publication regions to one grid cell eliminates irrelevant and duplicate messages; but, utilizing this constraint may not always be suitable for the application if publication regions of different sizes are needed. We study the effects that grid cells have on updated messages by constraining publication regions to an integral numbers of grid cells. We see that as we increase N the grid cell size become smaller resulting in less filtering at the destination to reduce irrelevant messages; but this comes at the cost of an increased number of duplicate messages. We also conclude that overall grid cell size with no constraints causes an increase in energy as grid cell size increases for grid based approaches.

5 CONCLUSION

The goal of this work was to explore and understand the energy constraining components of using real time data to drive on line distributed traffic simulations. Our research provides some light into the energy and power consuming components of such a system, an area that has not been extensively explored in prior research. The major components evaluated include the embedded traffic simulation itself, different priority queue implementations used for discrete event simulations, real time data communication, and the communication architecture used for communicating data within the distributed simulation.

We conducted a study on the implications that the simulation model plays on energy consumption for a specific transportation network. Our measurements indicated that a cellular automata model had higher energy consumption than queueing network based discrete simulation model. This was attributed to the fact that the cellular automata must evaluate every vehicle within the system during every time step, whereas the discrete simulation only evaluates vehicles that are at the front of each queue during each executed event. While these results are specific to a particular implementation operating on one hardware configuration, these results may help guide other explorations of energy consumption in traffic simulation models.

Many discrete event simulations rely on an efficient future event list implementation. A comparison of four different priority queue implementations, - a linked list, implicit heap, explicit heap, and splay tree - revealed that the memory system plays an important role on energy consumption. The power, energy, and time could be estimated by measuring the number of comparisons and swaps required by each implementation. The implicit heap consumed the least amount of energy and power among these four implementations, in part, due to good reference locality that appeared to result in improved cache performance.

Communicating real time data plays a vital role in on line traffic simulations. We assessed the consequences of different strategies for sending data. Our measurements illustrated the benefits of data aggregation on energy consumption.

The High Level Architecture data distribution management services provide methods that aim to efficiently distribute simulation data. Our investigations revealed that when comparing purely region based approaches to grid based approaches, the initial computation needed in region base approaches incurs a significant energy cost of whereas irrelevant and duplicate messages in grid based approaches represent a significant energy overhead. A distributed version of the region based approach helps to reduce the amount of computation required and increases scalability. The grid based approaches perform well in determining initial multicast group assignment. The irrelevant message problem is alleviated in the grid filtered region based approach by using a threshold parameter to compare subscribing and publishing federates. From an energy perspective, the fixed grid approach was most efficient for initial determination of group membership and the region based or distributed region based approach is most efficient for communications. Placing constraints on regions yielded less energy consumption. Placing constraints on grid cells could allow one to achieve communication efficiency close to a region based approach when this technique can be applied.

REFERENCES

[1] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *Computational Science-ICCS 2004*, ed: Springer, 2004, pp. 662-669.

[2] G. Allen, "Building a dynamic data driven application system for hurricane forecasting," *Computational Science–ICCS 2007*, pp. 1034-1041, 2007.

[3] M. Gaynor, M. Seltzer, S. Moulton, and J. Freedman, "A dynamic, data-driven, decision support system for emergency medical services," *Computational Science–ICCS 2005*, pp. 61-100, 2005.

[4] J. Mandel, L. Bennethum, M. Chen, J. Coen, C. Douglas, L. Franca, *et al.*, "Towards a dynamic data driven application system for wildfire simulation," *Computational Science–ICCS 2005*, pp. 197-227, 2005.

[5] K. L. Morse, *Interest management in large-scale distributed simulations*: Information and Computer Science, University of California, Irvine, 1996.

[6] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis* vol. 2: McGraw-Hill New York, 1991.

[7] I. S. Association, "1516–2010-IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA)," ed, 2012.

[8] D. Srinivasan, M. C. Choy, and R. L. Cheu, "Neural networks for real-time traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, pp. 261-272, 2006.

[9] D. Helbing, A. Hennecke, V. Shvetsov, and M. Treiber, "MASTER: macroscopic traffic simulation based on a gas-kinetic, non-local traffic model," *Transportation Research Part B: Methodological*, vol. 35, pp. 183-211, 2001.

[10] V. Adamo, V. Astarita, M. Florian, M. Mahut, and J. Wu, "Modelling the spill-back of congestion in link based dynamic network loading models: a simulation model with application," in *14th International Symposium on Transportation and Traffic Theory*, 1999.

[11] S. A. Boxill and L. Yu, "An evaluation of traffic simulation models for supporting its," *Houston, TX: Development Centre for Transportation Training and Research, Texas Southern University,* 2000.

[12] A. Adamatzky, *Game of life cellular automata* vol. 1: Springer, 2010.

[13] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal de physique I*, vol. 2, pp. 2221-2229, 1992.

[14] J. Esser and M. Schreckenberg, "Microscopic simulation of urban traffic based on cellular automata," *International Journal of Modern Physics C*, vol. 8, pp. 1025-1036, 1997.

[15] M. Hasan, M. Jha, and M. Ben-Akiva, "Evaluation of ramp control algorithms using microscopic traffic simulation," *Transportation Research Part C: Emerging Technologies,* vol. 10, pp. 229-256, 2002.

[16] M. Jha, K. Moore, and B. Pashaie, "Emergency evacuation planning with microscopic traffic simulation," *Transportation Research Record: Journal of the Transportation Research Board*, pp. 40-48, 2004.

[17] Q. Yang and H. N. Koutsopoulos, "A microscopic traffic simulator for evaluation of dynamic traffic management systems," *Transportation Research Part C: Emerging Technologies*, vol. 4, pp. 113-129, 1996.

[18] J. Barceló, *Fundamentals of traffic simulation* vol. 145: Springer, 2010.

[19] W. Burghout, H. N. Koutsopoulos, and I. Andreasson, "A discrete-event mesoscopic traffic simulation model for hybrid traffic simulation," in *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE,* 2006, pp. 1102-1107.

[20] A. S. Malik, H. M. Tawfik, and M. Adeel, "SmartRNA: A Road Network Analysis Simulator," in *Engineering Sciences and Technology, 2005. SCONEST 2005. Student Conference on*, 2005, pp. 1-6.

[21] T. Balasha and T. Toledo, "MESCOP: A Mesoscopic Traffic Simulation Model to Evaluate and Optimize Signal Control Plans," *Transportation Research Record: Journal of the Transportation Research Board,* pp. 1-9, 2015.

[22] K. M. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs," *IEEE Transactions on software engineering*, pp. 440-452, 1979.

[23] D. Jefferson and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Method, Part I: Local Control," The Rand Corporation, Santa Monica, California USADecember 1982.

[24] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, pp. 404-425, 1985.

[25] G. Pardo-Castellote, "OMG Data-Distribution Service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, 2003, pp. 200-206.

[26] R. Joshi and G.-P. Castellote, "A comparison and mapping of data distribution service and high-level architecture," *Technology, The Netherlands. His research interests include parallel and distributed computing, component based architectures, and embedded systems*, 2006.

[27] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating computer simulation systems: an introduction to the high level architecture*: Prentice Hall PTR, 1999.

[28] D. RTI, "1.3-Next Generation Programmer's Guide Version 5," *DoD, DMSO*, 2002.

[29] J. S. Dahmann and K. L. Morse, "High Level Architecture for simulation: an update," in *Proceedings. 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications (Cat. No.98EX191)*, 1998, pp. 32-40.

[30] G. Tan, Y. Zhang, and R. Ayani, "A hybrid approach to data distribution management," in *Distributed Simulation and Real-Time Applications, 2000.(DS-RT 2000). Proceedings. Fourth IEEE International Workshop on*, 2000, pp. 55-61.

[31] A. Boukerche and A. Roy, "Dynamic grid-based approach to data distribution management," *Journal of Parallel and Distributed Computing*, vol. 62, pp. 366-392, 2002.

[32] A. Boukerche, N. J. McGraw, C. Dzermajko, and K. Lu, "Grid-filtered region-based data distribution management in large-scale distributed simulation systems," in *Simulation Symposium, 2005. Proceedings. 38th Annual*, 2005, pp. 259-266.

[33] S. J. Rak and D. J. Van Hook, "Evaluation of grid-based relevance filtering for multicast group assignment," in *Proc. of 14th DIS workshop,* 1996, pp. 739-747.

[34] C. Raczy, J. Yu, G. Tan, S. Tay, and R. Ayani, "Adaptive data distribution management for HLA RTI," in *Proceedings of 2002 European Simulation Interoperability Workshop*, 2002.

[35] C. Raczy, G. Tan, and J. Yu, "A sort-based DDM matching algorithm for HLA," *ACM Trans. Model. Comput. Simul.,* vol. 15, pp. 14-38, 2005.

[36] K. Pan, S. J. Turner, W. Cai, and Z. Li, "An Efficient Sort-Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment," presented at the Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, 2007.

[37] K. Pan, S. J. Turner, W. Cai, and Z. Li, "A dynamic sort-based DDM matching algorithm for HLA applications," *ACM Trans. Model. Comput. Simul.,* vol. 21, pp. 1-17, 2011.

[38] J. Song, B. Xiang, X. Wang, L. Wu, and C. Chang, "Application of dynamic data driven application system in environmental science," *Environmental Reviews,* vol. 22, pp. 287-297, 2014.

[39] J. Wahle, L. Neubert, J. Esser, and M. Schreckenberg, "A cellular automaton traffic flow model for online simulation of traffic," *Parallel Computing,* vol. 27, pp. 719-735, 2001.

[40] R. Rodríguez, A. Cortés, and T. Margalef, "Injecting dynamic real-time data into a DDDAS for forest fire behavior prediction," in *Computational Science–ICCS 2009,* ed: Springer, 2009, pp. 489-499.

[41] I. D. Schizas and V. Maroulas, "Dynamic Data Driven Sensor Network Selection and Tracking," *Procedia Computer Science,* vol. 51, pp. 2583-2592, 2015.

[42] N. Patrikalakis, J. McCarthy, A. Robinson, H. Schmidt, C. Evangelinos, P. Haley, *et al.*, "Towards a dynamic data driven system for rapid adaptive interdisciplinary ocean forecasting," *Dynamic Data-Driven Application Systems. Kluwer Academic Publishers, Amsterdam,* 2004.

[43] R. Fujimoto, R. Guensler, M. Hunter, H.-K. Kim, J. Lee, J. Leonard, *et al.*, "Dynamic data driven application simulation of surface transportation systems," *Computational Science–ICCS 2006,* pp. 425-432, 2006.

[44] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, 1994, p. 2.

[45] D. C. Snowdon12, E. Le Sueur, S. M. Petters, and G. Heiser, "A platform for os-level power management," *The European Professional Society on Computer Systems 2009,* 2009.

[46] S. Singh, M. Woo, and C. S. Raghavendra, "Power-aware routing in mobile ad hoc networks," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, 1998, pp. 181-190.

[47] F. De Rango, M. Fotino, and S. Marano, "EE-OLSR: energy efficient OLSR routing protocol for mobile ad-hoc networks," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, 2008, pp. 1-7.

[48] J.-C. Cano and P. Manzoni, "A performance comparison of energy consumption for mobile ad hoc network routing protocols," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on*, 2000, pp. 57-64.

[49] R. Mayo and P. Ranganathan, "Energy consumption in mobile devices: why future systems need requirements–aware energy scale-down," *Power-Aware Computer Systems*, pp. 301-463, 2005.

[50] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," 2010.

[51] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, pp. 129-140, 2013.

[52] R. N. Mayo and P. Ranganathan, "Energy consumption in mobile devices: why future systems need requirements–aware energy scale-down," in *International Workshop on Power-Aware Computer Systems*, 2003, pp. 26-40.

[53] J. K. Nurminen and J. Noyranen, "Energy-Consumption in Mobile Peer-to-Peer - Quantitative Results from File Sharing," in *2008 5th IEEE Consumer Communications and Networking Conference*, 2008, pp. 729-733.

[54] L. M. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, 2001, pp. 1548-1557 vol.3.

[55] T. Pering, Y. Agarwal, R. Gupta, and R. Want, "<i>CoolSpots</i>: reducing the power consumption of wireless mobile devices with multiple radio interfaces," presented at the Proceedings of the 4th international conference on Mobile systems, applications and services, Uppsala, Sweden, 2006.

[56] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009, pp. 280-293.

[57] T. Profiler, "Qualcomm," ed.

[58] A. Biswas and R. Fujimoto, "Profiling energy consumption in distributed simulations," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, 2016, pp. 201-209.

[59] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, *et al.*, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*, 2010, pp. 105-114.

[60] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012, pp. 317-328.

[61] J. Dongarra, H. Ltaief, P. Luszczek, and V. M. Weaver, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architectures," in *Cloud and Green Computing (CGC), 2012 Second International Conference on*, 2012, pp. 274-281.

[62] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 153-168.

[63] S. Neal, R. Fujimoto, and M. Hunter, "Energy consumption of Data Driven traffic simulations," in *2016 Winter Simulation Conference (WSC)*, 2016, pp. 1119-1130.

[64] S.Neal and R.Fujiimoto, "Power Consumption of Future Event List Implementations in Discrete Event Simulation," in *2018 Spring Simulation-ANNS*, Baltimore, MD, 2018.

[65] S. A. Neal and R. M. Fujimoto, "Energy consumption of HLA data distribution management approaches," in *2017 Winter Simulation Conference (WSC)*, 2017, pp. 810-820.

[66] R. Fujimoto, M. Hunter, J. Sirichoke, M. Palekar, H. Kim, and W. Suh, "Ad hoc distributed simulations," in *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, 2007, pp. 15-24.

[67] U. FHWA, "Department of Transportation: ngsim: next generation simulation," ed, 2007.

[68] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour, "Two lane traffic simulations using cellular automata," *Physica A: Statistical Mechanics and its Applications*, vol. 231, pp. 534-550, 1996.

[69] F. Darema, "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements," presented at the International Conference on Computational Science, Kraków, Poland, 2004.

[70] N. Leavitt, *Big Iron Moves Toward Exascale Computing* vol. 45, 2012.

[71] S. Huck, "Measuring Processor Power," Intel Corporation2011.

[72] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, *et al.*, "United States Data Center Energy Usage Report," Lawrence Berkeley National Laboratory LBNL-1005775, June 2016.

[73] W. M. McCormack and R. G. Sargent, "Analysis of future event set algorithms for discrete event simulation," *Commun. ACM,* vol. 24, pp. 801-812, 1981.

[74] J. H. Kingston, "Analysis of tree algorithms for the simulation event list," *Acta Informatica,* vol. 22, pp. 15-33, April 01 1985.

[75] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary trees," presented at the Proceedings of the fifteenth annual ACM symposium on Theory of computing, 1983.

[76] D. W. Jones, "An empirical comparison of priority-queue and event-set implementations," *Communications of the ACM,* vol. 29, pp. 300-311, 1986.

[77] H. Chen, J. Wang, and L. Feng, "Research on the Dynamic Data-driven Application System Architecture for Flight Delay Prediction," *JSW,* vol. 7, pp. 263-268, 2012.

[78] Q. Long, "A framework for data-driven computational experiments of inter-organizational collaborations in supply chain networks," *Information Sciences,* vol. 399, pp. 43-63, 2017.

[79] J. Danjel and J. O. C. Van Hook, "Data distribution management in RTI 1.3," in *Proceedings of the Simulation Interoperability Workshop (SIW)*, 2004.

[80] G. Tan, R. Ayani, Y. Zhang, and F. Moradi, "Grid-based data management in distributed simulation," in *Simulation Symposium, 2000.(SS 2000) Proceedings. 33rd Annual*, 2000, pp. 7-13.

[81] D. Bedard, R. Fowler, M. Linn, and A. Porterfield, "PowerMon 2: Fine-grained, integrated power measurement," *Renaissance Computing Institute, Tech. Rep. TR-09-04,* 2009.