Maximizing Real-Time Distribution of Wind-Electricity to Electrical Thermal Storage Units for Residential Space Heating

by

Andrew Howard Barnes

Submitted in partial fulfilment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
August 2011

DALHOUSIE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "Maximizing Real-Time Distribution of Wind-Electricity to Electrical Thermal Storage Units for Residential Space Heating" by Andrew Howard Barnes in partial fulfilment of the requirements for the degree of Master of Applied Science.

Dated:	August 23, 2011
Supervisor:	_________________________________
Readers:	_________________________________
	_________________________________

Table of Contents

List of Tables

List of Figures

Abstract

Wind-electricity is unpredictable in both intensity and duration. This thesis presents the design and implementation of Client-pull and Server-push architectures for the distribution of wind-electricity to Electrical Thermal Storage (ETS) units to match the electrical load of the ETS units with the electricity generation levels.

Wind as an energy source is reviewed and the smart grid concept of a communication layer for the transmission, production and usage of electricity is explored. ETS operation is explained and a survey of the Client-pull and Server-push concepts.

These implementations are evaluated on their ability to dispatch wind-electricity over a full heating season, short term latency, single day performance and complexity.

Client-pull and Server-push architectures have nearly identical performance over a full heating season and identical performance over the 24 hour period evaluated. The Server-push architecture has lower short-term latency but is more complex than the Client-pull.

Glossary

Anthropogenic: Caused or produced by humans.

API: Application Programming Interface. A software interface which allows two applications to exchange information using a predefined set of functions and specifications.

GET: An HTTP method used to request information from the server without changing the server state.

IDE: Integrated Development Environment. A an application designed for software development which includes a source code editor, compiler, debugger and tools.

MyQSL: An open source relational database management system.
POST: An HTTP method that passes data to the server for processing. This may result in the creation or updating of an existing resource. The data is included in the body of the request.

PUT: An HTTP method that uploads a representation of a resource.

SQL: Structured Query Language. A database computer declarative language designed for managing data in relational database management systems.

UDP: User Datagram Protocol. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. 3

checksum: A fixed-size datum computed from an arbitrary block of digital data for the purpose of detecting accidental errors that may have been introduced during its transmission or storage. The integrity of the data can be checked at any later time by recalculating the checksum and comparing it with the stored one. If the checksums match, the data were almost certainly not altered.

Client-pull: a style of network communication where the initial request for data originates from the client, and then is responded to by the server.

database table: a set of data elements (values) that is organized using a model of vertical columns (which are identified by their name) and horizontal rows. A table has a specified number of columns, but can have any number of rows.

datagram: A basic transfer unit associated with a packet-switched network in which the delivery arrival time and order are not guaranteed. A datagram consists of header and data areas, where the header contains information sufficient for routing from the originating equipment to the destination without relying on prior exchanges between the equipment and the network. The source and destination addresses as well as a type field are found in the header of a datagram.

device id: A unique character string consisting of a series of letters and numbers used to identify a device within the system.

ETS: Electric Thermal Storage. A device which converts electrical energy into thermal energy and stores the thermal energy in ceramic bricks for later use for space heating.

multicast: The delivery of a message or information to a group of destination computers simultaneously in a single transmission from the source creating copies automatically in other network elements.

password: A sequence of characters and numbers that is assigned by a system administrator that is used in conjunction with a device id to verify the identity of the device communicating with the system.

RESTful: A web service that is stateless and uses the HTTP POST method to create a resource, GET to retrieve a resource, PUT to update a resource, and DELETE to delete a resource. It also exposes directory-like URIs to server resources or web pages.

unicast: the sending of messages to a single network destination identified by a unique address.

Virtual Private Network: A mechanism for providing secure, reliable transport over Internet. The VPN uses authentication to deny access to unauthorized users, and encryption to prevent unauthorized users from reading the private network packets.

Acknowledgements

I would like to express my gratitude to Dr. Larry Hughes for sharing his knowledge, encouragement, suggestions with me during the preparation of this thesis. Maintaining a full-time job and family while completing this work proved to be a greater challenge then I had expected.

My professional work experience prepared me for the technical aspects of my research, but Dr. Hughes experience in thesis preparation and formal documentation have been invaluable to me. Finishing this work would not have been possible without Dr. Hughes.

Before this work started I was not keenly aware of energy issues including peak oil, energy security, renewable power integration, or the smart grid and how these issues will affect our society during the next twenty years. These issues and many more have been explored under Prof. Hughes leadership in the Energy Research Group. The contribution that this group has made to my work should also be acknowledged. I am privileged to have been a member and hope to continue to be a part of that community after my research has finished.

Finally, I would like to thank my wife, Nancy Phalen. She has been a constant supporter through good times and bad. Her encouragement and understanding have allowed me to complete my studies.

Chapter 1 Introduction

The world is entering a period of climate change due to increased concentrations of CO_2 in the atmosphere from anthropogenic sources. Significant changes in surface temperature, rainfall, and sea level have been determined to be largely irreversible for more than 1000 years even if CO_2 emission were to stop [1]. Further increases in CO_2 levels beyond the current level of nearly 385 parts per million by volume are expected to increase sea level and dry season rainfall reductions in some areas over the next millennium [1]. CO_2 emissions are also affecting our oceans; some emissions into the atmosphere are absorbed by the oceans and combine with water (H_2O) to form carbonic acid (H_2CO_3). This affects the calcium carbonate saturation states, which impacts shell-forming marine organisms from plankton to benthic mollusks, echinoderms, and corals [2].

Energy security, the access to available and affordable energy sources, is also becoming a recognized issue for the three basic energy services: transportation, heating and cooling, and on-demand electricity. The energy needs of these services are largely met by oil, coal, natural gas or biomass, all of which are sources of CO_2 emissions and have recently been subject to price volatility. The situation is summed up in the executive summary of the World Energy Outlook 2008, "The world's energy system is at a crossroads. Current global trends in energy supply and consumption are patently unsustainable — environmentally, economically, socially. But that can — and must — be altered; there's still time to change the road we're on. It is not an exaggeration to claim that the future of human prosperity depends on how successfully we tackle the two central energy challenges facing us today: securing the supply of reliable and affordable energy; and effecting a rapid transformation to a low-carbon, efficient and environmentally benign system of energy supply. What is needed is nothing short of an energy revolution…." [3].

Mitigating climate change while maintaining energy security presents a significant challenge to how we produce and use energy. During the UN conference on climate change in Copenhagen in 2009, the Copenhagen Accord set non-binding objectives to limit the increase in global temperature to two degrees Celsius (2°C) above pre-industrial levels. Commitments made by members of the accord, even if fully implemented, are expected to be insufficient to meet the target of a 2°C rise in global temperature [4].

The global demand for electricity is expected to grow at a rate of 2.2% per year between 2008 and 2035 [4]. This increase is expected to be met with increasing use of both renewable and non-renewable energy sources. Under the New Policies Scenario of the Copenhagen Accord, renewable energy sources are expected to increase their share of global electricity production from 19% in 2008 to nearly 33% by 2035, with the increase coming largely from wind and hydro electricity [4]. To meet the 2°C rise in global temperature, the share of electricity production by renewable technologies would have to increase to 45% of global electricity production by 2035 [4].

Many jurisdictions are increasing their supply of renewable energy sources, most notably wind, in an effort to increase energy security and reduce green house gas emissions. Wind generated energy is renewable, environmentally friendly, and has a low-carbon footprint[5]. These factors contribute to making wind the fastest growing form of energy worldwide [6]. Despite its growth, one of wind's major drawbacks is intermittency [7]. At times electricity from a network of wind turbines located in the same geographical area (often referred to as a wind farm) may be produced in abundance, possibly exceeding demand. When this occurs, the excess wind electricity must be stored for later use, sold to another district, or left unused. On the other hand, when insufficient wind electricity is available, the shortfall needs to be met from rapid response generation equipment such as gas turbines or local hydro power, or purchased from other jurisdictions. Under ideal circumstances, the wind generated electricity would match the energy demand. These facts make wind not suitable for on-demand electricity usage without a mechanism to manage its intermittency. A more efficient way to use wind generated electricity is to apply it to another basic energy service that can take advantage of its intermittent nature to and reduce its CO_2 footprint for that service and increase energy security.

1.1 Energy Services

Energy usage can be divided into three services; on-demand electricity, heating and cooling, and transportation.

On-demand electricity is available the instant the service is needed. Examples of on-demand energy usage would be turning on a light switch, a TV, or an electric oven. Any delay in providing the electricity for on-demand use would be considered unacceptable. This energy service is always present regardless of the time of year.

The heating and cooling service is seasonal with the mix of energy used for heating or cooling depending on latitude. In northern latitudes, the heating service is dominant but these changes to cooling at latitudes near the equator. During periods of extreme cold or heat, the energy used for heating or cooling can place significant demand peaks on the electricity provider. The electricity provider must have sufficient generation capacity to meet these extreme energy demands or risk blackouts.

Transportation services include the movement of people and goods by airplanes, trains, ships, and automobiles. These forms of transportation typically use energy in the form of electricity or refined petroleum. Air transportation uses refined petroleum products in the form of jet fuel. Many trains use diesel fuel, but the use of electric trains is increasing. Subway systems typically use electricity for propulsion. The vast majority of automobiles use refined petroleum in the form of either gas or diesel as a fuel source. The use of electric cars is increasing and is seen as the next logical step in reducing the carbon dioxide produced by automobiles. Electric vehicles are also seen as an answer to the expected increase in price of gasoline as oil reserves decline.

1.2 Electrical Thermal Storage

Electrical Thermal Storage units (ETS) provide a mechanism to convert off-peak electrical energy into thermal energy for later use [8]. These units typically recharge during the overnight hours (such as 11pm to 7am) and use ceramic bricks which are heated up to about 800°C by electrical heating elements to store thermal energy [9]. The stored thermal energy is recovered for space heating by circulating air through the ceramic bricks then distributed to the house through a heating system designed to heat an entire home. Smaller units can be put directly into a room to heat a single area. Dual-purpose units are also available which provide domestic hot water in addition to space heating [10].

This is advantageous for electricity providers because it shifts the electrical demand for space heating from the daytime hours and early evening hours, when electrical demand is typically high to the overnight hours when demand is low. During the evening hours the electrical provider may be producing electricity in abundance. Some forms of electricity production such as coal-fired generation, or biomass cannot be turned on or off quickly and typically run at maximum capacity for maximum efficiency. The energy produced from these sources during the evening hours and kept for spinning reserve (energy that is kept in reserve to meet unexpected sudden demands in electricity usage) may not be used if demand is low. As an incentive, producer's offer customers a reduced off-peak price for the electricity used during periods of low demand, reducing the customers cost for space heating [11] .

Although this system does provide a mechanism for reducing the peak load seen by producers during the day-time hours and does reduce the customers cost for space heating, it is not the only possible application of ETS units. An ETS unit can be turned on at any point during the day if there is electricity available that requires storage and the ETS unit has unused storage capacity.

1.2.1 ETS Units for Wind Heating

ETS units are a natural storage medium for storing electrical energy for later use within a residence. Using ETS units to store energy generated from an intermittent supplier has been explored by Hughes [12]. This solution to mitigating the intermittence of wind energy balances the number of ETS units that are recharging to match the amount of intermittent energy being generated. As the generation levels change, ETS systems are either increase or decrease the collective load of the ETS systems to match the generation levels. To match generation and consumption levels, this system requires electrical generation information from the energy producers, and electrical consumption information from the distributed ETS units. The number of units available to store intermittent wind energy will also be needed. This information will need to be collected and processed by a control system that can communicate with the intermittent energy producers and the ETS systems to match generation and consumption levels.

The technology required to implement a control system for the distribution of intermittent energy to distributed storage devices such as ETS units exists today but is used to meet different consumer demands. Broadband internet connections allow for the distribution of radio and video to multiple customers simultaneously. The online gaming industry uses broadband internet for real-time multiplayer virtual environments allowing hundreds of players to interact in real time. Data exchange using broadband internet connections is now common place through the use of the World Wide Web. Clearly, there are many options for implementing a control system using existing technologies. The Steffes Corporation has done preliminary work using ETS units with a broadband control system as a means for load balancing with limited success [8]. It would appear that it may be possible to develop systems capable of utilizing intermittent electricity and provide other grid stability control options for power distributors.

1.3 Energy Flows, Processes, and Chains

Maximizing the use of domestically generated intermittent wind energy is an exercise in increasing jurisdiction energy security. The modeling of jurisdiction energy security has been explored by the use of energy flows, process and chains into an energy security framework [13]. Modeling the use of intermittent wind energy from production to end-use for space heating using this framework using flows, processes and chains will allow the development of the communication architectures necessary for energy distribution using a logical and systematic approach.

1.3.1 Flow Diagrams

Flow diagrams were initially used to show the flow of data in a system. In the context of an energy security model it is used to show the flow of energy within a jurisdiction. The generation, distribution, usage and storage of energy all provide paths for the flow of energy within a flow diagram.

Energy inputs are represented as rectangles to define energy inputs of a system. Arrows are used to show the direction of energy, or its flow. The actions that take place to either change the state of energy on its way to the destination, or change the levels of energy flowing from the source to the destination are defined as processes. In the context diagram processes are represented as circles labeled with the name of the process performing the energy state change. Energy stores are represented as two horizontal lines with the name of the store between the lines.

For demonstration purposes, Figure 1 shows a flow diagram for the conversion of corn to ethanol.

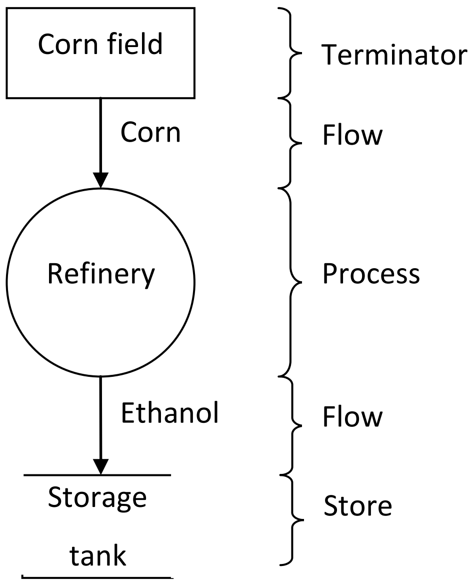


Figure 1 Flow Diagram for Corn to Ethanol

Two types of flow diagrams can be used in an energy security model, context diagrams which provide an overall picture of the energy distribution flow paths, and a behavioral diagram which defines the energy states and rules for moving between states.

1.3.2 Context Diagram

In the energy security framework, context diagrams are used to define a jurisdictions energy system. In the model used in this work, energy is supplied from four different sources: dispatchable nonrenewable sources such as coal fired generation, dispatchable renewable sources such as hydro electricity, non-dispatchable renewable sources such as wind, and grid intertie where energy is imported through the electrical grid from an adjacent jurisdiction. End uses of energy are also represented by terminators. In the model used in this work, there are four different end uses of energy in the form of electricity; on-demand electricity such as lighting, or electrical appliance use, on-demand space heating such as baseboard heating, storage such as ETS systems, and grid intertie where electrical energy is exported to an adjacent jurisdiction.

Energy flows from sources to destinations**.** Figure 2 shows the behavioral model for an arbitrary jurisdiction use of electricity. Since this work focuses on the use of ETS systems, they have been separated from on-demand energy usage as a separate end use.

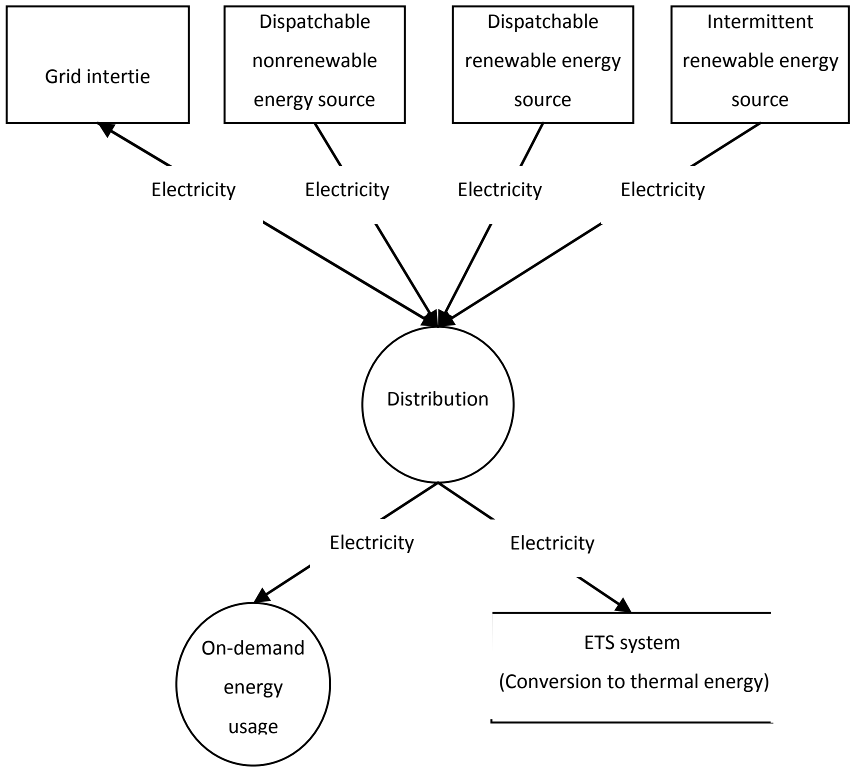

Grid intertie
Dispatchable nonrenewable energy source
Dispatchable renewable energy source
Intermittent renewable energy source
Electricity
Electricity
Electricity
Electricity
Distribution
Electricity
Electricity
On-demand energy usage
ETS system
(Conversion to thermal energy)

Figure 2 Space Heating Energy Flow Diagram

1.3.3 Behavioral Model

One of the factors that dictate the flow of energy in this model is the time of day. Traditional ETS evening recharge places an additional load during the evening when the ETS systems are recharging. The energy required to meet the demand is met by a set of the producers which supply energy to the grid. The rules which are used to determine which sources are used and at what level may vary from jurisdiction to jurisdiction and the output capacity of the individual suppliers. This work focuses on the distribution of wind energy for space heating by redirecting the energy to ETS units when wind energy is generated above the level required to meet the on-demand energy needs of a jurisdiction. As a result, the distribution rules will focus on monitoring the on-demand energy requirements and the generation levels of the energy producers.

The energy security model framework uses state diagrams for the rules associated with meeting a jurisdictions energy requirement. This energy security model uses three states to represent the jurisdiction.

Neutral: In this state the jurisdiction can meet its energy requirements through its own domestic supply. Demand and generation levels can be adjusted to form a balance that can be maintained through dispatchable loads and storage. This represents an ideal energy security state.

Exporting Energy: In this state energy is being produced within the jurisdiction beyond the demand and storage levels of the jurisdiction. In this state energy is being exported to an adjacent jurisdiction. This is a less desirable state because domestically generated energy is not being used within the jurisdiction.

Importing Energy: This state exists when a jurisdiction is incapable of meeting its energy requirements and must import energy to satisfy the demand. This is the least desirable state.

The connectors between these states represent the rules associated with the energy security framework. In the context of using ETS units to increase regional energy security the rules can be summarized as follows, if there is a surplus of wind generated energy after the on-demand load has been satisfied, a sub-set of the ETS systems will be turned to create a load equal to the surplus. If there is still a surplus after the recharging requirements of the ETS systems have been met, the additional power will be distributed to the grid intertie for use by an adjacent jurisdiction. If the on demand electricity requirements cannot be met by the domestic sources of energy (both renewable and non-renewable) the shortfall will be met by energy from an adjacent jurisdiction through the grid intertie.

1.4 Control Systems

A control system can follow one of two basic implementation strategies, client-pull, or server-push. In client-pull, clients initiate the communication with the server and request instructions from the server. This model is used for web browsers which connect to a web server through an internet connection and send a command to the server requesting information from the server, or provider information to the server. The protocol used by the World Wide Web is the Hypertext Transfer Protocol (HTTP) which is defined by the Internet Engineering Taskforce through the publication of a Request For Comments (RFCs) document; the most recent definition of the HTTP protocol is found in RFC 2616 [14] .

In server-push, the server initiates communication with the clients and sends commands to the clients as a group, or serially. This is used for distributing video and radio over internet connections. A common server-push technology used to send data to a number of clients simultaneously is through the use of User Datagram Protocol (UDP). This protocol is also defined by the Internet Engineering Taskforce with the most recent definition of the UDP protocol found in RFC 768 [15]. Distributing energy from intermittent sources can be accomplished using a control system designed using client-pull, server-push, or a combination of both. The concept of using a data communication layer to the electrical system is known as part of the Smart Grid.

Smart Grid builds on many of the technologies already used by electric utilities but adds communication and control capabilities that will optimize the operation of the entire electrical grid. Smart Grid is also positioned to take advantage of new technologies, such as plug-in hybrid electric vehicles, various forms of distributed generation, solar energy, smart metering, lighting management systems, distribution automation, and many more[16].

1.5 Objectives

The objective of this work is to design and test two control systems, one based upon the client-pull topology and other based upon a server push topology for the optimal distribution of intermittent wind energy to thermal storage units. These control systems will be compared on their ability to utilize wind-electricity over a full heating season, on their short term latency when the available wind generated electricity is increased, performance over a single day, and complexity.

1.6 Thesis Contents

The remainder of this thesis is organized as follows; chapter 2 presents a detailed examination of ETS units, the Smart Grid concept, and wind as an intermittent energy source. Chapter 3 presents frameworks for two methods for distributing energy from intermittent energy sources using the Smart Grid concept. Chapter 4 contains detailed sample implementations used for testing both the client pull and server push frameworks. Chapter 5 contains the results of testing both approaches showing the strengths and weakness. Chapter 6 summarizes the work and makes recommendations for further research in this area.

Chapter 2 Background

This chapter examines wind as an energy source, energy storage using ETS devices and the emerging Smart Grid concept as a means to control the distribution and consumption of electricity. Finally, different methods of implementing a control system using existing technologies are explored.

2.1 Wind as an Energy Source

Wind turbines convert the kinetic energy of the wind into electrical energy providing a renewable, fuel-free source of energy that generates no greenhouse gasses. World generation capacity for wind power reached 159 GW in 2009 with a growth rate of 31.7% [17]. The total potential for wind power on land and from near-shore has been estimated at 72 TW, but practical barriers exist making this upper limit unobtainable [18]; however, the estimated global wind production could reach 1.9 TW by 2020 [17]. Wind is expected to become a replacement energy source for other fossil based fuels as reserves decline in years to come [19].

Although these statistics are impressive, wind does present challenges as an energy source. Wind is difficult to forecast and can vary from second to second. The variability of wind speed varies the energy output. Wind turbines are rated by their maximum output capacity which is the maximum energy that the wind turbine can provide under optimal conditions. These optimum conditions occur rarely, making generation capacity not a valid indicator of actual energy production. The difference between the actual energy produced by a wind turbine and its rated maximum is known as its capacity factor. Reported capacity factors for wind turbines are in the range of 20-51% [20]. The actual capacity factor for a wind turbine is location dependent.

When multiple wind turbines are installed in the same location where wind conditions are favorable for energy production, the turbines are collectively called a wind farm. Because wind farm generated electricity output is variable in both duration and intensity, it is non-dispatchable; that is, it cannot be increased or decreased on demand). When wind electricity is generated, it must be used or the electricity is wasted.

The graph in Figure 3 shows the hourly power generation from the combined wind farms of the Atlantic Wind Test Site at North Cape, PEI and the Summerside, PEI.

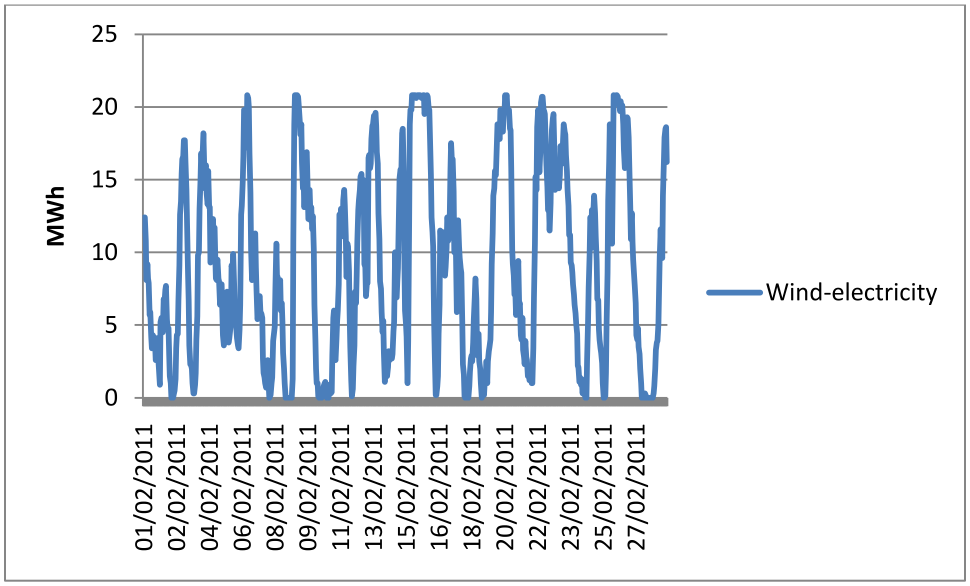


Figure 3: Distribution of Wind-electricity Generation (February, 2011)

This intermittency results in a reduction in efficiency of conventional dispatchable generation units as output of conventional units is reduced as wind power increases, thereby reducing the expected emissions benefits of wind power [21]. Intermittency also requires reserves for regulation at the 1-minute interval, load-following at the 5-minute interval, and operating reserves at the 10-minute interval to handle wind electricity's variability [21].

2.2 Smart Grid

Implementing a system that allows loads to know when energy is available will require communication between the appliance and the energy provider. The concept of adding a communication layer to the transportation, consumption, and generation of electricity has commonly become known as the Smart Grid [22]. The Smart Grid is not an entity that can be tangibly defined. The Smart Grid does not consist of wind turbines, solar panels, or tidal power projects, but rather the technologies that allow the power generated by those and other more traditional energy producers to be integrated into a energy grid that allows the optimum use of intermittent electricity supplies.

The following objectives are taken from the U.S Department of Energy as objectives for the smart grid [23]:

Intelligent: Capable of detecting grid overloads and rerouting power to minimize outages. Capable of responding autonomously when action is required faster than humans can respond.

Efficient: Capable of meeting increasing customer demand without increasing infrastructure.

Accommodating: Accepting energy from any and all forms of production and integrating any and all better ideas and technologies such as energy storage as they are market proven and ready to come online.

Motivating: Enabling end users to decide when and what forms of energy they wish to use based on their personal preferences using two way communications between consumers and producers.

Opportunistic: Creating new opportunities and markets by means of its ability to capitalize on plug-and-play innovation where and wherever appropriate.

Quality Focused: Free of lags, spikes, disturbances, and interruptions.

Resilient: Increasingly resistant to attack and natural disasters as it becomes more decentralized and reinforced with Smart Grid security protocols.

Green: Slowing the advance of global climate change and offering a genuine path toward significant environmental improvement.

The development of the smart grid is seen as key to reducing greenhouse gas emissions, energy independence and lower energy costs by the current United States Administration [24]. What is also being recognized is that the development of the Smart Grid will require standards for implementation for manufacturers, and energy producers, as well as infrastructure upgrades to allow the required two way communication.

The 2007 Energy Independence and Security Act passed in the United States gives "*primary responsibility to coordinate development of a framework that includes protocols and model standards for information management to achieve interoperability of smart grid devices and systems…" to* the National Institute of Standards and Technology (NSIT) [25]. The eight priorities set by the NSIT for smart grid development include [26]:

- Demand Response and Consumer Energy Efficiency
- Wide-Area Situational Awareness
- Energy Storage
- Electric Transportation
- Advanced Metering Infrastructure
- Distribution Grid Management
- Cyber Security
- Network Communications

The NIST Framework and Roadmap for Smart Grid Interoperability Standards identifies 75 existing standards for the ongoing development of the smart grid and identifies 15 high priority gaps and harmonization issues which require new or revised standards [26]. These standards are targeted for completion by early 2011[27]; however, these standards will need to be adopted by manufacturers. For example, the Society of Automotive Engineers has already created standards for 220V links between electric vehicles and the grid, but the Nissan Leaf due later this year uses a proprietary interconnect [28] .

Robert Poor, founder of Blue Dot, a company that offers advice to customer's on ways they can save on their energy bills, believes that the transition will come slowly, "It's at least a decade before this gets to gets into the market in the way it needs to be" [28].

Some initial steps are being taken to move toward a smarter grid. A smarter grid is defined by the US DOE as offering valuable technologies tools and techniques that can be deployed within the very near future or are already deployed today to allow the grid to operate more efficiently with less environmental [23] .

CanmetENERGY believes that the modernization of the electricity networks cannot be simplified to a single technology, but to a collection of applications each requiring precise technologies to operate [29]. Some technologies need to be installed at the customer's home or business while others target the power company's network. What is common in every region is the need to build a communication network or to establish partnerships to manage this quantity of data [29]. Existing or planned Smart Grid initiatives within Canada include the instillation of Smart Meters in Alberta, British Columbia, Manitoba, Ontario [29], the PowerShift Atlantic project in New Brunswick which aims to allow utility companies to better understand how customers will react to smart grid technologies and which electricity loads can be managed better by real time demand [30] and the Wind and Storage demonstration in a First Nations Community [31] to name a few.

2.3 Energy Storage

Electrical energy storage has become a topic of interest to both researchers and electricity providers over the past decade. The United Kingdom Low Carbon Transition Plan [32] includes energy storage in the list of key elements of a UK smart grid. Most existing energy storage implementations use large grid scale facilities to store significant volumes of energy that is produced in access for later return to the electrical grid when needed to provide on demand electricity. One popular method that is currently in use is pumped storage where during periods of low electricity demand, electricity is used to pump water from a lower elevation to a reservoir at a higher elevation creating a height differential. During periods of high demand, the water is allowed to flow back to the lower elevation through turbines and the generated electricity is put back onto the electrical grid. In the United States, there are 40 pumped hydro energy storage stations with a total capacity of nearly 20 GW, with hundreds of more worldwide in operation with a total capacity of 127 GW [33]. Another method that is proposed for large scale electrical storage in our energy future is the use of compressed air energy storage (CAES). CAES is seen as one of the technologies with the highest economic feasibility for an electrical system with better utilization of fluctuating renewable energy sources [34]. CAES is similar to basic gas turbine (GT) technology, but low-cost electricity is used to compress air into and underground cavern. During periods of peak demand, the compressed air is heated and expanded in a gas turbine, producing electricity that is then put back into the electrical grid. Both of the methods described above provide grid scale storage from a centralized location. Distributed storage provides an alternative that uses many smaller scale storage implementations where the stored energy is closer to the end use location. Plug-in electric vehicles (PEVs) are seen as having significant potential in the future grid as a form of distributed storage [35], but again, the target for this proposed implementation is in supplementing the need for on-demand electricity. Any large scale use of PEVs as a form of distributed storage would require agreements between PEV owners and utility providers and the implementation of a bi-directional communication system, as proposed by the smart grid initiative.

2.4 Space Heating

Electric baseboard heaters are inexpensive, easy to install method for heating a residence. An electric baseboard heater has an electrical heating element inside a metal housing: when the heater is turned on an electric current flows through the heating element. The resistance of the heating turns the electrical energy into thermal energy as the element heats up. The element then transfers the generated heat to the surrounding air through convection, causing the air to rise and new colder air to be pulled in from the bottom of the heater. As the hot air rises, it provides space heating for the room. Baseboard heaters are typically installed under windows, so that the heater's rising warm air will counteract falling cool air from the cold window glass [36]. Because baseboard heaters work on the principal of a resistive load that converts electrical energy into thermal energy at the time of use, baseboard heaters require a constant supply of electricity to supply the heating demand that may occur anytime during the day. The application of wind generated electricity for space heating using electric baseboard heaters has been explored by Hughes and been shown to be unreliable due to the fact that generation cannot be expected to follow demand [37].

Electrical thermal storage units are another form of electric heating, but the energy used for the heating does not require on demand electricity. The ETS units use ceramic bricks which are heated up to 700°C by electrical heating elements to store thermal energy. The stored thermal energy is recovered for space heating by circulating air through the device using an electric fan. A controller is used to regulate the flow of air through the device, transferring the thermal energy to the circulating air which is then circulated through the home to provide space heating.

Two types of ETS units are used for space heating: room and central. A room ETS is designed to provide space heating for a limited number of rooms in a house whereas a central ETS unit is larger and provides space heating for the entire building. Central ETS units can be sub-divided into forced hot air that provide space heating only and hydronic systems that can also provide residential hot water [38]. Each ETS unit has a maximum thermal storage capacity that is set by the manufacturer which will dictate the number of rooms that can be heated by the unit. Most ETS units operate in stand-alone mode requiring little or no operator control at a defined maximum output. The normal discharge time is 16 hours with up to eight hours of charge time. ETS units can discharge heat energy while it is being recharged for the following 16 hour discharge cycle. A common method for ETS usage today is through the use of time-of-day rates and smart meters. Under a time-of-day rate, the power utility charges a different rate for electricity usage depending on when the electricity is used. The smart meter accumulates energy usage based upon when the energy is used and the customer is billed accordingly. The goal of this type of rate plan is to modify consumer energy usage by making periods when the energy demand is low have a lower rate. This lower rate typically occurs during the evening hours and is therefore given a discounted rate, whereas a premium rate is used for the typical peak hours of the day [39]. Under this implementation, the smart meter does not need to know what the ETS unit is doing, it simply records the energy usage when it occurs. The ETS also does not need to report to the smart meter and can simply operate on a predefined charge cycle. This implementation is a method of load shifting, the energy required for space heating is simply taken when demand is low. No effort is made to target specific energy sources to charge the ETS units.

ETS units can however recharge at anytime and allow for an alternate implementation that allows for specific energy producers to be used to recharge the ETS units. The communication scheme envisioned by the development of the Smart Grid can allow for the communication of information from electricity providers and consumers. Coupling intermittent wind generated electricity with ETS units provides a dynamic controllable load that can be scaled to match the level of electricity production. This is a form of distributed storage, but the stored energy is not directed back to the electrical grid for on-demand use, it is used for another of the energy services, namely space heating. The feasibility of wind heating has been explored by Hughes [37] and been shown to have great potential. Implementation of coupling wind heating with ETS units will require the development of a control system that can distribute the wind energy to the ETS units.

2.5 Control Systems

The topic of control systems is broad a covers a wide range of devices and system architectures. A control system may be as simple as the darkness setting on a toaster, or as complicated as the ignition system for the space shuttle. Each of these systems collects data from the environment and modifies the system to reach a desired outcome. Our investigation of control systems architecture will be limited to what is needed for controlling ETS units based upon data reported by one of more wind farms.

This design will be a software based control system. It will require status information from the ETS units within the district and also have to send operating instruction to the ETS units. Wind farm generation data will also be needed to accurately dispatch the generated power. The geographic distance between the ETS units, wind farms, and the control system will require a communication link to transmit the bi-directional data. The hardware required to collect the ETS and wind farm data needed by the control system is assumed to exist and that a mechanism to send the data to the control system is already in place. The format of the data transmission will also need to be established using a protocol will that will provide data integrity, security and authentication to block attacks by rogue systems or devices intended to create grid instability. Finally, a hardware platform will be needed to hold the control system and associated software modules. The hardware platform must ensure scalability, archived data integrity, and adequate processor power to minimize system response time.

2.6 Communication Protocol

The communication protocol chosen for the control system is a fundamental aspect of overall system design. The Open System Interconnection (OSI) model defines a networking framework for implementing protocols in seven layers [40]. Each layer describes how its portion of the communication process should function, and how it interfaces to the layers above, below, and adjacent to it on other systems. The seven layers defined by the OSI model are Application, Presentation, Session, Transportation, Network, Data Link, and Physical layers. The Application layer is the highest layer of the OSI and does not provide services to any other layer. It defines the format and rules used for the transmission of data between applications but how the transmission occurs. The layers beneath the Application layer facilitate the transmission of the data that the application is seeking with no regard to what the data actually is. This work evaluates existing Application Layer protocols and their suitability for implementation in a control system for distributing wind generated electricity to ETS units. The design and implementation of a custom Application Layer protocol is possible, but may be unnecessary if proven, well established existing protocols exist that can be used. Two types of existing Application Layer protocols will be evaluated, Client Pull and Server Push.

2.6.1 Client-pull

In client pull communication, the client device initiates a conversation with the server, and a reply is sent from the server. The client can pass data to the server through the request, allowing the server to process the data and respond appropriately. Well established client pull technologies include TELENT for remote access, HTTP for web page access and S-HTTP which is used for secure web content and transactions.

2.6.1.1 TELNET Implementation

The TELNET specification can be found in RFC854 [41]. Telnet was intended to provide remote access to a computer system from a remote terminal (basic key board and screen with no underlying PC hardware) and no security provisions where provided in the initial standard. It is still used today in a secure form called SSH-2 (Secure SHell) [42] that provides a security layer to the standard TELNET protocol. Using this protocol, a client device could connect to a SSH-2 server and login. Once the connection was established, the client could run an application on the server with command line parameters to pass data to the server. The server side application would then execute, process the client parameters, and return instructions to the client in an ASCII format such as XML through the SSH-2 connection. Once the instructions were received, the client could terminate the connection.

2.6.1.2 Web Implementation

HTTP is used as the Application Layer protocol for web pages. This technology has been proven to be reliable with a significant body of software written to support its implementation. HTTP does not provide a security layer, but a modified version of with security is available called

SHTTP. SHTTP is often used for secure credit card transactions on the web. Newer technologies for delivering content to remote systems use Web services as a client pull. A "Web service" is defined by the W3C (World Wide Web Consortium) as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."[43]

The W3C also states, “We can identify two major classes of Web services, REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of ‘stateless’ operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations.” [44]

2.6.1.3 RESTful Web Services

Roy Fielding defined the Representational State Transfer (REST) as a style of software architecture for distributed hypermedia systems such as the World Wide Web in his 2000 doctoral dissertation [45].

REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of "representations" of "resources". A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.

At any particular time, a client can either be in transition between application states or "at rest". A client in a rest state is able to interact with its user, but creates no load and consumes no per-client storage on the set of servers or on the network.

The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used when the client chooses to initiate a new state transition.

REST was initially described in the context of HTTP, but is not limited to that protocol. RESTful architectures can be based on other Application Layer protocols if they already provide a rich and uniform vocabulary for applications based on the transfer of meaningful representational state. RESTful applications maximize the use of the pre-existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it[46].

A RESTful web service has three defined aspects:

- The URL of the web service (i.e., http://myservice/ets_heating)
- The MIME (Multipurpose Internet Mail Extensions) type of data supported by the web service
- The set of operations supported by the web service using HTTP methods (e.g., POST, GET, PUT, or DELETE).

The HTTP specification provides the following guidelines for the use of the POST request:
“The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

Using a POST request also has security implications for the transmitted data. These implications are explained in section 15.1.3 of the HTTP specification:“Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/disable the sending of Referer and From information.
Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol.

Authors of services which use the HTTP protocol SHOULD NOT use GET based forms for the submission of sensitive data, because this will cause this data to be encoded in the Request-URI. Many existing servers, proxies, and user agents will log the request URI in some place where it might be visible to third parties. Servers can use POST-based form submission instead."[47]:

2.6.1.4 Arbitrary Web Services

Two arbitrary web service protocols have emerged as popular choices by developers. The SOAP (Simple Object Access Protocol) protocols sit on top of the HTTP protocol and extend its functionality and **JSON** (JavaScript Object Notation). Each of these protocols is explained in brief below.

SOAP. The SOAP specification is currently maintained by the XML (Extensible Markup Language) Protocol Working Group of the World Wide Web Consortium but was initially created by Microsoft as an object-access protocol. The official definition of SOAP taken from the most recent SOAP 1.2 specification [48]:

> *SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.*

SOAP has the advantage of allowing two applications to exchange complex data types in a system and software independent format. XML is highly human readable and can be verified using a XSD (XML Schema Definition). This comes at the cost of a XML parser to process the data in the HTTP request. This additional layer will increase the CPU usage along with the increased bandwidth requirements of sending lengthy XML formatted data.

JSON. JSON is a lightweight text-based open standard designed for human-readable data interchange. It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for virtually every programming language.

The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json.

The JSON format is often used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web application, serving as an alternative to XML [49].

2.6.2 Server-push

In a server push technology, the server initiates a conversation with the client. This has the advantage of additional control by the server on the activities of the clients by instructing the clients to perform actions by when the server needs the actions to be taken. One disadvantage of all server push technologies is that the client devices must open a port to receive communication. This open port represents a possible security vulnerability for an external program to attack the client and disrupt its operation. Server push technology can come in three forms, broadcast, multicast, and unicast. Each has its own advantages and disadvantages.

2.6.2.1 Multicast

In multicast, the server sends a command to a subset of the devices within a network. Multicast has already been applied to internet solutions that distribute television using IP technology [50]. Implementing Multicast solutions has many challenges which are identified by the Network Working Group through RFC3170 [51] . An implementation using a multicast solution would require ETS units to register with the system when storage was available within the ETS unit. When the ETS unit was completely charged, it would unregister with the system. By keeping an accurate count of the ETS units that can accept wind generated electricity and the knowledge of how much electricity an ETS system will consume, the available wind power can be distributed. Multicast for turning on ETS units could be accomplished by having each ETS within the system join a VPN (Virtual Private Network) and be assigned and IP address within the VPN.

The server would send a Multicast message where the number of addresses in the multicast address range would match the number of ETS units that were required to be turned on. The multicast message would contain a command to turn on. Another multicast message would be sent to the remaining IP addresses within the VPN telling those ETS units to turn off. Using this approach, all ETS units can be directed to turn on or off using two multicast messages. The problem with this approach is that broadcast messages are not guaranteed to be delivered. In this particular case, it would not be a significant issue. To mitigate the risk of missing a message from the server, the same message could be sent several times. ETS units that had already received the message would already be in the correct state. Even if an ETS unit missed a message, it would take many ETS systems to make a significant impact on grid stability.

2.6.2.2 Broadcast

In a broadcast system, the server sends a command to all devices simultaneously within a network with no regard to what information the broadcast message contains. Each device receives the message but what is done with the data contained within the message is handled at higher levels of the OSI model. Using a broadcast approach would require the development of a packet that can be delivered to each device within the system that contained enough generic information to allow each instruct each device to turn on or off. A corresponding higher level OSI layer driver would also be needed by the storage devices to process this packet and either turns on or off. Broadcast messages do not have guaranteed delivery. This fact would also need to be considered in developing a system to distribute intermittent renewable energy.

2.6.2.3 Unicast

In a unicast approach, the server would be balancing the number of ETS units that are on or off with the amount of generated wind electricity by sending individual commands to ETS units. Since the server is communicating with a single device, acknowledgement of the command can be guaranteed. Care must be taken in this type of implementation that the server has enough resources to communicate to all the ETS units within the system within a reasonable amount of time. If there are more ETS units then the server can comfortable handle, additional servers and a load balancing layer would need to be added to ensure timely communication.

2.7 Summary

This chapter has shown that wind power is expected to play an important role in an energy future that uses increasing amounts of renewable energy. The issue of wind intermittency has also been shown to be its most challenging obstacle for integration into the existing power grid. The development of a new “Smart Grid” that uses a variety of technologies to increase the use and delivery of intermittent renewable energy is currently in development by a number of governments and will form the backbone of our future energy grid. Finally, the use of existing internet technologies has been explored for their suitability in developing a system for the distribution intermittent renewable energy sources. This research into using existing technologies in a new implementation for power distribution can be considered a “Smart Grid” activity.

The next chapter will explore the design and implementation of client pull and server push systems for the distribution of intermittent power to ETS units for residential space heating. The goal of these implementations is to use all the available wind power for space heating thereby reducing the carbon intensive non renewable energy sources needed and increase a jurisdictions energy security.

Chapter 3 System Design

This chapter presents the design of both the Server-push and Client-pull systems for the distribution of variable supplies of wind-electricity for storage in ETS devices. Although this work focuses on wind and ETS devices, each system is capable of accepting variable supplies of energy from any source for distribution to any device capable of storing energy for later use.

Modeling of the generation, distribution and use of wind generated energy for space heating is explored using energy chains to identify the processes, flows, and energy stores. The proposed implementations have been designed to have similar system architectures to reduce the design, implementation, and testing cycles for both systems. Frameworks for both implementations are first presented to show the similarities and differences of the approaches. This is followed by a detailed discussion of the system component requirements. The strengths and weaknesses of each system are then discussed.

3.1 The Client-pull Framework

Client-pull was defined in 2.6.1 as a system where client devices initiate a conversation with the server for the exchange of information. The server in this system has a passive role, only responding to requests made by the client devices. The framework must be designed to provide the processes and components necessary to distribute intermittent wind energy to ETS units when there is a surplus.

Distribution of energy requires accurate information on the current production information from the energy producers. To provide this information in a client pull framework, each producer must have a means of transmitting its production information. Each ETS system will also have operational information that will need to be transmitted for the system to make distribution decisions. The final requirement for the system is the current system load.

Figure 4 shows a generalized Client-pull framework for the distribution of intermittent wind energy to ETS devices satisfying the requirements explained above. It includes external sources of energy. The contribution of the external sources will depend on the on-demand electricity levels in the jurisdiction and the generation level of intermittent wind energy and supplement any domestic energy shortfall.

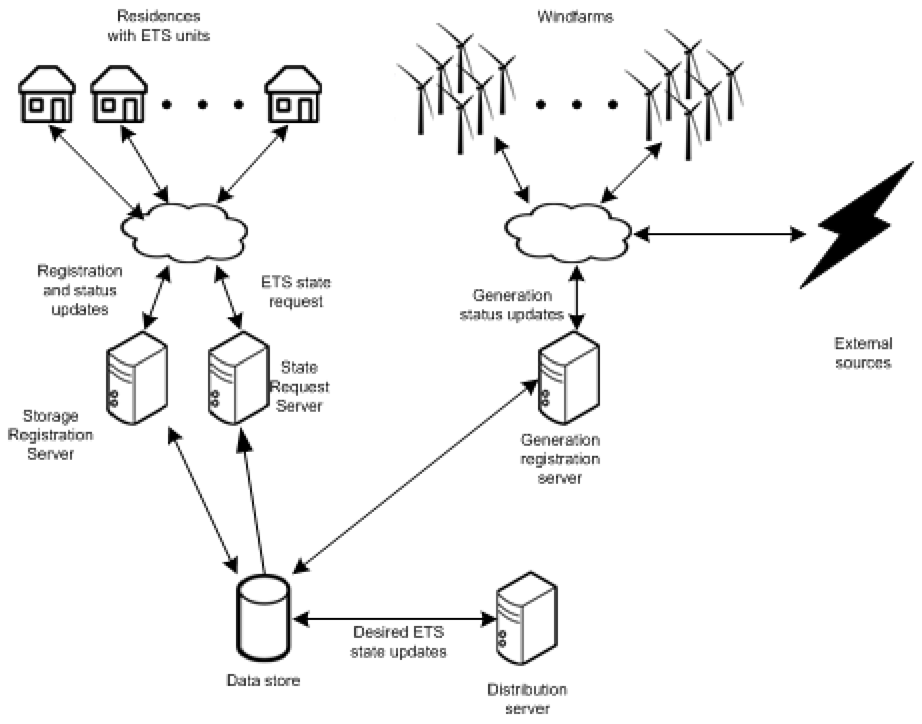

Residences with ETS units
Windfarms
Registration and status updates
ETS state request
Generation status updates
External sources
State Request Server
Storage Registration Server
Generation registration server
Desired ETS state updates
Data store
Distribution server

Figure 4 Client-pull Framework

The framework shown in Figure 4 has four servers. Each server is designed to handle a specific task required to implement the Client-pull Framework. These tasks include: producer registration and reporting, storage device registration, storage device state request, and energy distribution. Having four independent servers provides server resources to support large volumes of communication traffic.

The distribution of wind energy requires access to data from the energy producers, the ETS units, and the current on-demand electricity level. The storage of data for access by distributed systems is accomplished using a database. Before discussing the processes, the format of the database used by the system is first explained.

3.1.1 Database Schema

This framework requires that the ETS systems, power producers, and system processes exchange information. This information is generated and used asynchronously by entities within the system.   The information is stored in a system of database tables as it is generated and read from the tables when it is needed. The tables can be divided into two categories, input tables that contain data used by the system to distribute energy and archive tables used to record a history of energy production, usage, and distribution.

A minimum of four input tables are required for the Client-pull framework, each of which is described below:

Authentication Table: This table holds device id/password pairs for all the devices which can communicate with the system.   To gain access a device must pass a device id and an assigned password for that device. The use of device id/password pairs is required to prevent unauthorized system access by rogue systems that may try to introduce instability or compromise system security.

Storage Unit Table: This table holds a record of the currently allocated energy to each storage device. This table will have a device id field to identify the device, an energy allocation field to record the energy allocated to the device, and a time stamp to record when the energy was allocated. Summing the energy allocation field will indicate the total amount of energy allocated to storage units.

System Load Table: This table holds a record of the energy currently being used by loads within the system which are not storage devices. This table will have a device id field to identify the load, an electrical load field to record the energy being used by the device, and a time stamp to record when the energy was allocated. Summing the loads will give the total system load for the district.

Producer Table: This table holds a record of the energy currently being generated by the energy producers. It requires a device id field to identify the producer, an output field to record the energy being produced by the device, and a time stamp to record the last time the record was updated. Summing all the output fields give the total energy being produced in the district.

The archive tables are used by the system to record historical information on the production of intermittent energy, the system load, and the state of the ETS systems. This information is needed for evaluating system performance, new distribution algorithms, and as an archive for debugging in the event of a system failure. This framework uses four tables to archive system information.

Energy Allocation Table: Each time energy is allocated to a storage device, a record is added to this table. Each entry in the table records the device ID, the total energy currently being produced, the minimum and maximum possible recharge rates for the storage device, the current discharge rate of the storage device, the external temperature, the energy allocated to the device, a record index, and a timestamp of when the allocation was made.

System Load Archive: Each time a system load reports a change in its energy usage, a record is added to this table that records the device id, the energy usage, and a time stamp.

Producer Archive: Each time a energy producer reports a change in its energy production, a record is added to this table that records the device id, the energy production level, and a timestamp.

Emergency Archive: If an ETS unit becomes completely depleted during the daytime hours it must draw electricity from the grid to meet the space heating needs of the residence. If this occurs, the ETS unit writes a record to this table of the ETS device id, the time, and the energy required. This is used as a debug tool.

These tables are used by processes within the Client-pull framework. The following sections describe these processes and show the usage of the database tables.

3.1.2 Producer Registration and Reporting

The first process is the reporting and storage of wind energy production levels. Each wind-farm monitors and reports the current level of energy production to a generation registration server. This reporting is done using a data communication link that provides encrypted transportation, authentication, and guaranteed delivery (failure of delivery notification). This update will occur when either a specified time interval has passed since the last update, or when the production level has changed by a predetermined amount. A server will authenticate the client and archive a time stamped record of the production level the producer archive table and update the record in the producer table for this energy producer for later processing. After the data has been successfully archived, the server will reply to the client with the results of the archiving operation.

The key aspects of this process and an explanation of why they are needed are given below:

Encrypted transportation: The client will be transmitting a device id/password pair. Without encryption, this could be compromised by a rogue system and used to report incorrect production levels leading to grid instability.

Authentication: Without an authentication layer, a rogue system could pose as a wind farm and report incorrect production levels leading to grid instability.

Guaranteed delivery: The wind farm must ensure that its production data is reaching the server. If communication fails due to any one of a number of reasons, the wind farm must retry until communication is reestablished and delivery confirmed. This is to ensure the server has the most up to date generation data for energy distribution.

Update on Time Interval or Production Level Change: In this system, the intermittent energy producer is responsible for giving the server up to date production information. If for some reason the wind farm goes offline, it will not provide generation data to the server. For this reason, when a wind-farm reports generation data, the time must be recorded. By checking the time of the last update, the system can decide if the information is accurate enough to be used. It is also possible for the wind farm not to report production information if there has been no change in the production levels without a forced time interval update to the server. For this reason, the energy producer must report production information every time there is a change in production levels (the actual quantum required for an update would be jurisdiction dependant) or after a maximum period of time has passed (also jurisdiction dependant) since the last update. If the interval between the current time and the time of the last production update is greater than the maximum period allowed between production level updates, server can assume that the producer is off line and its production information is unreliable.

If the energy output of the wind farm were to change drastically over a period of time shorter than the forced update interval, the system would be using inaccurate generation data. Using a forced update when a predefined production level change has occurred since the last update ensures the system will always have accurate production levels for distribution to storage units. This process is described by Figure 5.

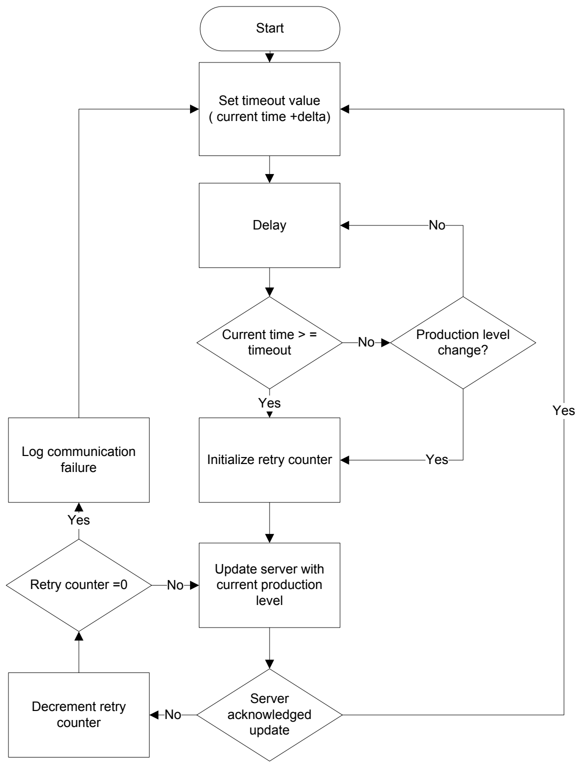


Figure 5 Intermittent Energy Producer Registration

3.1.3 Storage Device Registration

In this system, energy storage devices (ETS units) register with the system. When a device registers, it must report how much power it can absorb while recharging and its current charging state. During the evening hours, ETS units may be charging at a nominal rate to reach full charge by the end of the evening recharge cycle. If the minimal recharge rates of the ETS systems were ignored during the evening hours, the distribution system would allocate all the intermittent renewable energy in addition to the minimum recharge rate of the ETS units, forcing the system to use non-renewable energy sources for space heating. There is also no guarantee that all storage devices will be the same model, or be made by the same manufacturer. As a result the maximum recharge rate of each ETS system is needed by the distribution system to optimize the use of intermittent renewable energy. Registration is done using a data communication link that provides encrypted transportation, authentication, and guaranteed delivery (failure of delivery notification). Registration occurs at a specified time interval. A server will authenticate the client and archive a time stamped record of the registration, current operating state, and maximum recharge rate. This newly added ETS system will be queued for inclusion in the distribution algorithm during the next distribution operation. After the data has been successfully archived, the server will reply to the client with the results of the registration. When devices have been fully recharged they unregister, removing them from the intermittent energy distribution system.

The encryption, authentication, guaranteed delivery, and registration requirements of storage device registration are the same as those covered in 3.1.2.

Registration Timeout: As a client pull system, the clients are responsible for providing the server with accurate, up to date information. If an ETS system were to go offline due to a power outage or communications link failure, the distribution system would still think the ETS unit was active. By forcing a maximum time between registrations, the system can ensure that the registered ETS units are active by checking the current time against the registration time. Any units that had a registration time beyond the maximum time between registrations would be assumed to be off line, and not used for power distribution.

The ETS systems must also be intelligent enough to revert back to a standalone operation mode if communication with the server fails for an extended period of time. The ETS should continue to attempt to register with the system until the communication link is reestablished. This process is described in Figure 6.

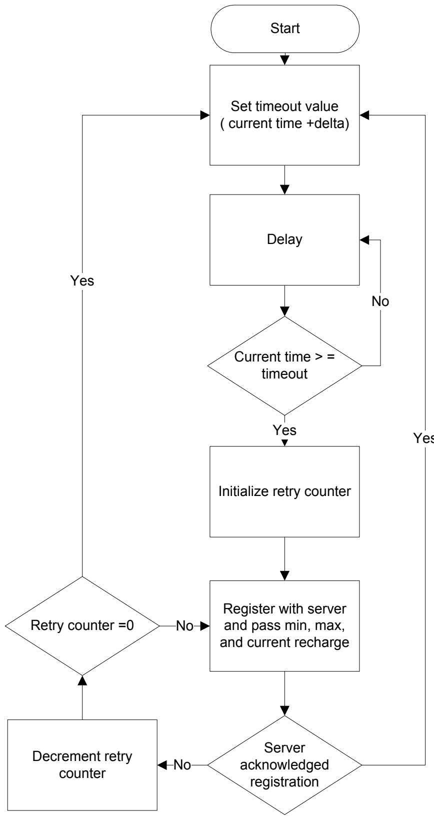

Start
Set timeout value (current time +delta)
Delay
No
Current time > = timeout
Yes
Initialize retry counter
Register with server and pass min, max, and current recharge
Server acknowledged registration
No
Decrement retry counter
Retry counter =0
No
Yes
Yes

Figure 6 Client-pull Storage Device Registration

3.1.4 Storage Device State Request

The ETS units will use a data communication link that provides encrypted transportation, authentication, and guaranteed delivery (failure of delivery notification) to periodically request instructions from a server to set its recharge rate. This service is provided by the state request server. The ETS device will pass its device id and password to the state request server. The state request server will then authenticate the ETS against the authentication table. The registration server then uses the device id that was passed from the ETS system to read the energy allocation for this ETS system from the storage unit table. The allocation information is then passed back to the ETS system. Since all ETS units will be communicating with this server, it may experience high traffic if there are a large number of ETS units registered or if the update period for receiving instructions is too short.

The encrypted transportation and authentication and guaranteed delivery requirements as the same as those covered in section 3.1.2, Energy Distribution

This process distributes the generated intermittent energy from the registered intermittent producers to the registered storage units (ETS systems). The distribution algorithm must account for the energy used during the evening hours for recharging and new units that have registered since the last energy distribution has been calculated. It should also allow each registered ETS system equal access to the available energy. The flowchart in Figure 7 describes the process.

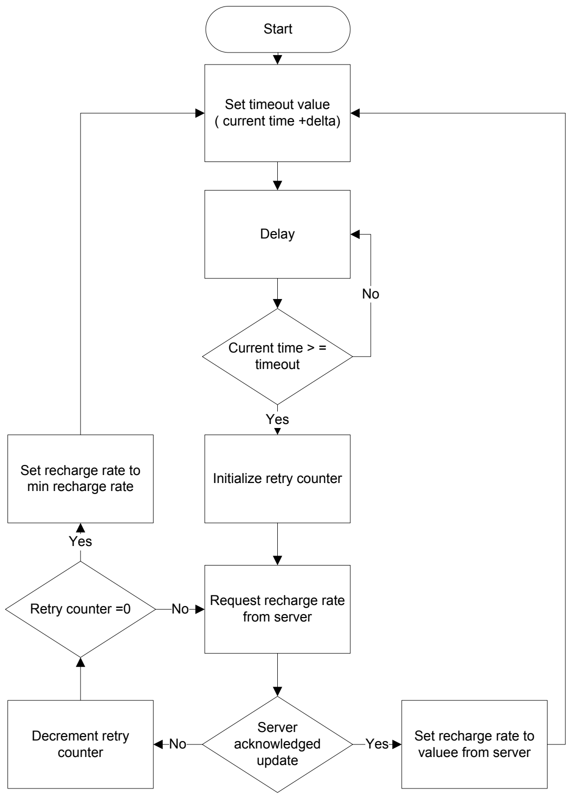

Start
Set timeout value (current time +delta)
Delay
No
Current time > = timeout
Yes
Initialize retry counter
Set recharge rate to min recharge rate
Yes
Retry counter =0
No
Request recharge rate from server
Decrement retry counter
No
Server acknowledged update
Yes
Set recharge rate to valuee from server

Figure 7 Client-pull Storage Unit State Request

3.1.5 Energy Distribution

This process does not directly touch any external systems. The data used for the distribution is extracted from the database input tables and the new distribution information is updated to the database in the storage units table. An archive of each allocation is stored in the energy usage table. This process should run whenever there is a change in a producers output, or when a time interval expires. A flow chart describing the process is shown in Figure 8.

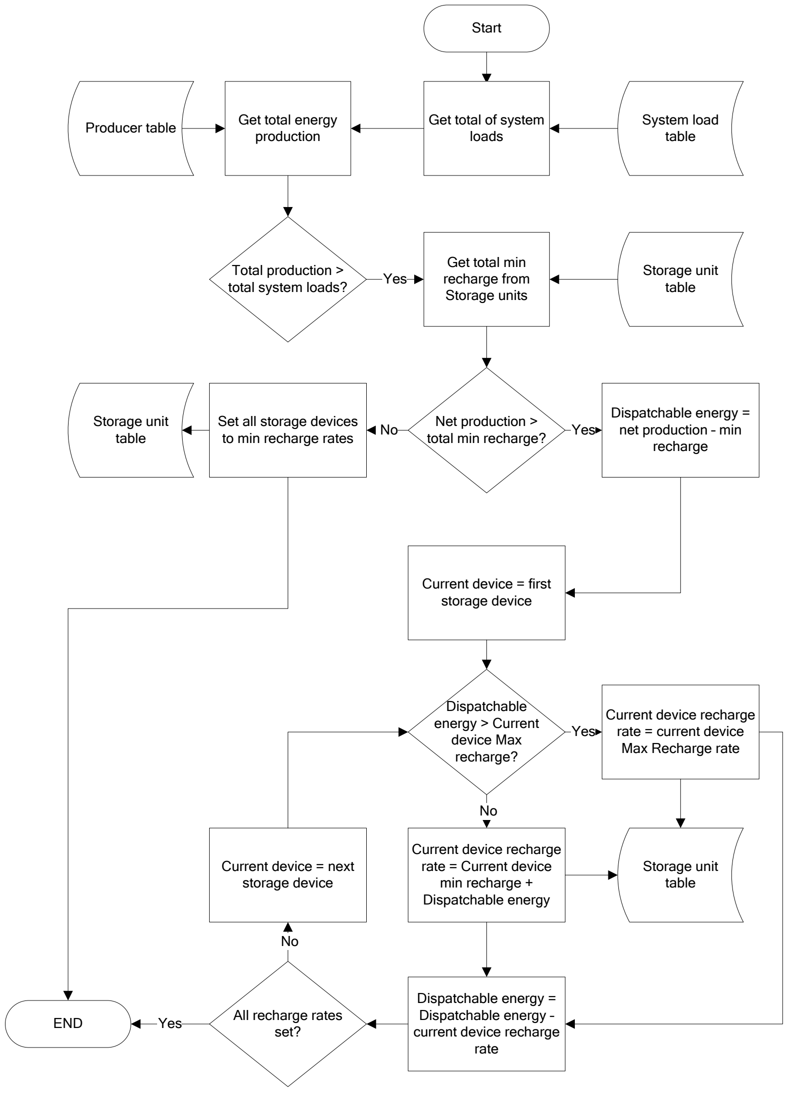
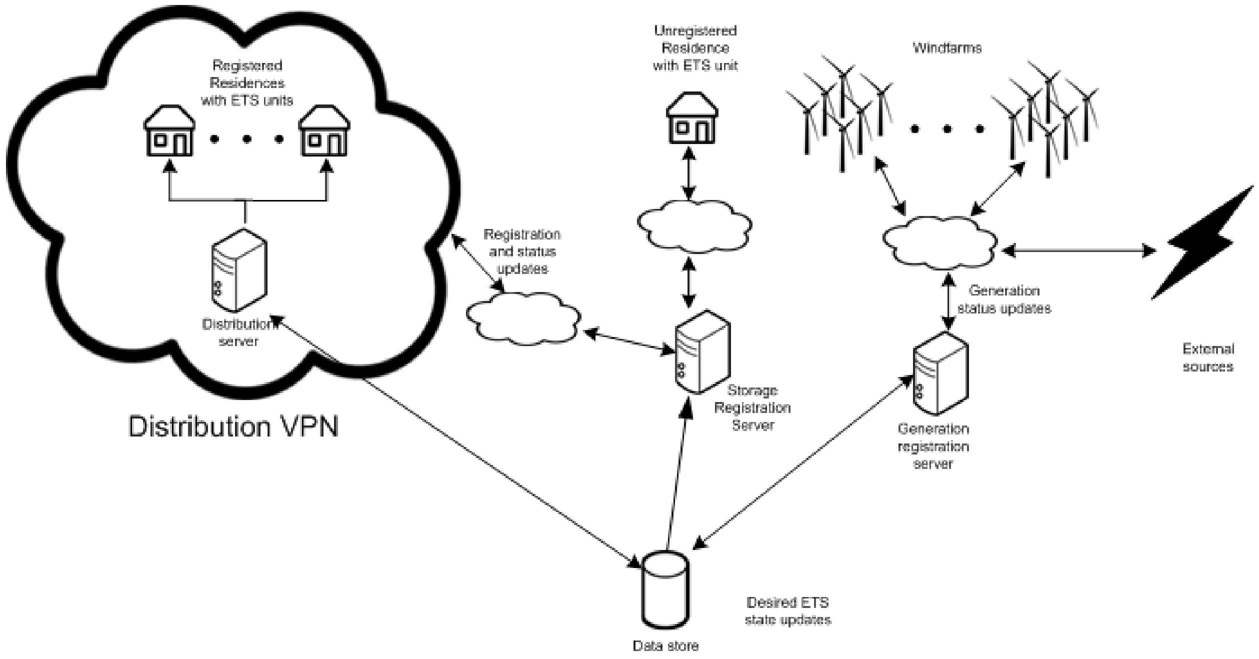

Start
Producer table
Get total energy production
Get total of system loads
System load table
Total production > total system loads?
Yes
Get total min recharge from Storage units
Storage unit table
Storage unit table
Set all storage devices to min recharge rates
No
Net production > total min recharge?
Yes
Dispatchable energy = net production - min recharge
Current device = first storage device
Dispatchable energy > Current device Max recharge?
Yes
Current device recharge rate = current device Max Recharge rate
No
Current device = next storage device
Current device recharge rate = Current device min recharge + Dispatchable energy
Storage unit table
No
END
Yes
All recharge rates set?
Dispatchable energy = Dispatchable energy - current device recharge rate

Figure 8 Client-pull Energy Distribution

3.2 The Server-push Framework

Section 2.6.2 introduced Server-push concept where information exchange is initiated by the server device and delivered to the client devices for processing. This exchange can be initiated either through a broadcast where all devices receive message sent from the server, multicast where the message is sent to all the client devices, or a unicast message where the server initiates a unique conversation with each client. Figure 9 presents a framework for a Server-push system for the distribution of intermittent wind energy to ETS devices. It includes external sources of energy. The contribution of the external sources will depend on the on-demand electricity levels in the jurisdiction and the generation level of intermittent wind energy and supplement any domestic energy shortfall.



Registered Residences with ETS units
Distribution server
Distribution VPN
Registration and status updates
Unregistered Residence with ETS unit
Storage Registration Server
Windfarms
Generation status updates
Generation registration server
External sources
Desired ETS state updates
Data store

Figure 9 Server-push Framework

This implementation differs from the client pull in that after an ETS registers with the system it joins a Virtual Private Network (VPN) to receive commands from a distribution server. The commands from the distribution server are delivered using a network broadcast protocol that is received and decoded by all ETS units simultaneously.

The database tables used in this framework are identical to that in the Client–pull discussed in 3.1.1 This system can be divided into four processes; New device registration, ETS units joining the VPN, Generation registration/reporting, and power distribution. Each of these processes is described below.

3.2.1 New Device Registration

An ETS device registers with the system through a registration system similar to the client pull implementation. ETS units use encrypted authentication and provide the power requirements of the ETS system during recharge. The server first authenticates the username/password pair, then checks to see if the device is already registered with the system. If the device is registered, it simply returns acknowledgement that the registration succeeded. If the device is not already registered, the device is added to the database as a new unit and the reply contains credentials needed to join a Virtual Private Network used for power distribution, and issued a Distribution ID used by the distribution server to identify this ETS unit within the VPN. Like the client pull implementation, ETS units are required to re-register with the system within a predetermined time window since it last registered. When devices become fully charged, they would unregister with the system, allowing other ETS systems more access to the available intermittent energy.

The encrypted transportation, authentication, and guaranteed delivery requirements as the same as those presented in 3.1.2. The registration timeout requirement is the same as that presented in 3.1.3. The Flowchart in Figure 10 describes the registration process.

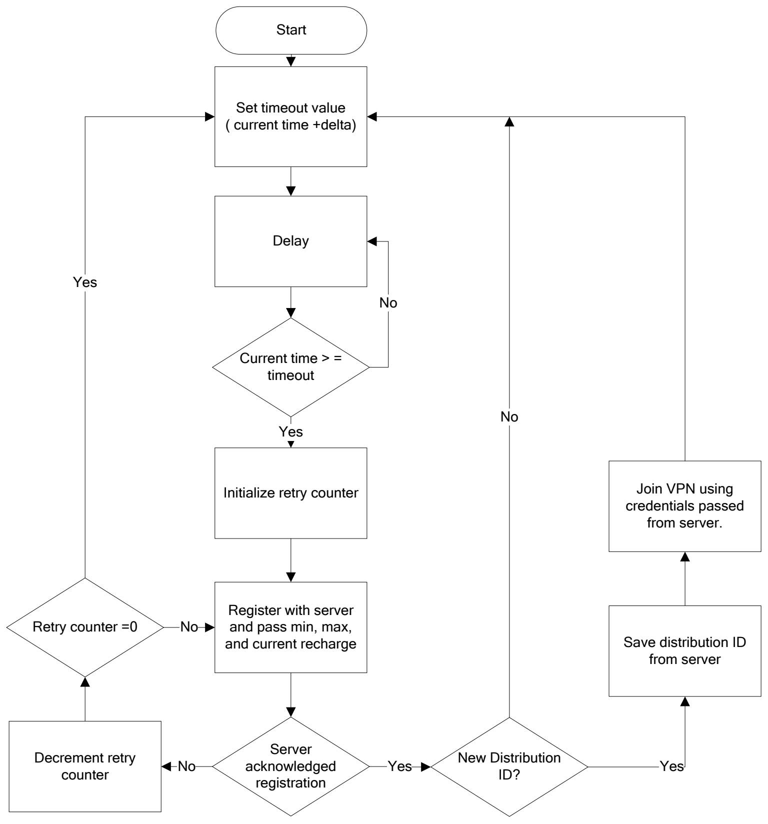

Start
Set timeout value (current time +delta)
Delay
No
Current time > = timeout
Yes
Initialize retry counter
Register with server and pass min, max, and current recharge
Retry counter =0
No
Yes
Decrement retry counter
No
Server acknowledged registration
Yes
New Distribution ID?
No
Yes
Save distribution ID from server
Join VPN using credentials passed from server.

Figure 10 Server-push Storage Device Registration

ETS Units Joining the VPN: Upon receiving credentials from the Registration server, the ETS unit will then join a VPN using the supplied address, username and password. After registration, it then prepares to receive multicast messages from the VPN distribution server for processing. The VPN must use encrypted transportation for sending information to other devices within the VPN.

The key aspects of the VPN are authentication and encryption which have already been covered in 3.1.2.

3.2.2 Producer Registration and Reporting

This process is identical to the one described in the Client-pull implementation in 3.1.2.

3.2.3 Power Distribution

Power distribution in the Server-push implementation shares many of the same aspects as the power distribution in the client pull system. Power distribution would be calculated the same way in both systems, but the server push implementation The Distribution server would use the Distribution IDs issued to the ETS systems during registration to send a broadcast message to the ETS units within the VPN. This message must be small enough to fit within a single multicast datagram to reduce system latency and eliminate the additional complexity of assembling a message from multiple datagrams. The message should contain have an appended checksum.

Broadcast Message: A broadcast message has the advantage of reaching all devices at virtually the same time, reducing network traffic and the latency of having each device poll for operation instructions. Limiting the size of the message to a single multicast datagram removes the requirement of assembling a command from multiple datagram's, simplifying ETS reception.

Checksum: By adding a checksum to the data transmitted in the Multicast packet, the ETS unit can ensure that the data was not modified during transmission or reception. Packets that failed to pass the checksum test would be discarded by an ETS system.

The flowchart in Figure 11 shows the process for energy distribution for Server-push.

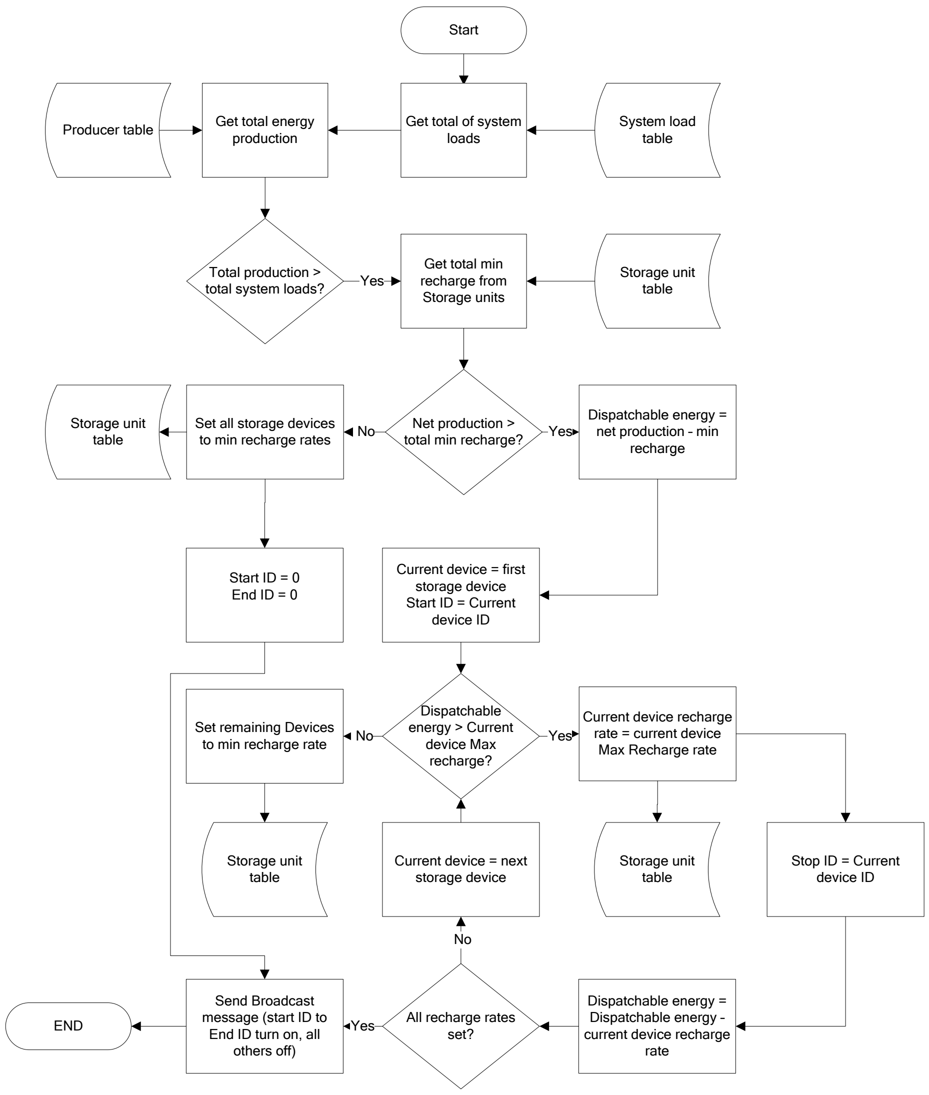

Start
Producer table
Get total energy production
Get total of system loads
System load table
Total production > total system loads?
Yes
Get total min recharge from Storage units
Storage unit table
Storage unit table
Set all storage devices to min recharge rates
No
Net production > total min recharge?
Yes
Dispatchable energy = net production - min recharge
Start ID = 0
End ID = 0
Current device = first storage device
Start ID = Current device ID
Set remaining Devices to min recharge rate
No
Dispatchable energy > Current device Max recharge?
Yes
Current device recharge rate = current device Max Recharge rate
Storage unit table
Current device = next storage device
Storage unit table
Stop ID = Current device ID
No
END
Send Broadcast message (start ID to End ID turn on, all others off)
Yes
All recharge rates set?
Dispatchable energy = Dispatchable energy - current device recharge rate

Figure 11 Server-push Energy Allocation

3.3 Residential Modeling and Energy Efficiency

The energy intensity of a residence for space heating will depend on several factors. These factors include the size of the residence, the desired internal temperature, the external temperature and the general construction of the home. This section contains an analysis of these parameters and how they can be used to derive a model for estimating residential space heating energy usage.

3.3.1 Degree Days

Heating degree-days for a given day are the number of Celsius degree that the mean temperature is below a target temperature. If the temperature is equal to or greater than the target temperature, then the number will be zero [52]. Degree days are useful in estimating the energy requirements of a residence based on the temperature difference. To model a system as accurately as possible degree hours or degree seconds provide a finer time scale.

3.3.2 Residential Parameters

The residential parameters that affect the energy intensity for space heating in a residence can be determined from The Office of Energy Efficiency for Canada Comprehensive Energy Use Database. This database provides statistics on the total energy used for space heating by vintage, the housing stock by vintage, and the floor space by vintage [53]. This allows the calculation of the Energy Intensity per square meter by vintage using Equation (3.1).

$$\boldsymbol{Energy\ Usage\ by\ vintage}\ \left({}^{J}\!/_{\boldsymbol{m^2}}\right) = \frac{\boldsymbol{Total\ Energy\ Use\ by\ Vintage\ (J)}}{\boldsymbol{Total\ Floor\ Space\ by\ Vintage\ (\ m^2\ )}} \qquad \textbf{(3.1)}$$

Dividing the results of Equation (3.1) by the number of degree seconds (degree hours multiplied by 3600) gives an energy intensity with units of J/(°C sec m^2) presented in Equation (3.2)

$$\boldsymbol{Energy\ Intensity}\ \left(\frac{\boldsymbol{J}}{{}^\circ\mathrm{C}\,\boldsymbol{\sec\ m^2}}\right) = \frac{\boldsymbol{Energy\ Usage\ by\ vintage}\ \left({}^{J}\!/_{\boldsymbol{m^2}}\right)}{\sum_{\boldsymbol{Jan\ 2008}}^{\boldsymbol{Dec\ 2008}} {}^\circ\mathrm{C}\ \boldsymbol{hours} \times \boldsymbol{3600}} \qquad \textbf{(3.2)}$$

The product of the Energy Intensity, the number of heating degrees, the interval of time, and the residence size gives an estimate of the energy used for space heating for that period of time.

3.4 Summary

This chapter presented two different frameworks for implementing a control system for the distribution of intermittent renewable energy using existing technologies applied in a nontraditional way. The client-pull framework model is taken directly from the implementation of the World Wide Web and RESTful web services. The Server-push implementation is similar to the technology used to provide television over the internet. A method to accurately model the residential space heating needs has also been presented. This method could be applied to any jurisdiction being modeled using the framework implementations. The next chapter presents an implementation of the frameworks and residential modeling of Summerside PEI.

Chapter 4 Framework Implementations

This chapter presents the implementations of the client-pull and server-push frameworks. From the design discussion in Chapter 3, both require input from the intermittent energy providers and registration of the energy storage devices when they can store energy and deregistration when they cannot. The significant difference between both implementations is in how the energy storage devices are controlled. These implementations run in a simulated environment but use the system components and protocols defined in this chapter.

The software implementations have been designed to use as much code reuse as possible to reduce the development and testing cycles and therefore have identical registration implementations for both the intermittent energy providers and the energy storage devices.

4.1 Communication Protocol

Both implementations use the HTTP protocol when a reply from the server is required by the storage device or intermittent energy producer. This protocol provides the guaranteed delivery (or failure notification) required by the design and described in Chapter 3. Security can be provided by using the SHTTP protocol which uses ordinary HTTP over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection. The encryption occurs at a lower layer of the OSI model and is decrypted to standard HTML before being passed to the Application Layer. Any system that implements a HTTP protocol can easily be migrated to use the encrypted HTTPS protocol by changing the server configuration.

4.1.1 RESTful Implementation

The three defined aspects of a RESTful web service were defined in 2.6.1. The first requirement is easily met through design, while the second enforces a standard form of data transmission; these implementations will use plain text for transmission. The third requirement limits the methods used to operate on the transmitted data using the HTTP protocol.

Each transmission from either an energy storage device or an intermittent energy producer contains a device-id/password pair used for authentication. For compliance as a RESTful web service and follow the recommendations in the HTTP protocol specification all commands sent to the distribution system will be done using the POST command.

4.1.2 Software and Tool Selection

Web services are provided through the use of web-servers configured to provide data rather than web pages. These types of web servers are commonly called Application Servers. There are many commercial Web servers available including Oracle WebLogic 11g Application Server [54], Microsoft Internet Information Server [55], and IBM WebSphere Application Server [56]. In addition to these commercial products, there are also a number of free application servers including Apache Tomcat [57], Oracle Glassfish Server [58], WebSphere AS Community Edition [59], Jetty [60], and JBoss [61] to name a few. Apache Tomcat web server was chosen to provide the required web services; it uses the Java language to implement servlets that handle HTTP requests from web clients. Servlets are similar to Java applications, but are executed by Tomcat to process a request for a web service handled by the Servlet. The Java API implements a base class for servlets that has default handlers for the defined HTTP commands (GET, PUT, POST). If a web service implements one or more of these commands, the application developer overrides the default handler and creates their own method for handling the request. Java applications (and servlets) can be written using a simple test based editor and compiled using a command line compiler, but integrated design environments (IDEs) have been created to allow for much smoother application development and debugging. This work uses the Java Eclipse IDE for the Web Service application development. The Tomcat Web Server integrates with the Eclipse IDE to simplify the development and testing cycles of Web Service design.

The protocol requirements of the Client-Pull framework can be completely met using HTTPS. The Server-Push framework requires a protocol supporting multicast and broadcast to send commands to the energy storage devices at the same time. The selected protocol to meet this aspect of the server push implementation is UDP.

Both framework implementations use a data store to hold information from the distribution of wind energy. These framework implementations use a MyQSL data base as the data store. MySQL provides an API that allows Java applications to connect, execute SQL commands, and process query replies for a MySQL database. This software API is implemented in a Java library called JConnector. The MySQL database and JConnector are both available for free download from www.mysql.com. A detailed description of the tables used in these implementations can be found in Appendix A.

4.2 Client-pull Implementation

This section provides implementation details on how the tables described in Appendix A are used to implement a Client-Pull architecture based on the framework described in Chapter 3.[1]

4.2.1 System Load

The system requires information of the current system load to determine if there is surplus energy to allocate to the storage devices. The system load in this simulated environment is implemented as a object oriented software construct call a class. A class is used in object oriented programming to implement the state and behavior of an entity. In this case, the class models the behavior of the system load by reading the data stored in the Wind Data Table based upon the current time as seen by the simulated environment to update its state, which is the system load in kWh. Upon updating its state by reading new data from the Wind Data Table, the class connects to the database and creates a new record in the System Load Table if one does not exist. If a record for this load is already present, it simply updates the records value. In the simulated environment, a servlet was not used to register the system load because the additional HTTP traffic on a single PC simulating the entire environment including the ETS unit, web server, and database was found to make the simulation run excessively slow.

4.2.2 Producer Registration

Intermittent energy producers register with the system to allow the energy they produce to be distributed to the registered storage devices. The framework presented in section 3.1.2 describes how these requirements can be met using a RESTful web service.

[1] A system that models the framework in Chapter 3 was implemented but due to the processing power available on a single PC, it was found that it was necessary to simplify simulation. The implementation presented combines some aspects of the framework but leaves the intent of a client-pull system intact. Only two servlets are used, one for producer registration and one for storage devices that combines the energy distribution and registration for storage units.

The flowchart in Figure 12 describes the implementation of a RESTful web service Servlet for producer. For clarity, some aspects of the source code such as error and range checking have been omitted from the flowchart.

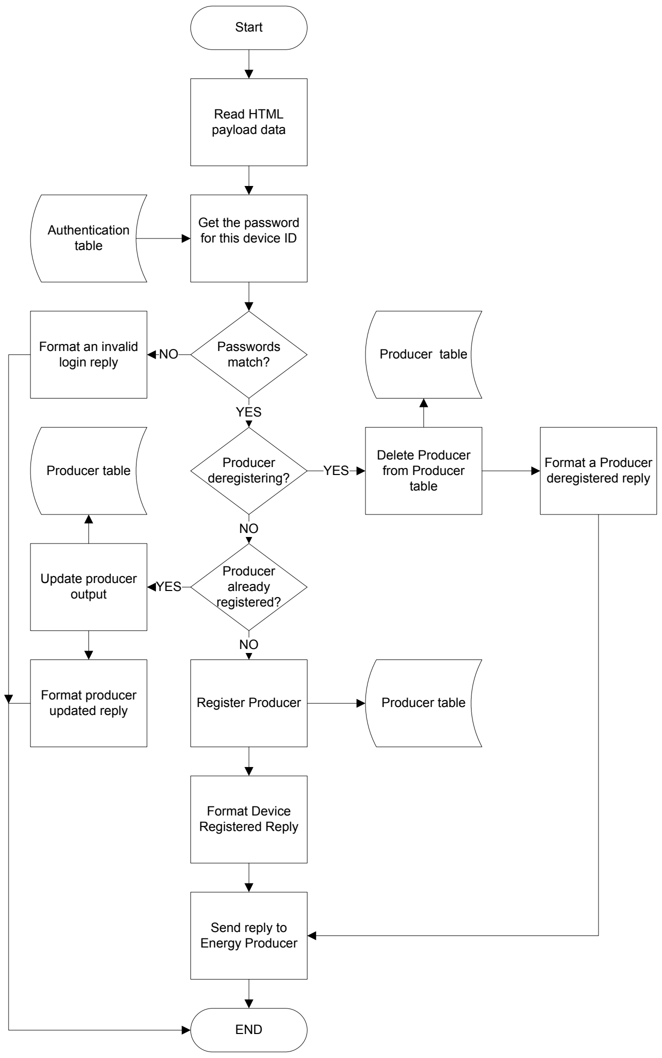

Start
Read HTML payload data
Authentication table
Get the password for this device ID
Format an invalid login reply
NO
Passwords match?
Producer table
YES
Producer table
Producer deregistering?
YES
Delete Producer from Producer table
Format a Producer deregistered reply
NO
Update producer output
YES
Producer already registered?
NO
Format producer updated reply
Register Producer
Producer table
Format Device Registered Reply
Send reply to Energy Producer
END

Figure 12: RESTful Web Service Producer Registration

4.2.3 Storage Device Registration/State Request

This servlet combines the distribution and registration servlets for storage devices. This implementation does not use device authentication to decrease the time required to respond to a storage device. An operational system would require the authentication step, but because we are simulating and not performing a security audit of the system it can be bypassed.

Figure 13 presents a flowchart that implements the storage Device Registration/State Request servlet. For clarity, some aspects of the source code such as error and range checking have been omitted from the flowchart.

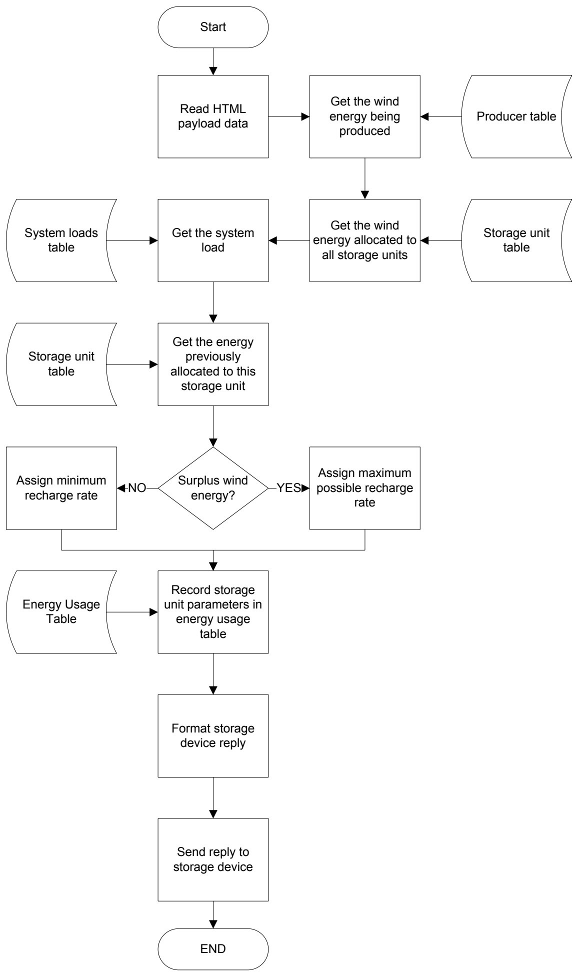

Start
Read HTML payload data
Get the wind energy being produced
Producer table
System loads table
Get the system load
Get the wind energy allocated to all storage units
Storage unit table
Storage unit table
Get the energy previously allocated to this storage unit
Assign minimum recharge rate
NO
Surplus wind energy?
YES
Assign maximum possible recharge rate
Energy Usage Table
Record storage unit parameters in energy usage table
Format storage device reply
Send reply to storage device
END

Figure 13 RESTful Web Service Storage Device Registration

4.3 Server-push System

This section provides implementation details on how the tables described previously are used to implement a Server-Push system based on the framework described in Chapter 3.[2]

[2] As with Client-Pull, the limitations of using a single PC to model the entire system forced some redesign of the framework but the spirit of the server-push system was preserved. In the framework defined in chapter 3, all storage devices monitor the same port for a command sent by the server that contains a packet of data specifying who should be on and who should be off. However, on a single PC it is not possible to bind multiple threads to a single port. As a result, an intermediate process was used that monitored the port for commands sent from the

4.3.1 System Load

The system load is simulated using the same object oriented class defined for the Server-Push implementation in section 4.2.1.

4.3.2 Producer Registration

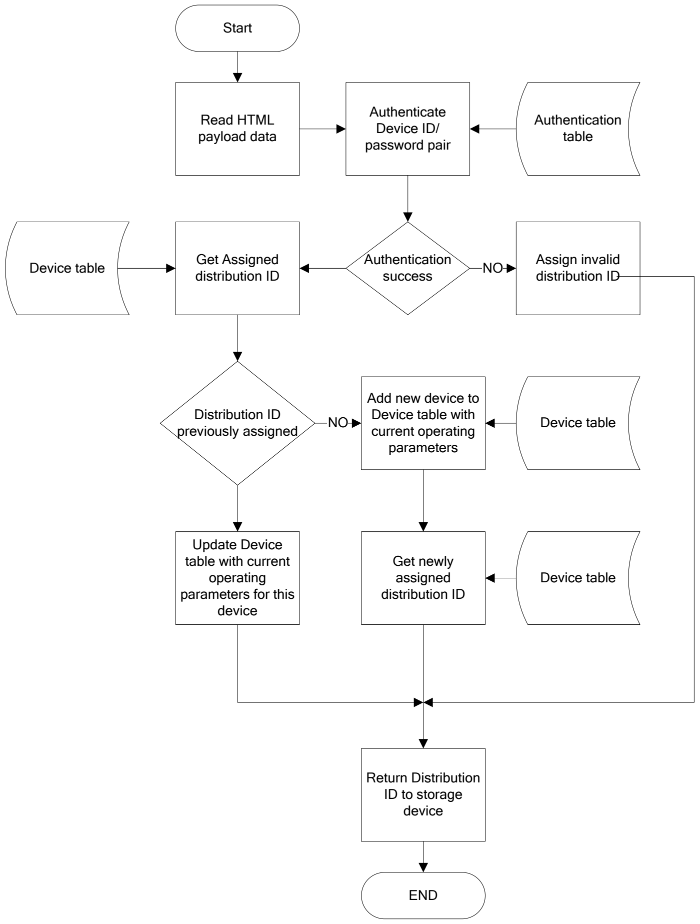
The energy producers in the Server-Push implementation report their energy production levels identically to the method presented in section 4.2.2.

4.3.3 Storage Device Registration

Storage devices are required to register with the system to be included in the distribution of intermittent energy. In the Server-Push implementation, a storage device is returned a distribution ID when it registers with the server. This distribution ID is a unique ID allocated to a single storage device. This number is assigned incrementally with the first device to register begin assigned ID 1, the second device ID 2 and so on. This allows the server to identify a range of storage devices that should act on a packet containing command instructions by specifying a range of ID’s. During device registration, operational parameters are passed from the storage device to the server. Devices continually register with the server on a periodic basis, or when their operational parameters have been changed by a command sent by the server. If a device has previously registered with the server, it is returned the distribution ID that has been previously assigned.

This is implemented as a servlet that runs on the web server associated with the system. A flow chart describing its operation is shown in Figure 14.

distribution process, then serially sent the command to the storage units which were modeled as application threads.



Start
Read HTML payload data
Authenticate Device ID/ password pair
Authentication table
Device table
Get Assigned distribution ID
Authentication success
NO
Assign invalid distribution ID
Distribution ID previously assigned
NO
Add new device to Device table with current operating parameters
Device table
Update Device table with current operating parameters for this device
Get newly assigned distribution ID
Device table
Return Distribution ID to storage device
END

Figure 14: Storage Device Registration

For clarity, some aspects such as error and range checking have been omitted from the flowchart.

4.3.4 Energy Distribution

Energy distribution in the Server-Push architecture uses a broadcast message sent to all ETS devices simultaneously using UDP. The full implementation of this framework requires a security layer for protection from rogue computers that attempt to introduce grid instability. The system presented in this work does not include the security layer since the increased processing power required was not available in a single PC modeling an entire system.

Broadcast Message

A Broadcast message has the advantage of reaching all devices at virtually the same time, reducing network traffic and the latency of having each device poll for operation instructions.

Limiting the size of the message to a single multicast datagram removes the requirement of assembling a command from multiple datagrams, simplifying ETS reception. If a broadcast message is missed by a client due to network delivery failure, or checksum failure, the ETS device will take no action and remain in its current state. This will introduce an error in the distribution of energy if the state should have changed but did not. Section 3.2.1 the Server-push framework discussion enforced a rule where ETS devices must re-register with the system after a predetermined amount of time. The state of the ETS device during re-registration could then be checked against the desired state. If the states were found not to match, the server could include an instruction to set the state of the ETS in its reply. This was not implemented because failure to receive a datagram was not part of the simulation.

The format of the broadcast message packet can be found in Appendix B.

4.4 Programming Language and Test Bed

The Client-pull and Server-push servlets were written in Java and implemented on the Tomcat Apache Web server. This was chosen due to budgetary constraints and the wealth of information available for developing servlets using Apache Tomcat. These applets were developed using the Eclipse IDE. Eclipse was chosen because it integrates easily with Apache Tomcat for the development of servlets.

An application was written in Java for testing the Client-pull and Server-push architectures. The Eclipse IDE played a part in choosing Java as the language for the test application. Eclipse projects can contain both servlets that can be deployed to Apache Tomcat directly from the IDE and java applications that can run on the development machine, both of which can be debugged in the IDE simultaneously. Java was also a language which I had limited exposure and I used this as an opportunity to gain experience in using it for both applications and servlets.

Java is an object oriented language. This allowed the wind farms, ETS devices, system load, and wind distributor to be simulated as instances of classes that modeled the elements. Multiple wind farms were different instances of the same wind farm class. The different vintages of residences were modeled as parameters in instances of an ETS device class. These instances were instantiated by a test application also written in java that allowed the number of residences based upon vintage to be defined by the user. Two versions of the test application were developed, one for the Client-pull architecture and one for the Server-push architecture. The Java source code for the servlets, test bed, and device classes can be found in Appendix C.

4.5 Summary

This chapter presented the implementation of a client pull and server push system for the distribution of intermittent wind energy based on the frameworks described in chapter 3. These implementations were subject to the limitations imposed by modeling a system with limited resources. The assumptions made and necessary modifications have been described. The registration of intermittent energy producers does not need to be limited to wind farms. Any device that produces intermittent energy could register with the system and allow its energy production to be distributed to storage units when there is an energy surplus. Storage units do not need to be limited to ETS systems. Any device capable of storing energy for later use could potentially register and be allocated energy using the implementations presented. A method to model the residential space heating needs for a jurisdiction has also been presented. The following chapter presents the results of simulating the Client-pull and Server-push implementations using the residential model for Summerside PEI.

Chapter 5 Simulation Results and Discussion

This chapter applies the Client-pull and Server-push architectures to a jurisdiction in a simulated environment to evaluate the performance of both systems over a full year analysis, and how each system responds to sudden changes in energy production. The jurisdiction being modeled is first discussed and the ETS systems required by the residences are identified. Next a short description of the test environment is given. Baseboard and evening recharge heating models used for comparison purposes.

During the evening hours the normal system load is lower because most people are sleeping and not working. It should be expected that wind-generated electricity will contribute to a net surplus condition most often during the evening hours when the electricity demand is low. This indicates that the Evening Recharge ETS system should use more wind generated energy then the baseboard system by focusing its energy usage to the period where wind generated energy is most often contributing to a net surplus energy condition. The Client-pull and Server-push systems should provide an improvement over the normal Evening Recharge ETS by optimizing the evening recharge to use as much wind energy as possible during both the day time and night time hours.

5.1 Jurisdiction Selection

This simulation uses Summerside PEI as the jurisdiction being evaluated. Summerside receives electricity from the 3 sources listed in Table 1(Larry Hughes, Greg Gaudet Interview. August 2010.). Each wind-farm supplies a varying amount of electricity, while the contract with NB Power provides a source of on-demand electricity that can be changed (i.e., increased or decreased, depending upon the wind supply) to meet Summerside's electricity requirements.

Table 1: Electricity Sources for Summerside Electric

Supplier	Type of electricity supply	Range
NB Power	On-demand, changeable	0 MW to winter-peak (about 23 MW)
West Cape wind farm	Variable	0 MW to 9 MW
Summerside wind farm	Variable	0 MW to 12 MW

If the wind generated electricity exceeds the on demand electricity requirements, the excess is sold back to NB power. The energy security context diagram showing the energy flow for Summerside PEI is shown in Figure 15.

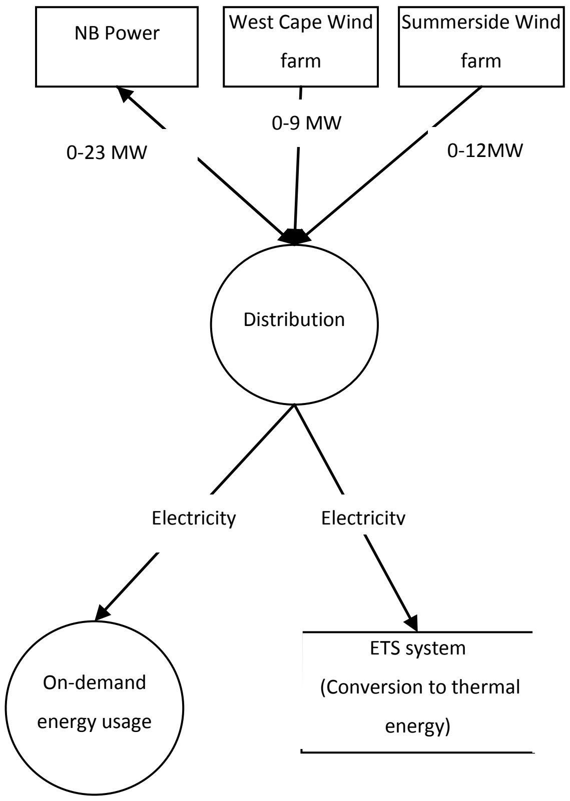


Figure 15 Summerside Energy Flow Diagram

5.1.1 Surplus Wind-electricity

Summerside collects power related metrics including the system load and the wind generation levels from the North Cape and Summerside wind farms on an hourly basis. This data has been shared with the Dalhousie Energy Research Group and has been used in this work.

Surplus wind-ectricity is defined in this work as the wind-electricity that is generated in access to the system load. The surplus wind-electricity can be determined from the Summerside data by summing the generation levels from the North Cape and Summerside wind farms then subtracting the system load. If the result is positive it represents surplus wind-electricity that is currently sold to NB Power at a discounted rate. A negative result requires the import of electricity from NB power to meet the shortfall between the demand and wind-electricity levels.

Figure 16 shows the Summerside electricity metrics for January 2011. The System Load shows a periodic usage curve corresponding to time of day. The Wind Generated-electricity is random in duration and intensity. The Surplus Wind-electricity has characteristics of both a random and periodic curve. Surplus Wind-electricity is targeted by the distribution systems to be directed to the ETS units rather than be sold to NB Power as an energy export. The energy available for redirection does not contribute significantly to the total energy used by the jurisdiction of Summerside during the heating season. The available Wind-electricity will vary from Jurisdiction to Jurisdiction and from year to year. When the system performance is evaluated the energy available for redirection will have to be taken into consideration.

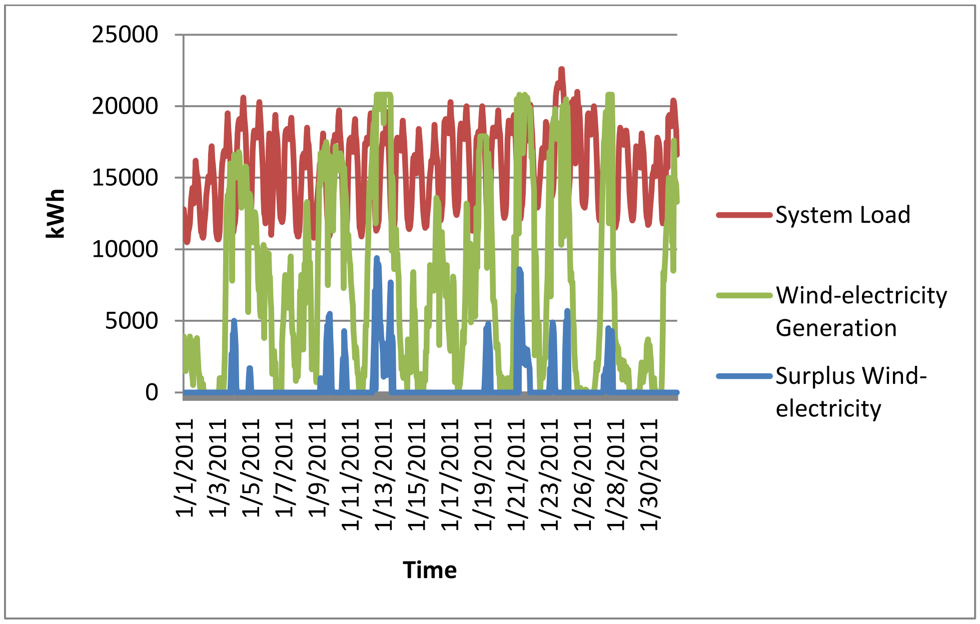


Figure 16 Surplus Wind-electricity (January 2011)

5.1.2 Residential Simulation

Section 3.3 presented a method to model the residences within a jurisdiction. This section applies this method to Summerside PEI to determine the parameters required for residential simulation

5.1.3 Degree Days

Environment Canada archives the temperature for each hour for selected locations in Canada. Using this data, Table 2 shows the total number of degree days for Summerside in 2008 for an internal target temperature of 19°C. The data has been centered on December which is the middle of a typical heating season.

Table 2 Degree Days by Month for Summerside 2008 [62]

Month	Degree hours
July	31.4
August	76.7
September	144.6
October	315.0
November	459.6
December	662.0
January	781.1
February	741.4
March	752.1
April	496.3
May	325.6
June	145.1

5.1.4 Summerside Residential Parameters

In Chapter 3, Equation (3.1) was developed to model the energy usage and Equation (3.2) for the space heating energy intensity of a residence. These equations are used along with the data from Table 1 and 2008 historical data from the Office of Energy Efficiency to develop a model for the residences of Summerside PEI shown in Table 3.

Table 3 Energy Intensity by Vintage

	Floor Space by Vintage (million m^2)	Housing Stock by Vintage (thousands)	Energy Use by Vintage (PJ)	Average home size (m^2)	Energy Usage (GJ/ m^2)	Energy Intensity (J/°C sec m^2)
Before 1946	1.5	13.4	0.9	110.1	0.600	1.408
1946–1960	0.3	2.7	0.2	128.8	0.667	1.565
1961–1977	0.8	6.5	0.3	125.6	0.375	0.880
1978–1983	0.6	4.7	0.2	122.1	0.333	0.782
1984–1995	1.9	14.6	0.5	132	0.263	0.618
1996–2000	0.8	5.6	0.2	137.6	0.250	0.587
2001–2005	1	7	0.2	147.5	0.200	0.469
2006–2008	0.6	4	0.1	146.1	0.167	0.391

The product of the energy intensity, the number of heating degrees, the interval of time, and the residence size gives an estimate of the energy used for space heating for that period of time.

A building's energy intensity can be used to determine the size of ETS unit needed to meet its space heating requirements: the higher the intensity, the larger the unit. The coldest recorded temperature for Summerside PEI was -29.9°C[63]. This value is used to calculate heating requirements of a residential structure by vintage shown in Table 4 by multiplying the energy intensity values from Table 3 by a full heating day with in internal temperature of 19°C and an external temperature of -29.9°C

Table 4 Extreme Heating Requirements

Vintage	Heating kWh
Before 1946	182
1946–1960	237
1961–1977	130
1978–1983	112
1984–1995	96
1996–2000	95
2001–2005	81
2006–2008	67

5.1.5 ETS Model Selection

The following section provides a method for determining the size of ETS model required to meet the extreme space heating needs of a residence. This requires knowledge of the charge and discharge cycle used for the ETS system. The storage capacity of the ETS system must be greater than or equal to the energy required for space heating during the discharge cycle. During the charge cycle, the ETS system can satisfy the space heating demands of the residence, and recharge for the following discharge cycle. This analysis uses a 16 hour discharge cycle and an 8 hour recharge cycle. For a 16 hour discharge, the storage requirements by vintage are given in Table 5.

Table 5 ETS Storage Requirements Under Extreme Conditions

Vintage	Heating kWh
Before 1946	122
1946–1960	158
1961–1977	87
1978–1983	75
1984–1995	64
1996–2000	63
2001–2005	52
2006–2008	45

The model of ETS required by vintage is found by selecting one with a storage capacity equal to or greater than the values in Table 5. This analysis uses Comfort-Plus Forced Air ETS systems built by the Steffes Corporation. These ETS systems can implement different charging input circuits allowing the ETS to recharge faster. The technical specifications sheets for the Steffes Comfort-Plus Forced-Air ETS models 4120, 4130, and 4140 give the maximum BTU loss per hour that can be maintained for different recharge-cycles hours in BTU and the selected recharge circuit, and storage capacity. This data is summarized below in Table 6.

Table 6 ETS System Parameters [64]

Model	4120			4130		4140	
Storage Capacity (kWh)	120			180		240	
Charging input (kW)	14	19.2	24.8	28.8	37.2	38.4	45.6
Maximum Maintainable heat loss							
8 Consecutive charge hours (kWh/hr)	5.98	8.21	10.02	12.31	14.42	16.41	19.23
12 Consecutive charge hours (kWh /hr)	8.97	12.31	13.35	18.46	19.23	24.62	25.64
18 Consecutive charge hours (kWh /hr)	13.46	18.46	23.85	27.69	35.77	36.92	38.47

From Table 5 and Table 6, a model 4120 ETS system has sufficient storage capacity to meet the space heating requirements for residences built after 1960. Residences built before 1960 require between 122kWh and 158kWh of storage capacity. This can be met by a model 4130.

The required charging input can be determined by taking the maximum possible discharge over a 24-hour period and dividing this by the length of the recharge cycle. Assuming an eight-hour recharge, the required charging input for each residence by vintage is shown in Table 7.

Table 7 Minimum Required Recharge Circuit

Vintage	Total discharge over 24 hours (kWh)	Required recharge circuit (kW)
Before 1946	182	23
1946–1960	237	30
1961–1977	130	16
1978–1983	112	14
1984–1995	96	12
1996–2000	95	12
2001–2005	81	10
2006–2008	67	8

This represents the minimum recharge circuit by vintage. The recharge circuit selected must be greater than or equal to this value but in this work the objective is to minimize the export of wind energy which can be most effectively accomplished by maximizing the ETS's capacity to use wind energy when it is available. To maximize this potential, all ETS units in this thesis will use the maximum recharge circuit defined for the required model.

5.2 Space Heating Simulations

These simulations evaluate the performance of the Client-pull and Server-push space architectures using Summerside PEI for the Jurisdiction under test. The simulations use two other forms of space heating for comparison purposes, Baseboard and ETS units which use a traditional Evening recharge.

5.2.1.1 Baseboard

Baseboard heating was discussed in section 2.4. The availability of wind power and the space heating needs of a residence are two independent processes. If there is wind generated electricity available when the space heating demand is present, the simulations assume the power has been used for space heating. If there is no wind generated electricity available, the needed energy is imported using the grid intertie.

5.2.1.2 Evening recharge

Some jurisdictions use the overnight hours (23:00 to 7:00) which are typically periods of low electrical demand to recharge the ETS systems[11]. The recharge rate in this model is calculated by dividing the available energy storage by the time remaining in the eight-hour recharge cycle plus the discharge rate over the previous hour as shown in Equation (5.1).

$$\boldsymbol{Rate_{Hour} = \frac{MaxChargeLevel - ChargeLevel_{Hour-1}}{Hours\ remaining\ in\ recharge} + Rate_{Hour-1}} \qquad \textbf{(5.1)}$$

5.2.1.3 Client-pull with Evening Recharge

This is similar to the Evening Recharge implementation in that it will ensure that the ETS is fully recharged during the overnight hours meet the space heating needs of the residence until the next recharge cycle. In addition to this minimum recharge rate during the evening hours, the Client-Pull architecture as described in 4.2 for the distribution of surplus wind generated energy to ETS devices is implemented.

5.2.1.4 Server-push with Evening Recharge

This model also uses the Evening Recharge implementation to ensure that the ETS is fully recharged during the evening hours to provide the space heating needs of the residence until the next recharge cycle. In addition to this minimum recharge rate during the evening hours, the Server-push architecture as described in 4.3 is implemented to distribute the surplus wind generated energy when it is available.

5.3 Multi-residence Simulation

This simulation evaluates a total of 60 residences. The number of residences was limited to 60 due to the computer power available; each heating season simulated took 24 hours to complete. Table 3 contains the number of residences by vintage. The distribution of residences used in this simulation matches the distribution of residences by vintage for Summerside PEI as shown in Table 8.

Table 8 Simulated Residences by Vintage

Vintage	Housing Stock by Vintage (thousands)	Residences simulated
Before 1946	13.4	13
1946–1960	2.7	3
1961–1977	6.5	7
1978–1983	4.7	5
1984–1995	14.6	15
1996–2000	5.6	6
2001–2005	7	7
2006–2008	4	4

5.4 Full Heating Season Simulation

This simulation evaluates the energy source used for space heating from September 2010 to May 2011 for the four different space heating models. Wind generation and load information was not available for May 2011 so data from May 2010 was used in its place.

Figure 17 to Figure 20 show the energy used by month for baseboard heating, evening only recharge, Client-pull with evening recharge, and Server-push with evening recharge respectively. The height of each bar represents the total energy used by the residences with the blue portion representing the energy that has been obtained from the wind. The red portion represents the energy obtained from outside Summerside.

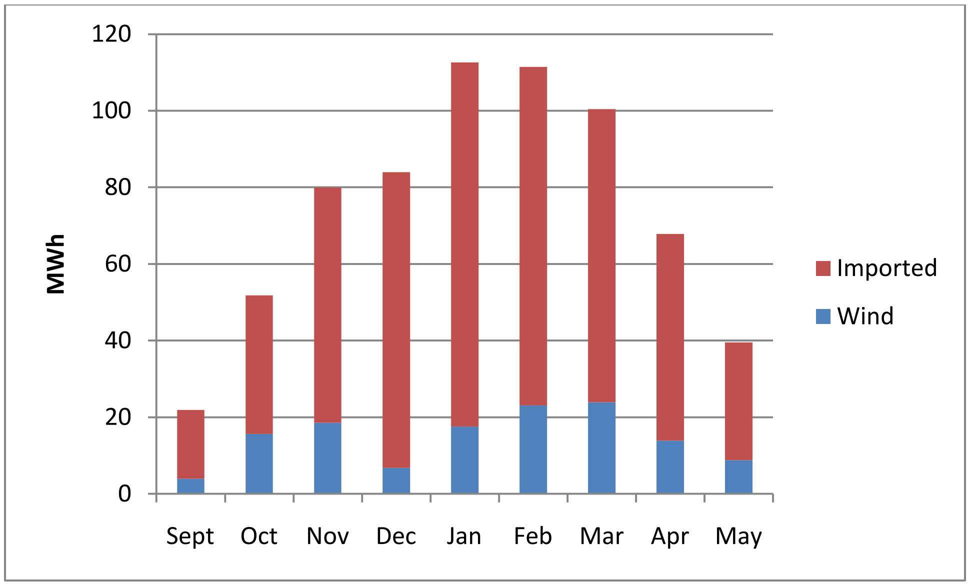


Figure 17 Baseboard Heating

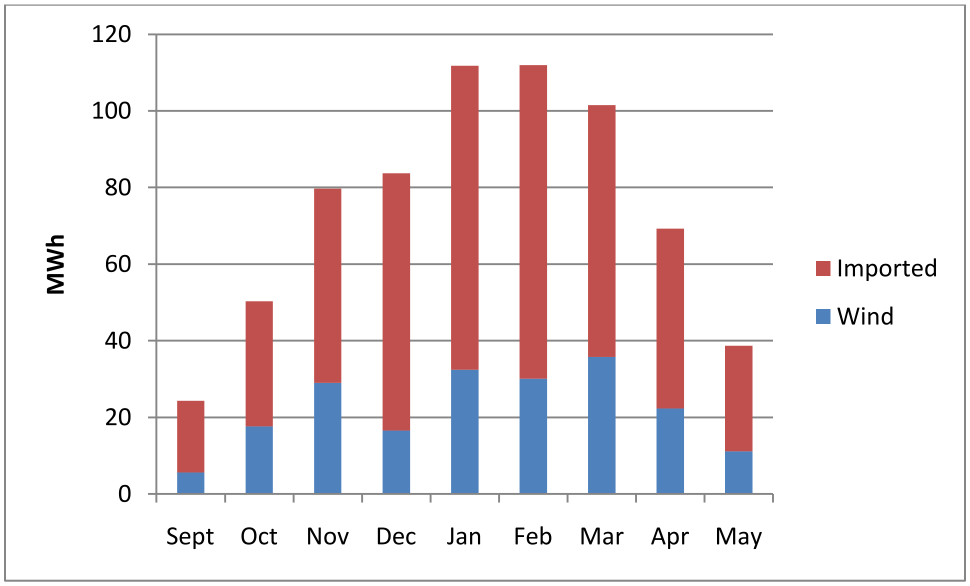


Figure 18 Evening Only Recharge

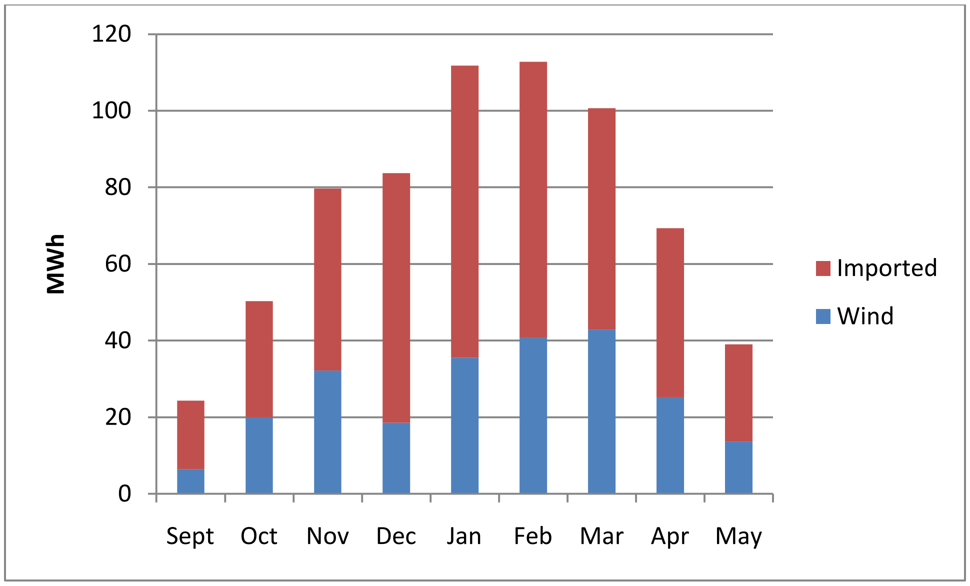


Figure 19 Client-pull with Evening Recharge

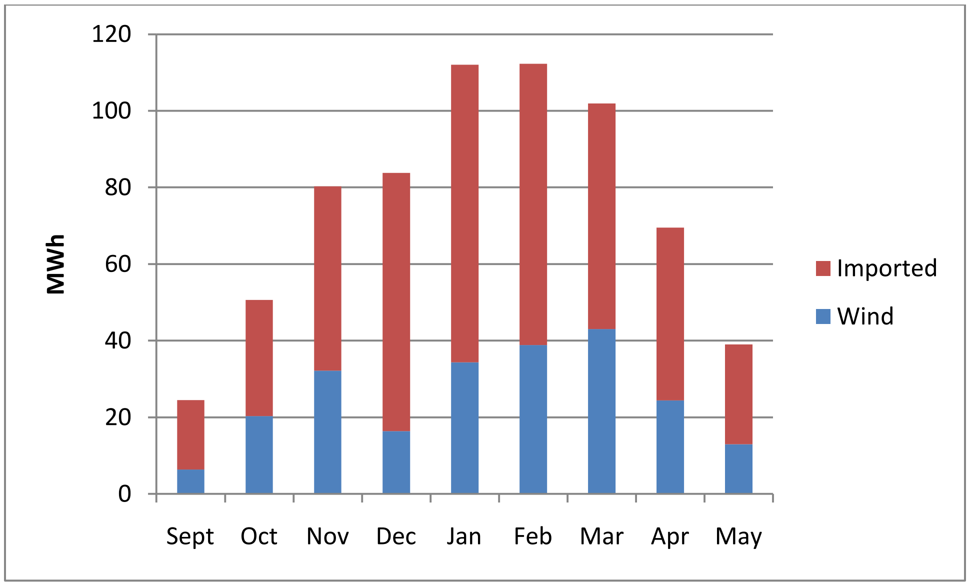


Figure 20 Server-push with Evening Recharge

Although the stacked bar graphs show how variations in the monthly use of wind generated electricity for space heating, they do not allow for an easy comparison of the different space heating models. To help clarify the seasonal heating data, the heating season totals by energy source are shown in Table 9 as percentages.

Table 9 Space Heating Energy Totals by Source

	Baseboard	Evening recharge	Client-pull	Server-push
Wind	19.7%	29.9%	35.0%	34.0%
Imported	80.3%	70.1%	65.0%	66.0%

The Client-pull and Server-push implementations offer an advantage in utilizing domestic wind generated electricity over the existing evening recharge model for ETS systems and base board heating. The Client-pull implementation has a slight advantage over Server-push which can be attributed to the fact that the Client-pull model can set an ETS recharge rate to a fraction of the ETS unit maximum therefore matching the energy demand with the energy supply. The Server-push system must use the reported maximum recharge rates of the ETS systems and match as closely as possible the demand with the existing supply of wind generated energy. These figures are encouraging given the limited availability of surplus wind-electricity as shown in Figure 16.

5.5 Response Time Analysis

This analysis explores how quickly each system can respond to a change in the production of wind energy. In the analysis, the initial state is where there is a surplus of wind energy being produced, but it can be completely absorbed by the ETS units within the system. Next an abrupt change in the production of wind energy is introduced which is above the total capacity that can be absorbed by the ETS units. The simulation tracks the total amount of energy being used by the ETS units over time and the amount of wind energy being produced.

In the Client Pull implementation, each ETS unit polls the server on an interval to determine its operating state. The expected behavior of the system to an abrupt change in wind generation is a nearly linear increase in the total energy being used by the ETS units until the maximum amount that can be absorbed is reached. The interval for the system to reach this upper limit should be less than or equal to the interval period used by the ETS units to poll the server. In this analysis, the interval was set between 30 and 60 seconds. A system of 60 ETS units was used for the analysis. Only 60 ETS units were used due to the limited computing power available for the simulation.

The results of how the Client Pull system responds to an abrupt change in the production of wind energy is shown below in Figure 20.

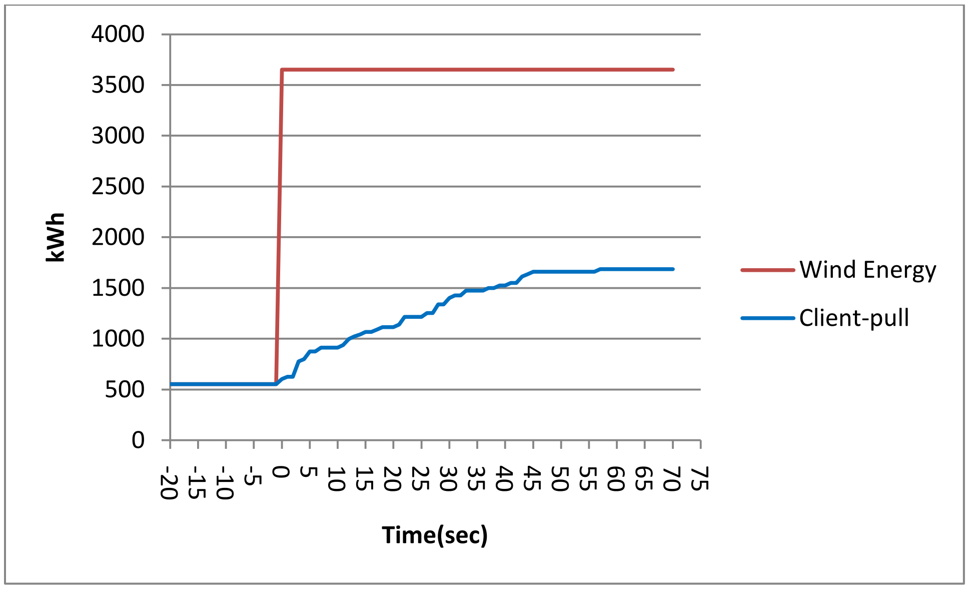


Figure 21 Client-pull Response Time Analysis

Before the transition in the generation of wind energy, all of the available wind energy was absorbed by the ETS units. After the transition as ETS units polled the server, additional wind energy was distributed to the ETS units over a period of approximately 1 minute, until all ETS units were recharging at their maximum rate. This observed behavior matches the expected behavior. This response time could be reduced by increasing the frequency that the ETS units request state information from the server, but this would increase network traffic and also place additional load on the server.

The Server-push implementation was tested against the same energy production profile to evaluate its response time. Before the transition, it is expected that nearly all (or slightly more) of the wind energy will be allocated to the ETS units. After the transition, a new distribution packet will be generated and sent to all ETS units that should allocate the maximum amount of wind energy almost immediately. A system of 60 ETS units was used for the analysis. Only 60 ETS units were used due to the limited computing power available for the simulation.

The results of how the Server Push system responds to an abrupt change in the production of wind energy are shown below in Figure 21.

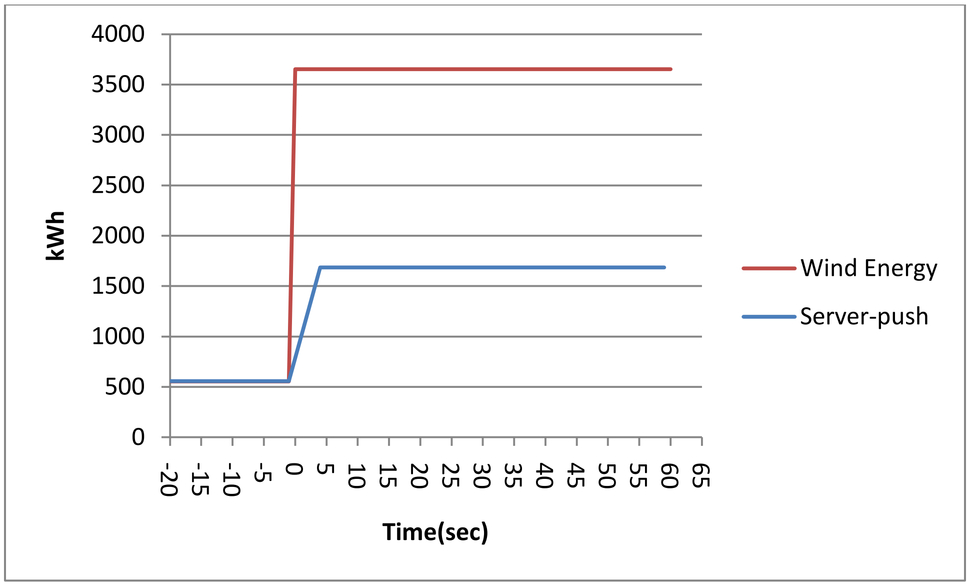


Figure 22 Server-push Response Time Analysis

Before the transition, the Server Push system was allocating slightly more energy then was being produced. This is because the absolute difference between the generated and used levels was minimized by turning on an additional unit. After the transition, the system responded within 4 seconds. This delay was due to the processing time required to simulate the ETS units, send the UDP packet, and handle the responses from the ETS units as they processed the UDP packets and reported back to the registration server, all being done on the same PC. The responsiveness of the Server-push system makes it ideal for the implementation of a load balancing mechanism when used with the surplus storage capacity discussed in 5.4.

5.6 24 Hour Analysis

The graphs in Figure 22 and Figure 23 show the energy allocation for a single residence of each vintage over a 24 hour period from noon on Feb 26 2011 to noon on Feb 27 2011 in the Client-pull and Server-push architectures. This period was selected because it has a period during the day time hours where wind energy is being produced in surplus and periods where there is no wind power available. Also during the evening hours there is a period where wind is also generated in surplus. The wind energy uses the right axis while the residences use the left.

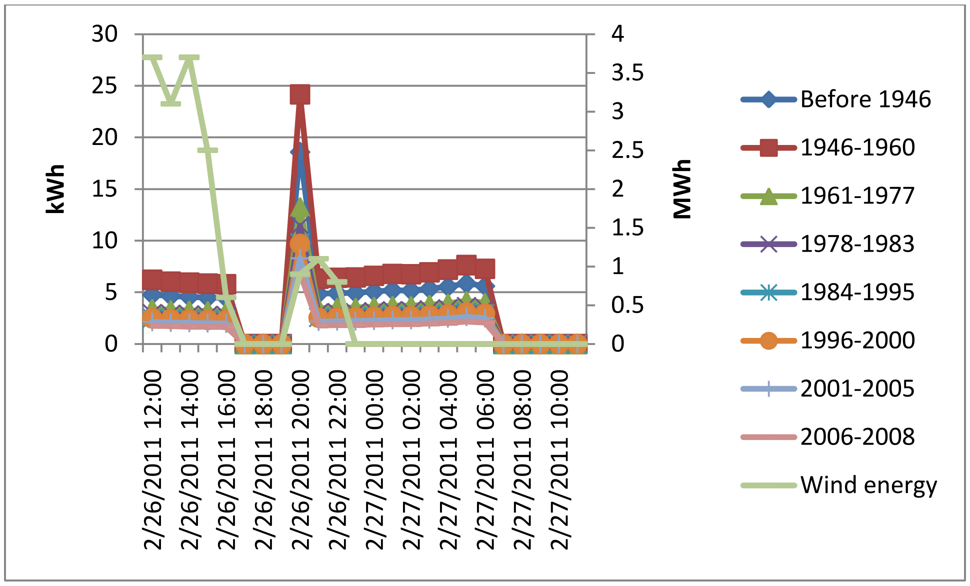


Figure 23 Client-pull 24 Hour Analysis

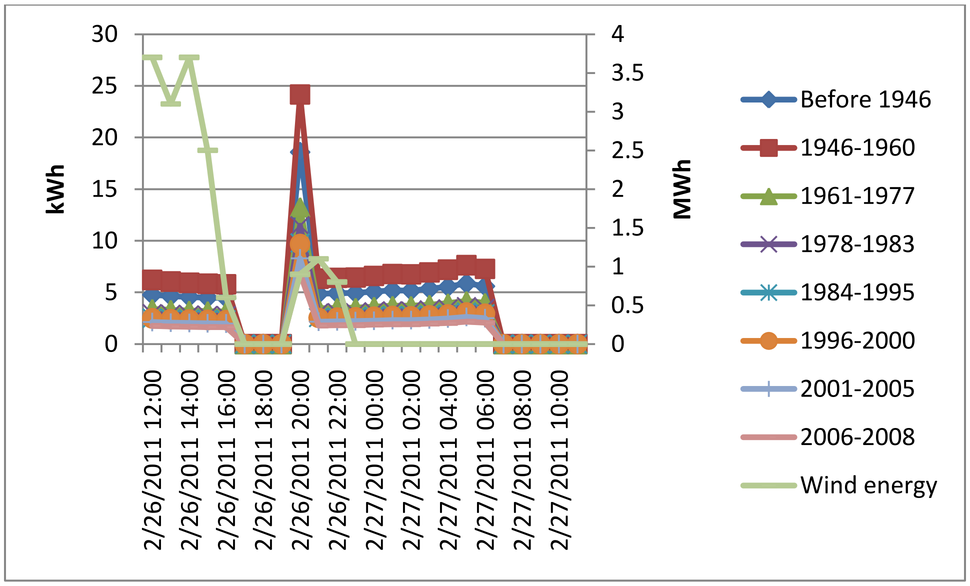


Figure 24 Server-push 24 Hour Analysis

These two graphs are identical. Since there is more wind energy available then can be used, all ETS devices are recharging at their maximum rate. It is interesting to note that not all of the wind energy is being used during the day time hours. This is a result of each ETS unit being fully charged and simply replenishing the hourly discharged energy needed for space heating and there not being enough ETS devices to absorb all the available wind energy. After the wind energy is no longer available at 17:00 all ETS devices stop recharging but begin again at 20:00

(three hours before the evening recharge cycle) and quickly recharge then continue to recharge the hourly discharge until 7:00 the following morning.

5.7 Architecture Complexity, Bandwidth and Growth

The Server-push architecture implementation has a more complex design. Its firmware design must implement a client for connecting to a VPN, and also implement the UDP protocol. The network on which the system is deployed must also support sending UDP packets through routers and switches. This is in addition to the HTTP protocol that is implemented by both systems.

The increased complexity of the Server-push architecture does however provide an advantage in bandwidth requirements. The Client-pull system is continually polling the server for system updates. As the number of ETS units being supported by the system, the bandwidth requirements will increase. For the Server-push architecture, only one packet which is then distributed to each storage device is required and only when a change in power distribution is required. A Server-push architecture should support a larger install base before bandwidth bottlenecks become a significant issue. This simulation used in this work could only implement a limited number of storage devices due to the limited hardware and processor power available.

5.8 Summary

This chapter presented the implementations of the Client Pull and Server Push systems, and an analysis of the ability of each to effectively use wind energy to minimize energy exports. It also presented a response time analysis of each system. Each system was seen to have a strength and a weakness. The Client-pull system utilized of intermittent wind electricity more effectively then the Server-push system. The Server-push system clearly outperformed the Client-pull system in responding to changes wind energy production levels.

These observations show that the implementation chosen for a system to distribute wind generated energy for space heating will depend largely on the goals of the system designers. An optimum system could also be designed that implemented aspects of both systems by using broadcast messages to turn units on or off quickly as seen in the Server-push implementation and also use the polling technique seen in the Client-pull system to optimize the use of wind generated energy.

The emergency recharge database table discussed in section 3.1.1 was intended to record when an ETS device became completely depleted. This database table contained no records after the simulations. This indicates that the ETS units selected did not become fully depleted. The unused storage capacity could be used to increase the use of intermittent wind energy by developing an algorithm that recharged an ETS system during the evening to a predefined level that is a fraction of its maximum storage, increasing the storage capacity to be used by intermittent wind energy generated in the day time hours. Another option would be to use the additional storage as a mechanism for load balancing.

Chapter 6 Concluding Remarks

The objective of this thesis was to develop a system to maximize the distribution and utilization of intermittent renewable wind energy within a jurisdiction by employing the storage capacity of ETS units. Two approaches were considered to meet this goal, a Client-pull architecture based upon using the HTTP protocol where the ETS units would poll a server for operating instructions and a Server-push architecture where a server would send operating instructions to all ETS devices simultaneously using a UDP broadcast message to the ETS units though a VPN. These implementations were chosen because they allowed the use of existing proven technologies implemented to satisfy the communication requirements of the energy distribution system.

6.1 Results summary

Section 1.5 outlined the comparison criteria for the Client-pull and Server-push approaches. The discussion below presents each of these criteria with an analysis of the results.

6.1.1 Wind-electricity Utilization

Both the Client-pull and Server-push architectures differed by only 1% from each other but did improve the wind-electricity utilization over the traditional ETS recharge. The results were not conclusive enough to support one being superior over the other and may be due to the limited amount of surplus wind-electricity available to be distributed as shown in Figure 16. Further testing using wind data with more surplus wind-electricity may provide a clearer picture of the differences between the two systems.

6.1.2 Short Term Latency

The Server-push architecture had a significantly faster response time (4 seconds) then the Client-pull (60 seconds). The actual response time in a distributed network was not evaluated due to the limitations of the simulation environment being a single PC. The faster response time of the Server-push does come with a cost. The Server-push implementation does not provide a mechanism for the server to know that the client received the sent command.

6.1.3 Single Day Performance

During the day selected for analysis (Feb 26-27 2011), both the Client-pull and Server-push architectures distributed wind-electricity identically by accurately distributing the surplus wind-electricity to ETS units.

6.1.4 Complexity

Both the Client-pull and Server-push architectures used Web SHTTP protocols for registration and status update. The Client-pull used the same Web SHTTP protocol for communication with the ETS devices. The Server-push architecture implementation required the additional sending of UDP datagrams over a VPN for passing operating instructions to the ETS units. This makes the Server-push architecture the more complex system.

6.2 Recommended Implementation

The selection of one system over the other will depend on the objectives of the jurisdiction. The simplicity of the Client-pull architecture makes it an easier system to implement in a jurisdiction then the Server-push which makes it the natural first step in developing a surplus wind-electricity distribution system. If the goals of the jurisdiction are to be able to quickly pass operating instructions to ETS units for rapid response, then the additional complexity of the Server-push architecture may be justified.

6.3 Limitations

This work was done using a simulated environment with all system software running on a single general purpose PC. This limited the depth to which the system could be tested as follows

- The VPN presented in 3.2.1 was not implemented due to the additional latency and complexity of implementing it on a single PC.
- Implementation on a single PC restricted the use of a UDP packet for sending operating instructions in the Server-push architecture because only one process can monitor a port at a time. A single process UDP monitoring scheme was required which passed the instruction to the simulated ETS devices allowed this limitation to be mitigated, but not eliminated.

- For simplicity, the distribution algorithm in the Server-push architecture always started with the lowest push id. This does not allow all ETS units equal access to the surplus wind energy. A scheme that used the ETS units in a form of circular queue, turning on units at the start and turning them off from the end would be a fairer system, but would be more complex to design.
- A single PC was not capable of simulating all of the system elements as independent threads, with each system device connecting and reporting independently. This would have been a closer approximation to what would happen in a real world environment. System elements needed to be handled serially to ensure that each was processed.

Many of these limitations could be removed by having multiple PC's for the server-side implementation and a network of PC's to simulate the ETS units running within a VPN. Another limiting factor in this work was the low level of available surplus wind-electricity as shown in Figure 16. This obscured the differences of the two systems in evaluating wind-electricity utilization over a full heating season.

6.4 Future Work

Some interesting observations were made during testing that could lead to further work in this area. These areas include:

Load Balancing: All ETS units reacted to the Server-push broadcast message in two seconds. This responsiveness of the Server-push architecture could be exploited to use ETS units in load balancing. Network latency which may be an issue for load balancing was not addressed in this work due to the simulation occurring on a single PC.

Thermal Storage Optimization: By using a worst case for determining the thermal storage requirements of an ETS within a residence there was thermal storage potential that was underutilized. A recharging algorithm that set an evening recharge thermal storage maximum as a fraction of the maximum possible by the ETS device based on the expected temperature for the following day would provide additional storage potential in the ETS unit to capitalize on more intermittent wind energy during the day time hours if it were to become available and potentially reduce the level of non-renewable energy required to recharge the ETS unit during the next evening recharge cycle.

Circular Distribution Queue: Designing a circular distribution queue for the Server-push architecture would allow for a fairer distribution of wind energy to ETS devices.

Multi-framework Implementation: Blending the best aspects of each system (load matching the energy supply in the Client-pull and low latency of the Server-push) into one implementation.

References

[1] Susan Solomona and et al., "Irreversible climate chage due to carbon dioxide emissions," *Proceedings of the National Academy of Sciences of the United States of America*, pp. 1704-1709, January 2009.

[2] Scott C. Doney, Victoria J. Fabry, and Richard A. Feely, "Ocean Acidification: The Other CO2 Problem," *Annual Review of Marine Science*, pp. 169-192, 2009.

[3] International Energy Agency, "World Energy Outlook 2008," 2008.

[4] International Energy Agency, "World Energy Outlook 2010," 2010.

[5] C. Menz Fredric and Vachon Stephan, "The effectiveness of different policy regimes for promoting wind power: Experiences from the states," *Energy Policy*, pp. 1786-1796, September 2006.

[6] World Wind Energy Association, "World Wind Energy report 2009," 2010.

[7] Georg Marsh, "From Intermittent to variable;Can we manage the wind?," *Renewable energy focus*, pp. 42-47, September/October 2009.

[8] Tom Steffes, Phone conversation with Tom Steffes regarding ETS units and control, Oct 4, 2010.

[9] South Kentucky Rural Electric Cooperative Corperation. South Kentucky Rural Electric Cooperative Corperation. [Online]. http://www.skrecc.com/ets.htm [Accessed: 28 October 2010].

[10] Steffes Corperation, Comfort Plus Hydronic Furnace Specifications for 240V units.

[11] Nova Scotia Power. (2010) Electrical Thermal Storage. [Online]. http://www.nspower.ca/en/home/residential/homeheatingproducts/electricalthermalstorage/default.aspx [Accessed: 10 December 2010].

[12] Larry Hughes, "Meeting residential space heating demand with wind-generated electricity," *Renewable Energy*, vol. 35, no. 8, pp. 1765-1772 , August 2010.

[13] Larry Hughes, A framework for determining and improving the relative energy security of a jurisdiction's energy system, 2011.

[14] Internet Working Group. (1999, June) Hypertext Transfer Protocol -- HTTP/1.1. [Online]. http://www.ietf.org/rfc/rfc2616.txt [Accessed: 22 Jan 2011].

[15] Postel J. (1980, August) User Datagram Protocol. [Online]. http://tools.ietf.org/html/rfc768 [Accessed: 29 August 2011].

[16] NEMA. (2001) NEMA The Association of Electrical and medial Imaging Equipment manufacturers. [Online]. http://www.nema.org/gov/energy/smartgrid/whatIsSmartGrid.cfm [Accessed: 22 March 2011].

[17] World Wind Energy Association, "World Wind Energy Report 2009," Bonn, 2009.

[18] Archer Cristina L. and Jacobson Mark Z., "Evaluation of global wind power," *Journal of Geophysical Research*, vol. 110, 2005.

[19] Hannes Weigt, "Germany's wind energy: The potential for fossil capacity replacement and cost saving," *Applied Energy*, vol. 86, no. 10, October 2009.

[20] Nicolas Boccard, "Capacity factor of wind power realized values vs. estimates," *Energy Policy*, vol. 37, no. 7, 2009.

[21] M.H. Albadi and E.F. El-Saadany, Overview of wind power intermittency impacts on power systems, 2010.

[22] Kari Larsen, "Smart grids – A Smart Idea?," *renewable energy focus*, pp. 62-67, 2009.

[23] U.S Department of Energy, "The Smart Grid: An Introduction," 2008.

[24] Electric Power Research Institute, "Report to NIST on the Smart Grid Interoperability Standards Roadmap," 2010.

[25] 110th Congress, Energy Independence and Security Act of 2007, 2007.

[26] National Institute of Standards and Technology, NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 1.0, 2010, January.

[27] NSIT. (December, 23) NIST & the Smart Grid. [Online]. http://www.nist.gov/smartgrid/nistandsmartgrid.cfm [Accessed: 17 January 2011].

[28] Rick Merritt. (2010, May) EE Times News and Analysis. [Online]. http://www.eetimes.com/electronics-news/4199756/Smart-grid-standards-expected-by-mid-2011 [Accessed: 29 September 2010].

[29] David Beauvais, Smart Grid- Activities in Canada, May 3, 2010.

[30] Natural Resources Canada. (2010, July) Natural Resources Canada. [Online]. http://www.nrcan-rncan.gc.ca/media/newcom/2010/201058-eng.php

[31] Cowessess First Nation. Cowessess First Nation Wind Turbine Battery Project. [Online]. http://www.cowessessfn.com/our-departments/economic-development/wind-energy-project [Accessed: 18 November 2010].

[32] HM Government, "The UK Low Carbon," London, National strategy for climate and energy 2009.

[33] Elizabeth Ingram, "Pumped storage development activity snapshots.," *Hydro Review*, pp. 12-25, December 2009.

[34] S Van der Linden, "Bulk energy storage potential in the USA, current developments and future prospects.," *Energy*, vol. 15, 2008.

[35] Jasna Tomic and Willett Kempton, "Using fleets of electric-drive vehicles for grid support," *Journal of Power Sources*, vol. 168, no. 2, 2007.

[36] BC Hydro. (2010, October20) BC Hydro. [Online]. http://www.bchydro.com/etc/medialib/internet/documents/Power_Smart_FACT_sheets/FACTS_Electric_Baseboard_Heaters.Par.0001.File.FACTS_electric_baseboard_heaters.pdf

[37] Larry Hughes, "Meeting residential space heating demand with wind-generated electricity ," *Renewable Energy*, vol. 35, no. 8, pp. 1765-1772, August 2010.

[38] Steffes Corperation. ETS Off Peak Headting ETS Systems. [Online]. http://www.steffes.com/off-peak-heating/hydronic-furnace.html

[39] Nova Scotia Power. (2010) Nova Scotia Power Website. [Online]. http://www.nspower.ca/en/home/residential/homeheatingproducts/electricalthermalstorage/timeofdayrates.aspx [Accessed: 30 September 2010].

[40] International Organization for Standardization, ISO/IEC 7498-1:1994, 1994.

[41] J. Postel and J. Reynolds. faws.org. [Online]. http://www.faqs.org/rfcs/rfc854.html [Accessed: 8 October 2010].

[42] Internet Engineering Task Force. The Secure Shell (SSH) Authentication Protocol. [Online]. http://tools.ietf.org/html/rfc4252 [Accessed: 8 October 2010].

[43] W3C Working Group. (2010) WC3. [Online]. http://www.w3.org/TR/ws-gloss/ [Accessed: 8 October 2010].

[44] W3C Working Group. (2010) wc3. [Online]. http://www.w3.org/TR/ws-arch/#relwwwrest [Accessed: 8 October 2010].

[45] Roy Thomas, Architectural Styles and the Design of Network-based Software Architectures, 2000.

[46] (2010, October) Wikipedia. [Online]. http://en.wikipedia.org/wiki/Representational_State_Transfer#cite_note-Fielding-Ch5-0 [Accessed: 8 October 2010].

[47] Network Working Group. (1999, June) Hypertext Transfer Protocol -- HTTP/1.1. [Online]. http://www.w3.org/Protocols/rfc2616/rfc2616.html [Accessed: 26 April 2011].

[48] WC3. (2010) WC3. [Online]. http://www.w3.org/TR/soap12-part1/ [Accessed: 10 October 2010].

[49] D. Crockford. (2006, July) The application/json Media Type for JavaScript Object Notation (JSON). [Online]. http://www.ietf.org/rfc/rfc4627.txt?number=4627 [Accessed: 17 January 2011].

[50] D. Negru, A. Mehaoua, Y. Hadjadj-aoul, and C. Berthelot, "Dynamic bandwidth allocation for efficient support of concurrent digital TV and IP multicast services in DVB-T networks," *Science Direct*, vol. 29, no. 6, 2006.

[51] B. Quinn, Celox Networks, and K. Almeroth. (2001) IP Multicast Applications:Challenges and Solutions. [Online]. http://tools.ietf.org/html/rfc3170 [Accessed: 11 October 2010].

[52] Environment Canada. (2011, May) National Climate Data and Information Archive. [Online]. http://climate.weatheroffice.gc.ca/prods_servs/glossary_e.html [Accessed: 18 Feburary 2011].

[53] Natural Resources Canada. (2005, April) Office of Energy Efficiency. [Online]. http://oee.nrcan.gc.ca/corporate/statistics/neud/dpa/comprehensive_tables/ [Accessed: 13 July 2011].

[54] Oracle. Oracle web site. [Online]. 2011 [Accessed: 27 July 2011].

[55] Microsoft Inc. (2011) IIS. [Online]. http://www.iis.net/overview [Accessed: 27 July 2011].

[56] IBM. (2010) IBM Software. [Online]. http://www-01.ibm.com/software/websphere/?lnk=mhpr# [Accessed: 27 July 2011].

[57] The Apache Software Foundation. (2011) Apache Tomcat. [Online]. http://tomcat.apache.org/ [Accessed: 27 July 2011].

[58] Oracle. Oracle web site. [Online]. http://www.oracle.com/us/products/middleware/application-server/oracle-glassfish-server/index.html [Accessed: 27 July 2011].

[59] IBM. (2010) IBM web page. [Online]. http://www-01.ibm.com/software/webservers/appserv/community/ [Accessed: 27 July 2011].

[60] Mort Bay Consulting. (2011) Codehaus web site. [Online]. http://jetty.codehaus.org/jetty/ [Accessed: 27 July 2011].

[61] Redhat. JBoss web site. [Online]. http://www.jboss.org/overview.html [Accessed: 27 July 2011].

[62] Environment Canada. (2011, May) National Climate Data and Information Archive. [Online]. http://www.climate.weatheroffice.gc.ca/climateData/hourlydata_e.html?Prov=PE&StationID=10800&timeframe=1&cmdB2=Go&cmdB1=Go&Month=12&Day=1&Year=2008&cmdB2=Go [Accessed: 25 May 2011].

[63] Environment Canada. (2011, May) Canadian Climate Normals 1971-2000. [Online]. http://www.climate.weatheroffice.gc.ca/climate_normals/results_e.html?stnID=6547&lang=e&dCode=1&province=PEI&provBut=Search&month1=0&month2=12 [Accessed: 11 June 2011].

[64] STEFFES Corperation, Technical Data Sheet, Comfort Plus Forced Air Electric Thermal Storage Heating System Models 4120, 4130, 4140.

Appendix A: Database Schema

Authentication table

Table name: auth_table

Description: This table is used by the Web Services to authenticate the device connecting to the system.

Field Name	Datatype	length	description
deviceid	char	32	a 32 or less character string that uniquely identifies this device
password	char	32	a 32 or less character string used for authentication

Storage units

Table name: storage_units

Description: This table maintains the list of devices that have registered with the system and the energy that is currently allocated to that unit.

Field Name	Datatype	length	description
Deviceid	char	32	a 32 or less character string that uniquely identifies this device
Energy_ allocation	double	1	The current power allocated to this storage device (kW)

Energy Production

Table name: producers

Description: This table holds the most recent output level for each intermittent power provider in the system.

Field Name	Datatype	length	description
Deviceid	char	32	a 32 or less character string that uniquely identifies this device
Output_kw	double	1	The current power output of the intermittent producer in kW

Emergency Storage

Table name: emergency_storage

Description: This table records when an ETS unit runs out of thermal storage and requires immediate energy from the grid to meet the space heating needs of the residence. This was added to ensure that the ETS units selected for a residence were not undersized and aid in the analysis of the system data. Ideally this table will have no records.

Field Name	Datatype	length	description
aDateTime	Longint	64 bit	The time the event occurred in msec since Jan1, 1970
Deviceid	char	32	a 32 or less character string that uniquely identifies this device
ResSize	double	1	The size of the residence in m^3
EmergencyRecharge	double	1	The energy taken from the grid in kWh to meet the current hours space heating needs
EnergyIntensity	double	1	Factor used for determining the energy requirements of the residence
Output_kw	double	1	The current power output of the intermittent producer in kW

System Loads

Table name:System_loads

Description:This table records the system loads for the district for the current hour. This is needed to determine if and when there is surplus wind energy.

Field Name	Datatype	length	description
Deviceid	char	32	a 32 or less character string that uniquely identifies this load
Electrical_load	double	1	The current power consumption of the load in kWh

System data

Table name: wind_data_table

Description: This table contains information parsed from the chronological data provided by Summerside PEI. This includes the time, system load, external temperature and the generation levels for two wind farms.

Field Name	Datatype	length	description
aDateString	char	32	a 32 or less character string that provides a text version of the record time.
aDateTime	LongInt	64bit	The record time in msec since Jan 1 1970
Powerload	double	1	The energy used by the system for the current hour in MW
WestCape	double	1	Hourly power output from the WestCape wind farm in kW
Summerside	double	1	Hourly power output from the WestCape wind farm in kW
Temperature	double	1	The external temperature for the current hour in °C

Energy statistics

Table name: energy_stats

Description: This is a summary table that records on an hourly basis system performance metrics used for analysis.

Field Name	Datatype	length	description
aDateTime	LongInt	64bit	The record time in msec since Jan 1 1970
WindEnergyAvailable	double	1	This is the net wind energy available from the wind farms after the system loads have been satisified.
EnergyForSpaceHeating	double	1	The total energy used for space heating by all residences for the current hour.
ImportedEnergyForSpaceHeating	double	1	The energy that must be imported for space heating due to insufficient wind energy.
ExportedWindEneryg	double	1	The net excess energy that can be exported after the system loads and space heating requirements for all residences has been met.
EveningHours	Integer	1	A field used to simplify analysis. 1= 23:00 to 7:00, 7:00 to 23:00.
aDateString	char	32	a 32 or less character string that provides a text version of the record time.
RecordIndex	Integer	1	Used to record the order in which records were added.

Energy Usage

Table name: energy_usage

Description: This table records when the energy allocated to any storage device in the system changes. It provides a chronological history of the systems allocation. There is a significant amount of data in this table related to allocating data. Not all the fields are populated in the server push implementation due to the time required to perform the querying and insertion of the additional data.

Field Name	Datatype	length	description
aDateTime	LongInt	64bit	The record time in msec since Jan 1 1970
Deviceid	char	32	a 32 or less character string that uniquely identifies this device
EnergyProduced	double	1	Total intermittent energy produced.
SystemLoad	double	1	The system load when the record was written
EnergyAvailable	double	1	The net energy available for distribution to storage devices.
EnergyAllocatedBefore	Double	1	The total energy allocated before re-allocation.
EnergyAllocatedToUnit	Double	1	The energy allocated to this device during re-allocation.
MinRecharge	Double	1	The minimum recharge for this device when the record was written
MaxRecharge	Double	1	The maxnimum recharge for this device when the record was written
CurrentRecharge	Double	1	The energy allocated to this device during re-allocation.
LastRecharge	Double	1	The recharge rate assigned to this unit previously
CurrentStorage	Double	1	The energy stored in the device in kWh.
CurrentDischarge	Double	1	The discharge of the storage device in kWh.
ExtTemp	Double	1	The external temperature of the residence.
DateString	Char	32	A text version of the date and time.
RecordIndex	Integer	1	A sequential index for ordering the records in the table.

Appendix B : Server-push Broadcast Message Format

Message format

The Multicast message is structured as shown below

```
Address      Name     Type     Notes
0x00-0x03    SOH      32UINT   value = 0x87654321
0x04-0x07    BTF      32UNIT   bytes to follow (including checksum)
0x08-0x0B    VERSION  32UINT   current version of packet (1)
0x0C-0x0F    CMD      32UINT   command code
0x10-0x13    STARTID  32UINT   start id
0x14-0x17    ENDID    32UNIT      end id
0x18-0x1F    CHECKSUM 64UNIT   Alder 64 bit check sum from SOH to ENDID
```

This format allows a command to be sent to a range of ETS devices, or a command to be sent to a single device within the system.

SOH

The Start of Header value simply provides a starting point for the message.

BTF

This allows the size of the command to have a variable length and provides the handler to determine where the checksum is located in the data stream.

VERSION

As the system develops, the structure of the messages may change. Providing a version number in the packet allows the system to handle the packet appropriately.

CMD

This is the action to be taken by the ETS system. In the development of this system the following commands have been defined.

```
CMD_MULTI_ON = 1;  // units within a defined range turn on
CMD_MULTI_OFF = 2;     // units within a defined range turn off
CMD_ALL_ON = 3;    // all units turn on
CMD_ALL_OFF = 4;   // all units turn off
CMD_SINGLE_ON = 5;     // the identified unit turns on
CMD_SINGLE_OFF = 6;    // the identified unit turns off
CMD_X_MULTI_ON = 7;    // units within a defined range turn on, all others turn off
CMD_X_MULTI_OFF = 8;   // units within a defined range turn off, all others turn on
```

STARTID

This is the device ID of the first ETS unit that should act on this command. ETS units my act using this parameter if they are outside of this range, depending on the command issued.

ENDID

This is the device ID of the last ETS unit that should act on this command. ETS units my act using this parameter if they are outside of this range, depending on the command issued.

CHECKSUM

By adding a checksum to the data transmitted in the Multicast packet, the ETS unit can ensure that the data was not modified during transmission or reception. Packets that failed to pass the checksum test would be discarded by an ETS system.

Appendix C: Java Source Code

```
// server-push architecture source
// Each file has been appended into one.  Break the files at the comment above the 'package'
keywords

// configuration class
package net.wattbox.config;

/**
 * @author barnes
 * Holds the configuration variables
 *
 * <p>
 * Some system configuration information is spanned across class instances.
 * Rather then have these parameters defined in the class source code for each class,
 * these configuration parameters are stored in the Config class.
 *
 */
public class Config {
    public static final String dBaseName = "jdbc:mysql:///WattBoxPush";
    public static final String dBaseUserName = "root";
    public static final String dBasePassword = "cuc002";
    public static final String DriverClass = "com.mysql.jdbc.Driver";
    public static final double SystemScale = 1.0;
    public static final boolean RealTimeRun = false;
    public static final int TimeIncrement = 1000*60*60; // one hour
    /* 24 hour analysis by vintage start and end*/
    public static final String SimulationStart= "2011-02-26 00:00:00";
    public static final String SimulationEnd =  "2011-02-27 13:00:00";
     /* response time analysis start and end*/
    //public static final String SimulationStart= "2010-01-16 22:50:00";// 2010-01-16 22:50:00
WestCape increased to 4000 from 8000 for the response time analysis
    //public static final String SimulationEnd =  "2010-01-16 23:15:00";
    /* heating season analysis  start and end*/
    //public static final String SimulationStart= "2010-09-01 00:00:00";
    //public static final String SimulationEnd =  "2011-05-31 23:00:00";
}

// the SimETS class
package net.wattbox.gui;
import java.net.*;
import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
//import java.text.DateFormat;
//import java.text.SimpleDateFormat;
import java.util.*;
import java.util.regex.Pattern;

import net.wattbox.config.Config;
/**
 * Simulates an ETS (Electric Thermal Storage) device
 * @author barnes
 *
 */
public class SimETS implements Comparable<SimETS> {

    final String Digits    = "(\\p{Digit}+)";
    final String HexDigits  = "(\\p{XDigit}+)";
        // an exponent is 'e' or 'E' followed by an optionally
        // signed decimal integer.
    final String Exp    = "[eE][+-]?"+Digits;
    final String fpRegex   =
            ("[\\x00-\\x20]*"+ // Optional leading "whitespace"
            "[+-]?(" +    // Optional sign character
            "NaN|" +      // "NaN" string
            "Infinity|" + // "Infinity" string

            // A decimal floating-point string representing a finite positive
            // number without a leading sign has at most five basic pieces:
            // Digits . Digits ExponentPart FloatTypeSuffix
            //
            // Since this method allows integer-only strings as input
            // in addition to strings of floating-point literals, the
            // two sub-patterns below are simplifications of the grammar
            // productions from the Java Language Specification, 2nd
            // edition, section 3.10.2.

            // Digits ._opt Digits_opt ExponentPart_opt FloatTypeSuffix_opt
            "((("+Digits+"(\\.)?("+Digits+"?)("+Exp+")?)|"+

            // . Digits ExponentPart_opt FloatTypeSuffix_opt
            "(\\.("+Digits+")("+Exp+")?)|"+

           // Hexadecimal strings
           "((" +
            // 0[xX] HexDigits ._opt BinaryExponent FloatTypeSuffix_opt
            "(0[xX]" + HexDigits + "(\\.)?)|" +

            // 0[xX] HexDigits_opt . HexDigits BinaryExponent FloatTypeSuffix_opt
            "(0[xX]" + HexDigits + "?(\\.)" + HexDigits + ")" +

            ")[pP][+-]?" + Digits + "))" +
             "[fFdD]?))" +
             "[\\x00-\\x20]*");// Optional trailing "whitespace"

    static final int PacketVersion = 1;
    static final int BTF = 24;
    static final int SOH = 0x87654321;

    static final int CMD_MULTI_ON = 1;               // units within a defined range turn on
    static final int CMD_MULTI_OFF = 2;              // units within a defined range turn off
    static final int CMD_ALL_ON = 3;             // all units turn on
    static final int CMD_ALL_OFF = 4;                // all units turn off
    static final int CMD_SINGLE_ON = 5;              // the identified unit turns on
    static final int CMD_SINGLE_OFF = 6;         // the identified unit turns off
    static final int CMD_X_MULTI_ON = 7;         // units within a defined range turn on  all
others turn off
    static final int CMD_X_MULTI_OFF = 8;        // units within a defined range turn off all
others turn on


    public String DeviceID;
    private String Password;
    private int PushID=-1;
    private double MaxRecharge;
    private double MaxDischarge;
    private double MaxStorage;
    private double CurrentRecharge;
    private double CurrentStorage;
    private double CurrentDischarge;
    private double MinRecharge;
    private double VolumeToHeat;
    private double EnergyIntensity = 1;
    private double TargetTemp;
    private double ExtTemp;
    private Random generator = new Random();
    private long LastTick=0;
    private double PostMaxRecharge = 0;
    private double LastDischarge = 0;

    public double EnergyFromGrid;
    int Port;
    double temperature;
    private GregorianCalendar LocalInit;
    private GregorianCalendar VirtualInit;
    private GregorianCalendar EndTime;
    private GregorianCalendar HitServerTime;

    String  McastAddr;
    Connection con = null;
    Statement stmt= null;
    ResultSet rs;

    TimeZone zone = TimeZone.getTimeZone("Canada/Atlantic");
    GregorianCalendar HoldCalendar = new GregorianCalendar(zone);
    DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    public GregorianCalendar ETSCalendar;
    boolean TimeToRegister = false;
    int timescale; // this is a scale factor for the timestep
    int timestep; // this is how often the system runs the tick_task

    Timer tick_timer;
    Timer register_timer;
    TimerTask register_task;
    TimerTask tick_task;
    boolean Registered = false;


    /**
     * @return The port monitored for UDP packets
     */
    public int getPort() {
        return Port;
    }

    /**
     * Sets  the port monitored for UDP packets
     * @param port the port to be monitored
     */
    public void setPort(int port) {
        Port = port;
    }

    /**
     * @return the Multicast address string used for port monitoring
     */
    public String getMcastAddr() {
        return McastAddr;
    }

    /**
     * Sets the Multicast address string used for port monitoring
     * @param mcastAddr a string in the form xxx.xxx.xxx where x is a number.
     */
    public void setMcastAddr(String mcastAddr) {
        McastAddr = mcastAddr;
    }
    /**
     * Sets the time of the ETS to the passed value
     * @param aTime a long value used to set the ETS time in MSEC since 1970
     */
    public void setETSTime (long aTime)
    {
        ETSCalendar.setTimeInMillis(aTime);
    }

    /**
     * @return The next time the ETS will contact the server in MS since 1970
     */
    public long getHitServerTime()
    {
        return HitServerTime.getTimeInMillis();
    }
```

```
/**
 * @return a character string containing the DeviceID
 */
public String getDeviceID() {
    return DeviceID;
}

/**
 * @param deviceID the new deviceID to be used by the ETS
 */
public void setDeviceID(String deviceID) {
    DeviceID = deviceID;
}
/**
 * @return a character string containing the password used for this ETS
 */
public String getPassword() {
    return Password;
}
/**
 * @param password the new password for this device
 */
public void setPassword(String password) {
    synchronized (this) {
        Password = password;
    }
}
/**
 * @return the id (integer value) used to identify this device in UDP commands
 */
public int getPushID() {
    return PushID;
}
/**
 * @param pushID sets the id (integer value) used to identify this device in UDP commands.
 * Normally this value is assigned by the server during registration
 */
public void setPushID(int pushID) {
    synchronized (this) {
        PushID = pushID;
    }
}

/**
 * @return The maximum recharge rate of this ETS
 */
public double getMaxRecharge() {
    return MaxRecharge;
}

/**
 * @param maxRecharge assigns the maximum recharge rate (kWh) of this ETS per hour
 */
public void setMaxRecharge(double maxRecharge) {
    synchronized (this) {
        MaxRecharge = maxRecharge;
    }
}

/**
 * @return the maximum possible discharge (kWh) of this ETS per hour
 */
public double getMaxDischarge() {
    return MaxDischarge;
}

/**
 * @param maxDischarge assigns the maximum possible discharge (kWh) per hour
 */
public void setMaxDischarge(double maxDischarge) {
    synchronized (this) {
        MaxDischarge = maxDischarge;
    }
}

/**
 * @return the maximum storage capacity of the ETS in kWh
 */
public double getMaxStorage() {
    return MaxStorage;
}

/**
 * @param maxStorage assigns the maximum possible storage of the ETS in kWh
 */
public void setMaxStorage(double maxStorage) {
    synchronized (this) {
        MaxStorage = maxStorage;
    }
}

/**
 * @return the current recharge rate of the ETS in kWh
 */
public double getCurrentRecharge() {
    return CurrentRecharge;
}

/**
 * @param currentRecharge assigns the current recharge rate of the ETS in kWh
 */
public void setCurrentRecharge(double currentRecharge) {
    synchronized (this) {
        CurrentRecharge = currentRecharge;
    }
}

/**
 * @return the current storage level of the ETS device in kWh
 */
public double getCurrentStorage() {
    return CurrentStorage;
}

/**
 * @param currentStorage assigns the current storage level of the ETS device in kWh
 */
public void setCurrentStorage(double currentStorage) {
    synchronized (this) {
        CurrentStorage = currentStorage;
    }
}

/**
 * @return the current discharge rate of the ETS in kWh
 */
public double getCurrentDischarge() {
    return CurrentDischarge;
}

/**
 * @param currentDischarge
 */
public void setCurrentDischarge(double currentDischarge) {
    synchronized (this) {
        CurrentDischarge = currentDischarge;
    }
}

/**
 * @return the Minimum recharge level required by the ETS device
```

```
 */
public double getMinRecharge() {
    return MinRecharge;
}

/**
 * @param minRecharge assigns the minimum recharge rate for the ETS
 */
public void setMinRecharge(double minRecharge) {
    synchronized (this) {
        MinRecharge = minRecharge;
    }
}

/**
 * @return the volume of the residence the ETS is heating
 */
public double getVolumeToHeat() {
    return VolumeToHeat;
}

/**
 * @param volumeToHeat sets the volume of the residence the ETS is heating
 */
public void setVolumeToHeat(double volumeToHeat) {
    synchronized (this) {
        VolumeToHeat = volumeToHeat;
    }
}

/**
 * @return the internal target temperature for the residence
 */
public double getTargetTemp() {
    return TargetTemp;
}

/**
 * @param targetTemp assigns the internal target temperature for the residence
 */
public void setTargetTemp(double targetTemp) {
    synchronized (this) {
        TargetTemp = targetTemp;
    }
}

/**
 * @return the external temperature of the residence
 */
public double getExtTemp() {
    return ExtTemp;
}

/**
 * @param extTemp assigns the external temperature of the residence
 */
public void setExtTemp(double extTemp) {
    synchronized (this) {
        ExtTemp = extTemp;
    }
}

/**
 * @param aVal a multiplier used to accelerate time
 */
public void setTimeScale(int aVal) {
    synchronized (this) {
        timescale = aVal;
    }
}
```

```
    /**
     * @return the multiplier used to accelerate time
     */
    public int getTimeScale() {
        return timescale;
    }

    /**
     * @param aVal assigns a discrete time interval used when a tick occurs
     */
    public void setTimeStep(int aVal) {
        synchronized (this) {
            timestep = aVal;
        }
    }

    /**
     * @return a discrete time interval used when a tick occurs
     */
    public int getTimeStep() {
        return timestep;
    }
    /**
     * UDP command hsndler
     *
     * @param cmd  the command sent
     * @param startID the start id for the command
     * @param endID the end if for the command
     * <p>
     * This method handles UDP commands sent by the server.  Only one process can have acccess to
     * a port at a time so it was necessary to have the UDP commands handled by the owner of this
class and the
     * command passed into the ETS device using a function call.
     */
    public void HandleUDPCmd ( int cmd, int startID, int endID)
    {
        TimeToRegister = true;
        //if (!EveningHours())
        {
            switch (cmd)
            {
                case CMD_ALL_ON:
                    CurrentRecharge = PostMaxRecharge;
                //  System.out.println("UDP: AllON");
                    break;
                case CMD_ALL_OFF:
                    CurrentRecharge = MinRecharge;
                //  System.out.println("UDP: AllOFF");
                    break;
                case CMD_MULTI_ON:
                    if ((PushID >=startID)&&(PushID<=endID))
                        CurrentRecharge = PostMaxRecharge;
                    break;
                case CMD_MULTI_OFF:
                    if (!((PushID >=startID)&&(PushID<=endID)))
                        CurrentRecharge = MinRecharge;
                    break;
                case CMD_SINGLE_ON:
                    if (PushID ==startID)
                        CurrentRecharge = PostMaxRecharge;
                    break;
                case CMD_SINGLE_OFF:
                    if (PushID ==startID)
                        CurrentRecharge = MinRecharge;
                    break;
                case CMD_X_MULTI_ON:
                    if ((PushID >=startID)&&(PushID<=endID))
                        CurrentRecharge = PostMaxRecharge;
                    else
                        CurrentRecharge = MinRecharge;
                    break;
                case CMD_X_MULTI_OFF:
                    if (!((PushID >=startID)&&(PushID<=endID)))
                        CurrentRecharge = MinRecharge;
                    else
                        CurrentRecharge = PostMaxRecharge;
                    break;

            }

        }
    HitServer();
    }

    /**
     * This method reads the current temperature from a MySQL database table using the current
time as
     * seen by the ETS device.
     *
     */
    public void ReadExtTemp()
    {
        String query = null;
        synchronized (this){

        try {
            query = "SELECT * FROM wind_data_table WHERE aDateTime <=
"+ETSCalendar.getTimeInMillis()+" ORDER BY aDateTime DESC LIMIT 0,1;";
            rs = stmt.executeQuery(query);
            if (rs.next())
            {
                ExtTemp = rs.getDouble("Temperature");
                //System.out.println("Time "+formatter.format(NextRecordTime.getTime())+"Power
output:"+rs.getDouble("WindOutput"));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        }
    }

    /**
     * This method archive the current storage of the ETS device in a MySQL table
     */
    public void ArchiveStorage()
    {
        String query = null;
        synchronized(this) {

        try {
            query = "INSERT chargelevel (DateString,aDateTime,DeviceID,ChargeLevel)
VALUES('"+dfm.format(ETSCalendar.getTime())+"',"+ETSCalendar.getTimeInMillis()+",'"+DeviceID+"',"
+CurrentStorage+");";
            stmt.execute(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        }
    }

    /**
     * @return returns true if the current time as seen by the ETS deivce is during the evening
hours (23:00 to 07:00)
     */
    public boolean EveningHours()
    {
        boolean result=true;
        long DayStart;
        long DayEnd;
        // determine if we are in the 'auto recharge time';
        // get rid of the fractional days
        long Hour = ETSCalendar.get(Calendar.HOUR_OF_DAY);
        long Minute = ETSCalendar.get(Calendar.MINUTE);
        long Second = ETSCalendar.get(Calendar.SECOND);
        long Delta = Hour*60*60+Minute*60+Second;

        DayStart =  60 * 60 * 7;  // 7 am
        DayEnd =  60 * 60 * 23; // 11 pm

        if ((Delta > DayStart)&(Delta < DayEnd))
        {
            result = false;
        }
        return result;
    }

    /**
     * This method is called by the owner of the ETS instance to discharge, recharge and register 
with the server if required.
     */
    public void run()
    {
        long TimeDelta = 0;
        // determine how long has passed since the timer last ran
        TimeDelta = (ETSCalendar.getTimeInMillis() - LastTick)/1000;
        // read the current temperature
        ReadExtTemp();

        // first handle any depletion since the past tick
        DoDischargeAndRecharge((int) TimeDelta);
        //Hit the server and find out if we should be on of off
        if (ETSCalendar.getTimeInMillis() > HitServerTime.getTimeInMillis())
        {
            HitServer();
            int randomDelay = (generator.nextInt(30)+30)*1000;// delay somewhere in the next 20 
seconds
            HitServerTime.setTimeInMillis(ETSCalendar.getTimeInMillis()+ randomDelay);// hit the 
server ~ 60 seconds from now
        }
        LastTick=ETSCalendar.getTimeInMillis() ;
    }


    /**
     * This method connects to the HTTP server registration and status updates.
     */
    public void HitServer()
    {
        HttpURLConnection connection = null;
        BufferedReader rd = null;
        InputStream is = null;
        String line = null;
        URL serverAddress = null;
        String agent = "WattBoxETS";
        String type = "application/x-www-form-urlencoded";
        String data ="";
        try {
            data = URLEncoder.encode("deviceid", "UTF-8") + "="
            + URLEncoder.encode(DeviceID, "UTF-8");
            data += "&" + URLEncoder.encode("password", "UTF-8") + "="
            + URLEncoder.encode(Password, "UTF-8");
            data += "&" + URLEncoder.encode("MinRecharge", "UTF-8") + "="
            + URLEncoder.encode(Double.toString(MinRecharge), "UTF-8");
            data += "&" + URLEncoder.encode("MaxRecharge", "UTF-8") + "="
            + URLEncoder.encode(Double.toString(PostMaxRecharge), "UTF-8");
            data += "&" + URLEncoder.encode("CurrentRecharge", "UTF-8") + "="
            + URLEncoder.encode(Double.toString(CurrentRecharge), "UTF-8");
        //  System.out.println("Current recharge reported as "+CurrentRecharge);
            data += "&" + URLEncoder.encode("ExtTemp", "UTF-8") + "="
            + URLEncoder.encode(Double.toString(ExtTemp), "UTF-8");
            data += "&" + URLEncoder.encode("CurrentStorage", "UTF-8") + "="
            + URLEncoder.encode(Double.toString(CurrentStorage), "UTF-8");
            data += "&" + URLEncoder.encode("CurrentDischarge", "UTF-8") + "="
            + URLEncoder.encode(Double.toString(CurrentDischarge), "UTF-8");
            //if (addDevice)
                data += "&" + URLEncoder.encode("action", "UTF-8") + "="
                + URLEncoder.encode("register", "UTF-8");
            //else
            //  data += "&" + URLEncoder.encode("action", "UTF-8") + "="
            //  + URLEncoder.encode("unregister", "UTF-8");

            data += "&" + URLEncoder.encode("PostTime", "UTF-8") + "="
            + URLEncoder.encode(dfm.format(ETSCalendar.getTime()), "UTF-8");
        } catch (UnsupportedEncodingException e1) {
            e1.printStackTrace();
        }

        try {
            serverAddress = new URL(
                    "http://localhost/WattBoxPush/GetState"); // set
            // up
            // out
            // communications
            // stuff
            connection = null; // Set up the initial connection
            connection = (HttpURLConnection) serverAddress.openConnection();
            connection.setRequestMethod("POST");
            connection.setRequestProperty("User-Agent", agent);
            connection.setRequestProperty("Content-Type", type);
            connection.setRequestProperty("Content-Length",
                    Integer.toString(data.length()));
            connection.setUseCaches(false);
            connection.setDoInput(true);
            connection.setDoOutput(true);
            connection.setReadTimeout(30000);
            // Send request
            DataOutputStream wr = new DataOutputStream(
                    connection.getOutputStream());
            wr.writeBytes(data);
            wr.flush();
            wr.close();
            // Get Response
            if (connection.getResponseCode() == 200)
            {
                is = connection.getInputStream();
            }
            else
            { /* error from server */
                is = connection.getErrorStream();
            }
            //is = connection.getInputStream();
            rd = new BufferedReader(new InputStreamReader(is));
            StringTokenizer st;
            String key;
            String val;
//          StringBuffer response = new StringBuffer();
            //CurrentRecharge = MinRecharge;
            while ((line = rd.readLine()) != null) {
                //System.out.println(line);
                st = new StringTokenizer(line, "=");
                while (st.hasMoreElements())
                {
                    key = st.nextToken();
                    if (st.hasMoreElements())
                        val = st.nextToken();
                    else
                        val = "NULL";
                    if (key.matches("PushID"))
                    {
                      if (Pattern.matches(fpRegex, val))
                          PushID = Integer.valueOf(val);
                    }
                }
            }

            //if (!addDevice) PushID = -1;
            connection.disconnect();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (ProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // close the connection, set all objects to null
            if (connection != null)
                connection.disconnect();
            rd = null;
            connection = null;
        }
        ReadExtTemp();
    }



    /**
     * constructor for ETS
     * @param DeviceID The deviceID
     * @param Password The password
     * @param MaxRecharge The maximum recharge rate of the ETS in kWh
     * @param MaxDischarge The maximum discharge rate of the ETS in kWh
     * @param MaxStorage The maximum storage of the ETS in kWh
     * @param VolumeToHeat The volume of the residnce to heat
     * @param aEnergyIntensity a parameter used to calcuate the energy required to heat the
residence
     * @param TargetTemp target internal temperature of the residnce
     */
    public SimETS(String DeviceID, String Password, double MaxRecharge,
            double MaxDischarge, double MaxStorage, double VolumeToHeat,
            Double aEnergyIntensity, double TargetTemp) {
        TimeZone zone = TimeZone.getTimeZone("Canada/Atlantic");
        ETSCalendar = new GregorianCalendar(zone);
        EndTime = new GregorianCalendar(zone);
        HitServerTime = new GregorianCalendar(zone);
        LocalInit = new GregorianCalendar(zone);
        VirtualInit = new GregorianCalendar(zone);
        synchronized (this) {
            setDeviceID(DeviceID);
            setPassword(Password);
            setMaxRecharge(MaxRecharge);
            setMaxDischarge(MaxDischarge);
            setMaxStorage(MaxStorage);
            setCurrentStorage(MaxStorage*0.7);
            setMinRecharge(0);
            setCurrentDischarge(0);
            setVolumeToHeat(VolumeToHeat);
            setTargetTemp(TargetTemp);
            setTimeScale(1);
            ExtTemp = TargetTemp;
            EnergyIntensity = aEnergyIntensity;
            initialize();
        }
        //(new Thread(new UDPListner())).start();
    }

    /**
     * Initialization routine that establishes a connection to the database and creates a
statement.
     */
    private void initialize()
    {
        try {
            con = DriverManager.getConnection("jdbc:mysql:///wattboxpush",
                "root", "cuc002");
            stmt = con.createStatement();


        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
    /**
     * records emergency energy usage when the ETS is fully depleated in a MySQL table
     * @param aVal the energy required by the ETS device
     */
    public void ArchiveEmergencyStorge(double aVal)
    {
        Statement stmt;
        String query = null;
        try {
             stmt = con.createStatement();
            query = "INSERT INTO emergency_storage (aDateTime," +
                                             "DeviceID," +
                                             "ResSize,"+
                                              "EnergyIntensity,"+
                                             "EmergencyRecharge," +
                                             "ExtTemp) values ("+
                                             ETSCalendar.getTimeInMillis()+","+
                                             "'"+DeviceID+"',"+
                                             +VolumeToHeat/2.5+","+
                                             +aVal+","+
                                             +EnergyIntensity+","+
                                             +ExtTemp+");";

            stmt.execute(query);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * Discharges the ETS
     * @param seconds
     *
     * Discharges the ETS energy store based upon the heating requirements of the residence and
the elaspsed time.
     */
    public void  Discharge(int seconds) {
        double TempDelta;
        LastDischarge = CurrentDischarge;
        CurrentDischarge = 0;
        TempDelta = TargetTemp-ExtTemp;
        if (TempDelta > 0)
            CurrentDischarge = EnergyIntensity*VolumeToHeat*TempDelta*seconds*2.77777777778e-7;
// this is the kwh used to space heating during this interval
        CurrentStorage -= CurrentDischarge;


        if (CurrentStorage < 0)
        {
            ArchiveEmergencyStorge(-CurrentStorage);
            CurrentStorage =0;
        }
    }

    /**
     * @return the number of seconds remaining in the evening recharge cycle
     */
    public long SecondsForRecharge()
    {
        long  result=0;
        long DayStart;
        long DayEnd;
        // determine if we are in the 'auto recharge time';

        // get rid of the fractional days
        long Hour = ETSCalendar.get(Calendar.HOUR_OF_DAY);
        long Minute = ETSCalendar.get(Calendar.MINUTE);
        long Second = ETSCalendar.get(Calendar.SECOND);
        long Delta = Hour*60*60+Minute*60+Second;

        DayStart =  60 * 60 * 7;  // 7 am
        DayEnd =  60 * 60 * 23; // 11 pm

        if (Delta >= DayEnd)
        {
            result = DayStart+(24*60*60-Delta);
        }
        else if (Delta <= DayStart)
        {
            result  = DayStart - Delta;
        }
        return result;
    }

    /**
     * @param seconds The number of seconds to recharge the ETS.  This function also calculates
the min recharge rate
     * and a maximum possible recharge rate
     */
    public void Recharge(int seconds)
    {
        // keep it simple, simply recharge based upon the existing parameters
        double HoursForRecharge=1;
        CurrentStorage += (CurrentRecharge*seconds/3600);
        // don't over charge
        if (CurrentStorage >MaxStorage)
            System.out.println("Over charged");
        CurrentStorage = Math.min(CurrentStorage,MaxStorage);



        /*************************************
         * This models an ETS that uses max wind any time
         */
        // calculate the maximum recharge rate needed to recharge over the next hour (can't be >
MaxRecharge)

        MinRecharge = 0;
        PostMaxRecharge = Math.min(MaxRecharge,MaxStorage-CurrentStorage);
        // the minimum recharge rate depends on the time of day
        if (EveningHours())
        {
            // during the evening we recharge at an optimum rate to level out the load
            HoursForRecharge =SecondsForRecharge() /3600.0;
            // ensure no division by zero , a minimum of 5 minutes (0.0833 hours) must available
for recharge
            if (HoursForRecharge >= 0.0833)
                MinRecharge = Math.max(0,((MaxStorage-CurrentStorage-
LastDischarge)/HoursForRecharge))+LastDischarge;
            // the MinRecharge can not be greater then the maximum recharge
            MinRecharge = Math.min(MinRecharge, MaxRecharge);
            // the max recharge must be greater then or equal to the min recharge
            PostMaxRecharge = Math.max(PostMaxRecharge, MinRecharge);
        }
        // the MinRecharge can not be greater then the maximum recharge
        MinRecharge = Math.min(MinRecharge, MaxRecharge);
        // the max recharge must be greater then or equal to the min recharge
        PostMaxRecharge = Math.max(PostMaxRecharge, MinRecharge);
    }

    /**
     * @param seconds
     */
    public void DoDischargeAndRecharge(int seconds) {

        Discharge(seconds);
        Recharge(seconds);
        if (CurrentStorage < 0)
            CurrentStorage = 0;
        if (CurrentStorage > MaxStorage)
        {
        //  System.out.println("Current recharge set to Min after fully charging");
            CurrentStorage = MaxStorage;
            CurrentRecharge = MinRecharge;
        }
    }

    /**
     * @param StartTime the time the simulation starts
     * @param aEndTime The time the simulation ends
     */
    public void InitTimeStamp(Date StartTime,Date aEndTime) {
        LocalInit.setTime(new Date());              // local time the system started
        VirtualInit.setTime(StartTime);
        EndTime.setTime(aEndTime);                // ets time to stop simulating
        ETSCalendar.setTime(StartTime);              // set the start time for the ETS
        LastTick = ETSCalendar.getTimeInMillis();
        //int randomDelay = (generator.nextInt( 10)+10)*1000;// delay somewhere in the next 20
seconds
        //timer.schedule(new tick_task(), randomDelay, 1000);
        HitServerTime.setTimeInMillis(VirtualInit.getTimeInMillis());
    }
    /* (non-Javadoc)
     * @see java.lang.Comparable#compareTo(java.lang.Object)
     *
     * helper function to sort ETS devices based on when they will hit the server
     */
    @Override
    public int compareTo(SimETS o) {

        long delta;
        delta = HitServerTime.getTimeInMillis()-o.getHitServerTime();
        if (delta >0 ) return 1;
        else if (delta < 0) return -1;
        else return 0;
    }
}


// system load class
package net.wattbox.gui;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;
import java.util.Timer;
import java.util.TimerTask;
import net.wattbox.config.Config;


/**
 * @author barnes
 *
 *Simulates a system load
 */
/**
 * @author barnes
 *
 */
public class SimSystemLoad {
    GregorianCalendar ThisRecordTime;
    GregorianCalendar NextRecordTime;
    double SystemLoad;
    long HoldTime;
    long DeltaTime;

    String DeviceID = "SummerSideLoad";
    String Password;
    Connection con = null;
    Statement stmt;
    ResultSet rs;
    String query;
    GregorianCalendar ETSCalendar;
    GregorianCalendar EndTime;

    /**
     * Constructor for the system load
     *
     * Initializes a connection to a MySQL database
     */
    public SimSystemLoad()
    {
        try {
            Class.forName(Config.DriverClass).newInstance();
            con = DriverManager.getConnection(Config.dBaseName,
                      Config.dBaseUserName, Config.dBasePassword);
            stmt = con.createStatement();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        DeviceID = "SummerSideLoad";
        TimeZone zone = TimeZone.getTimeZone("GMT+0");
        ETSCalendar = new GregorianCalendar(zone);
        ThisRecordTime =  new GregorianCalendar(zone);
        EndTime =  new GregorianCalendar(zone);
        NextRecordTime =  new GregorianCalendar(zone);
    }

    /**
     * Sets the time of the ETS to the passed value
     * @param aTime a long value used to set the ETS time in MSEC since 1970
     */
    public void setETSTime (long aTime)
    {
        ETSCalendar.setTimeInMillis(aTime);
    }
    /**
     * This method is called by the owner of the system load instance to update the system load
in the database based on the current time.
     */
    public void run()
    {
        // get the new output value from the database if it's past the next update time
        if (ETSCalendar.after(NextRecordTime))
        {
            try
            {
                query = "SELECT PowerLoad FROM wind_data_table WHERE aDateTime =
"+Long.toString((NextRecordTime.getTimeInMillis()));
                rs = stmt.executeQuery(query);
                if (rs.next())
                {
                    SystemLoad = rs.getDouble("PowerLoad");
                    HoldTime=NextRecordTime.getTimeInMillis();
                    DeltaTime = NextRecordTime.getTimeInMillis()% (60*60*1000);
                    HoldTime -=DeltaTime;
                    //System.out.println("Time "+formatter.format(NextRecordTime.getTime())+"Power
output:"+rs.getDouble("WindOutput"));
                }
                // now record the time of the next record
                query = "SELECT * FROM wind_data_table WHERE aDateTime >
"+Long.toString((NextRecordTime.getTimeInMillis()))+ " LIMIT 0, 1";
                rs = stmt.executeQuery(query);
                if (rs.next())
                {
                    NextRecordTime.setTimeInMillis(rs.getLong("aDateTime"));

                }
                rs.close();
                // update the producers table
                query ="UPDATE system_loads SET electrical_load =
"+SystemLoad/Config.SystemScale+" WHERE DeviceId = '"+DeviceID+"';";
                stmt.executeUpdate(query);
            }
            catch (SQLException e)
            {
                e.printStackTrace();
            }
        }
    }

    /**
     * @param StartTime the time the simulation starts
     * @param aEndTime The time the simulation ends
     */
    public void InitTimeStamp(Date StartTime,Date aEndTime) {
        // this procedure sets the local time of each ETS to the same value
        DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        SimpleDateFormat formatter = new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
        // Set the start time
        ETSCalendar = new GregorianCalendar();
        ETSCalendar.setTime(StartTime);
        EndTime.setTime(aEndTime);
        ThisRecordTime.setTime(StartTime);

        // Initialize the record times by checking the database
        try
        {
            query = "SELECT * FROM wind_data_table WHERE aDateTime >=
"+Long.toString((ThisRecordTime.getTimeInMillis()))+ " LIMIT 0, 2";
            rs = stmt.executeQuery(query);
            if (rs.next())
            {
                SystemLoad = rs.getDouble("PowerLoad");
                ThisRecordTime.setTimeInMillis(rs.getLong("aDateTime"));
                System.out.println("First record:  "+dfm.format(ThisRecordTime.getTime()));
                if (rs.next())
                {
                    NextRecordTime.setTimeInMillis(rs.getLong("aDateTime"));
                    System.out.println("Second record:  "+dfm.format(NextRecordTime.getTime()));
                }
            }
            rs.close();
            query = "INSERT system_loads (DeviceID,electrical_load)
values('"+DeviceID+"',"+SystemLoad/Config.SystemScale+");";
            stmt.execute(query);
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
        System.out.println("SystemLoad initialized to "+formatter.format(ETSCalendar.getTime()));
    }
}


// simulated windfarm class
package net.wattbox.gui;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;
import java.net.URLEncoder;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.SimpleDateFormat;
//import java.text.DateFormat;
//import java.text.SimpleDateFormat;
//import java.text.SimpleDateFormat;
//import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;
import java.util.Timer;
import java.util.TimerTask;

import net.wattbox.config.Config;


public class SimWindFarm {
    GregorianCalendar ThisRecordTime;
    GregorianCalendar NextRecordTime;
    double PowerOutput;
    String DeviceID;
    String Password;
    Connection con = null;
    Statement stmt;
    ResultSet rs;
    String query;
    GregorianCalendar ETSCalendar;
    private GregorianCalendar EndTime;


    /**
     * Consructor for the simulated wind farm class
     * @param aDeviceID  The device ID used for registration
     * @param aPassword  The device password used for registration
     */
    public SimWindFarm(String aDeviceID, String aPassword)
    {
        try {
            Class.forName(Config.DriverClass).newInstance();
            con = DriverManager.getConnection(Config.dBaseName,
                      Config.dBaseUserName, Config.dBasePassword);
            stmt = con.createStatement();

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        ETSCalendar = new GregorianCalendar();
        ThisRecordTime =  new GregorianCalendar();
        NextRecordTime =  new GregorianCalendar();
        EndTime =  new GregorianCalendar();

        DeviceID = aDeviceID;
        Password = aPassword;
    }
    /**
     * Sets the time of the ETS to the passed value
     * @param aTime a long value used to set the ETS time in MSEC since 1970
     */
    public void setETSTime (long aTime)
    {
        ETSCalendar.setTimeInMillis(aTime);
    }

    /**
     * Registers a wind farm with the server
     * @param addDevice Adds the device to the server if true, deletes the device if false
     * @return
     * @throws IOException
     */
    public boolean RegisterWithServer(boolean addDevice) throws IOException {
        boolean result = false;
        HttpURLConnection connection = null;
        BufferedReader rd = null;
        InputStream is = null;
        //String line = null;
        URL serverAddress = null;
        String agent = "WattBoxWindfarm";
        String type = "application/x-www-form-urlencoded";
        String data = null;

        data = URLEncoder.encode("deviceid", "UTF-8") + "="
                + URLEncoder.encode(DeviceID, "UTF-8");
        data += "&" + URLEncoder.encode("password", "UTF-8") + "="
                + URLEncoder.encode(Password, "UTF-8");
        data += "&" + URLEncoder.encode("output_kw", "UTF-8") + "="
                + URLEncoder.encode(Double.toString(PowerOutput/Config.SystemScale), "UTF-8");
        //+ URLEncoder.encode(Double.toString(PowerOutput), "UTF-8");
        if (addDevice)
            data += "&" + URLEncoder.encode("action", "UTF-8") + "="
                    + URLEncoder.encode("register", "UTF-8");
        else
            data += "&" + URLEncoder.encode("action", "UTF-8") + "="
                    + URLEncoder.encode("unregister", "UTF-8");

        try {
            serverAddress = new URL(
                    "http://localhost/WattBoxPush/ProducerRegistration"); // set
            connection = null; // Set up the initial connection
            connection = (HttpURLConnection) serverAddress.openConnection();
            connection.setRequestMethod("POST");
            connection.setRequestProperty("User-Agent", agent);
            connection.setRequestProperty("Content-Type", type);
            connection.setRequestProperty("Content-Length",
                    Integer.toString(data.length()));
            connection.setUseCaches(false);
            connection.setDoInput(true);
            connection.setDoOutput(true);
            connection.setReadTimeout(30000);
            // Send request
            DataOutputStream wr = new DataOutputStream(
                    connection.getOutputStream());
            wr.writeBytes(data);
            wr.flush();
            wr.close();
            // Get Response
            is = connection.getInputStream();
            rd = new BufferedReader(new InputStreamReader(is));
            rd.close();
            result = true;
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (ProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // close the connection, set all objects to null
            if (connection != null)
                connection.disconnect();
            rd = null;
            connection = null;
        }
        return result;
    }
    /**
     * This method is called by the owner of the wind farm instance to update the produciton
level in the database using a HTTP server based
     * on the current time .
     */
    public void run()
    {

        // get the new output value from the database if it's past the next update time
        if (ETSCalendar.after(NextRecordTime)||(ETSCalendar.getTimeInMillis() ==
NextRecordTime.getTimeInMillis()))
        {
            try
            {
                query = "SELECT * FROM wind_data_table WHERE aDateTime =
"+Long.toString((NextRecordTime.getTimeInMillis()));
                rs = stmt.executeQuery(query);
                if (rs.next())
                {
                    PowerOutput = rs.getDouble(DeviceID);
                    long HoldTime=NextRecordTime.getTimeInMillis();
                    long DeltaTime = NextRecordTime.getTimeInMillis()% (60*60*1000);
                    HoldTime -=DeltaTime;
                    //System.out.println(DeviceID+" "+"Time
"+formatter.format(NextRecordTime.getTime())+"Power output:"+rs.getDouble(DeviceID));
                }
                // now record the time of the next record
                query = "SELECT * FROM wind_data_table WHERE aDateTime >
"+Long.toString((NextRecordTime.getTimeInMillis()))+ " LIMIT 0, 1";
                //System.out.println( "Query sent to MySQL: "+query);
                rs = stmt.executeQuery(query);
                if (rs.next())
                {
                    NextRecordTime.setTimeInMillis(rs.getLong("aDateTime"));
                    //System.out.println("Next record:  "+dfm.format(NextRecordTime.getTime()));

                }
                else
                {
                    System.out.println("Next record
NOTFOUND!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
                }
                // update the producers table

            }
            catch (SQLException e)
            {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            // update the database with the latest production data
            try
            {
                RegisterWithServer(true);
            }
            catch (IOException e)
            {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
    /**
     * @param StartTime the time the simulation starts
     * @param aEndTime The time the simulation ends
     */
    public void InitTimeStamp(Date StartTime,Date aEndTime) {
        SimpleDateFormat formatter = new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
        // Set the start time
        ETSCalendar = new GregorianCalendar();
        ETSCalendar.setTime(StartTime);
        ThisRecordTime.setTime(StartTime);

        EndTime.setTime(aEndTime);

        // Initialize the next record time by checking the database
        try
        {
            // get the current production level
            query = "SELECT * FROM wind_data_table WHERE aDateTime <= 
"+Long.toString(ThisRecordTime.getTimeInMillis())+" ORDER BY aDateTime DESC LIMIT  0,1;";
            rs = stmt.executeQuery(query);
            if (rs.next())
            {

                PowerOutput = rs.getDouble(DeviceID);
                try
                {
                    RegisterWithServer(true);
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            // now initialize the next record time
            query = "SELECT * FROM wind_data_table WHERE aDateTime >= 
"+Long.toString(ThisRecordTime.getTimeInMillis())+" LIMIT  0,1;";
            rs = stmt.executeQuery(query);
            if (rs.next())
            {
                NextRecordTime.setTimeInMillis(rs.getLong("aDateTime"));
            }
        }
        catch (SQLException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //timer.schedule(new tick_task(), 500,1000);
        System.out.println("WindFarm "+DeviceID+" initialized to 
"+formatter.format(ETSCalendar.getTime()));
    }
}


package net.wattbox.gui;

//import java.util.Calendar;
//import java.text.DateFormat;
//import java.text.SimpleDateFormat;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;
import java.util.Timer;
import java.util.TimerTask;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.sql.*;
import java.util.zip.Adler32;
import java.util.zip.CheckedInputStream;



/**
 * @author barnes
 *This class is used to recalculate the distribution of wind energy and sends a
 *command to the ETS units using the UDP protocol.
 */
public class WindDistributor {
    static final int PacketVersion = 1;
    static final int BTF = 28;
    static final int SOH = 0x87654321;
    static final int CMD_MULTI_ON = 1; // units within a defined range turn on
    static final int CMD_MULTI_OFF = 2; // units within a defined range turn off
    static final int CMD_ALL_ON = 3; // all units turn on
    static final int CMD_ALL_OFF = 4; // all units turn off
    static final int CMD_SINGLE_ON = 5; // the identified unit turns on
    static final int CMD_SINGLE_OFF = 6; // the identified unit turns off
    static final int CMD_X_MULTI_ON = 7; // units within a defined range turn on
                                         // all others turn off
    static final int CMD_X_MULTI_OFF = 8; // units within a defined range turn
                                         // off all others turn on

    boolean AllOn = false;

    String DestAddress = "localhost";
    int DestPort = 4536;

    public int tail = -1;
    public int head = -1;
    public int PacketNumber = 0;
    boolean TimerRunning = false;
    TimerTask tick_task;
    private double NetEnergy=0;
    private GregorianCalendar ETSCalendar;
    private GregorianCalendar EndTime;
    private GregorianCalendar StartDelay;
    int timescale = 600; // this is a scale factor for the timestep
    int timestep = 1; // this is how often the system runs the tick_task
    DatagramSocket socket;
    DatagramPacket packet;
    InetAddress address;

    ByteArrayOutputStream ByteStream = new ByteArrayOutputStream();
    // now create a data output stream so we can write primitives
    DataOutputStream UDPData = new DataOutputStream(ByteStream);
    TimeZone zone = TimeZone.getTimeZone("Canada/Atlantic");
    GregorianCalendar HoldCalendar = new GregorianCalendar(zone);
    Statement stmt;
    ResultSet rs;

    DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    private Connection con = null;
```

```
/**
 * Sets the time of the ETS to the passed value
 * @param aTime a long value used to set the ETS time in MSEC since 1970
 */
public void setETSTime (long aTime)
{
    ETSCalendar.setTimeInMillis(aTime);
}

/**
 * Simple constructor that calls the initialize function
 */
public WindDistributor() {
    initialize();
}

/**
 *  initializes the class variables
 */
private void initialize() {
    ETSCalendar = new GregorianCalendar(zone);
    EndTime = new GregorianCalendar(zone);
    StartDelay = new GregorianCalendar(zone);
    String dip = DestAddress;
    try {
        address = InetAddress.getByName(dip);
    } catch (UnknownHostException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    }
    try {
        socket = new DatagramSocket();
    } catch (SocketException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

    try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        con = DriverManager.getConnection("jdbc:mysql:///WattBoxPush",
                "root", "cuc002");
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    try {
        stmt = con.createStatement();
    } catch (SQLException e3) {
        // TODO Auto-generated catch block
        e3.printStackTrace();
    }
}

/**
 * Builds and sends a UDP packet containing operating instructions for
 */
private void SendRequest() {

    try {
        ByteStream.reset();

        // create a byte array output stream so we can access the data as an
        // array of bytes

        /*
         * The format for the UDP packet sent by this class is as follows
         *
         * address Name Type Notes 0x00-0x03 SOH 32UINT value = 0x87654321
         * 0x04-0x07 BTF 32UNIT number of bytes to follow (including
         * checksum) 0x08-0x0B VERSION 32UINT current version of packet (1)
         * 0x0C-0x0F CMD 32UINT command code 0x10-0x13 STARTID 32UINT start
         * id 0x14-0x17 ENDID 32UNIT end id 0x18-0x1F CHECKSUM 64UNIT Alder
         * 64 bit check sum from SOH to ENDID
         */

        try {
            UDPData.writeInt(SOH);
            UDPData.writeInt(BTF);
            UDPData.writeInt(PacketNumber);
            PacketNumber++;
            UDPData.writeInt(PacketVersion);
            // write the appropriate command
            if (tail < head)
                UDPData.writeInt(CMD_X_MULTI_ON);
            else if (tail > head)
                UDPData.writeInt(CMD_X_MULTI_OFF);
            else if (AllOn)
                UDPData.writeInt(CMD_ALL_ON);
            else {
                if (head == -1)
                    UDPData.writeInt(CMD_ALL_OFF);
                else
                    UDPData.writeInt(CMD_X_MULTI_ON);
            }
            // write the start and stop ID's
            UDPData.writeInt(tail);
            UDPData.writeInt(head);
            System.out.println("Wind  startID:" + tail + " endID:" + head
                    + " Packet Number:" + (PacketNumber - 1));
            // append the checksum
            long HoldCRC = CalcCRC(ByteStream);
            UDPData.writeLong(HoldCRC);

        } catch (IOException e1) {
            e1.printStackTrace();
        }
        packet = new DatagramPacket(ByteStream.toByteArray(),
                ByteStream.size(), address, DestPort);
        socket.send(packet);
    } catch (IOException io) {
    }
}

/**
 * @param aStream
 * @return a long value containing the Adler32 checksum of the data in the stream
 */
private long CalcCRC(ByteArrayOutputStream aStream) {
    long result = 0;

    ByteArrayInputStream bais = new ByteArrayInputStream(
            aStream.toByteArray());
    CheckedInputStream cis = new CheckedInputStream(bais, new Adler32());
    byte[] tempBuf = new byte[aStream.size()];
    try {
        while (cis.read(tempBuf) >= 0) {
        }
        result = cis.getChecksum().getValue();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

/**
 * Resets the head and tail values to -1
 */
```

```
    public void ResetHeadTail() {
        head = -1;
        tail = -1;
    }

    /**
     * @return the lowest pushid of the ETS units
     */
    public boolean getFirst() {
        boolean result = false;

        try {
            rs = stmt
                    .executeQuery("SELECT * from storage_units ORDER BY PushID LIMIT 0,1;");
            if (rs.next()) {
                tail = rs.getInt("PushID");
                result = true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return result;
    }

    /**
     * Distributes the intermittent wind power to the ETS devices
     * @return true if there was  change in the wind power distribution
     */
    public boolean DistributeWind() {
        boolean result = false;
        double PowerProduced = 0;
        double MinLoad = 0;
        double PowerAvailable = 0;
        double EnergyGenerated = 0;
        double EnergyAllocatedBefore = 0;
        double EnergyAllocatedAfter = 0;
        double ThisAllocation = 0;
        boolean TurnAllOff = false;
        boolean DoAllocation = false;
        double MinDiff = 0;
        double MaxDiff  =0;
        double ExtTemp = 0;
        if (!TimerRunning) {
            TimerRunning = true;
            String query = "unasigned";
            ResetHeadTail();
            try {
                rs = stmt.executeQuery("SELECT SUM(output_kw) as total_energy from producers;");
                if (rs.next()) {
                    PowerProduced = rs.getDouble("total_energy");
                    EnergyGenerated = PowerProduced;
                }
                // account for the system load
                //Get the system load
                double SystemLoad = 0;
                query = "SELECT SUM(electrical_load) as total_energy from system_loads;";
                rs = stmt.executeQuery(query);
                if (rs.next())
                    SystemLoad = rs.getDouble("total_energy");

                rs = stmt.executeQuery("SELECT SUM(CurrentRecharge) as total_energy from 
storage_units;");
                if (rs.next()) {
                    EnergyAllocatedBefore = rs.getDouble("total_energy");
                }
                if ((PowerProduced - SystemLoad) > 0) {
                    if (((PowerProduced - SystemLoad) != NetEnergy)){
                        NetEnergy = (PowerProduced - SystemLoad);
                        // account for any energy that is being used that can not be
                        // turned off
                        rs = stmt.executeQuery("SELECT SUM(MinRecharge) as total_energy from
storage_units;");
                        if (rs.next()) {
                            MinLoad = rs.getDouble("total_energy");
                            PowerAvailable = PowerProduced - MinLoad - SystemLoad;
                            EnergyAllocatedAfter = MinLoad;
                        }
                        if (PowerAvailable > 0) {
                            if (getFirst()) {
                                AllOn = false;
                                TurnAllOff = false;
                                // there is access power to distribute and devices
                                // to take it
                                rs = stmt.executeQuery("SELECT * from storage_units ORDER BY
PushID;");
                                // power is distributed on a first come, first
                                // served basis
                                while ((PowerAvailable > 0) && (rs.next())) {
                                    DoAllocation = true;
                                    if ((PowerAvailable - rs.getDouble("MaxRecharge"))<0)
                                    {
                                        // Only turn on if we will use more wind power then we will
have to import
                                        MaxDiff = Math.abs(PowerAvailable -
rs.getDouble("MaxRecharge"));
                                        MinDiff = Math.abs(PowerAvailable -
rs.getDouble("MinRecharge"));
                                        if (MinDiff < MaxDiff)
                                        {
                                            // better off not turning on, we'll import more then we
would export
                                            DoAllocation = false;
                                            PowerAvailable = -1;
                                        }
                                    }

                                    if (DoAllocation)
                                    {
                                        head = rs.getInt("PushID");
                                        ThisAllocation = (rs.getDouble("MaxRecharge") -
rs.getDouble("MinRecharge"));
                                        PowerAvailable -= ThisAllocation;
                                        EnergyAllocatedAfter += ThisAllocation;
                                        ExtTemp = rs.getDouble("ExtTemp");

                                    }
                                    // System.out.println("Total:"+EnergyAllocatedAfter+" Allocated
"+ThisAllocation+" Min:"+rs.getDouble("MinRecharge")+" Max:"+rs.getDouble("MaxRecharge"));
                                }
                                SendRequest();
                                result = true;
                            }
                        }
                        // now record the energy usage
                        query = "INSERT energy_distribution
(EnergyProduced,EnergyUsedBefore,EnergyAllocatedAfter,ExtTemp,DateString) VALUES("
                                + EnergyGenerated
                                + ","
                                + EnergyAllocatedBefore
                                + ","
                                + EnergyAllocatedAfter
                                + ","
                                + ExtTemp
                                + ",'"
                                + dfm.format(ETSCalendar.getTime()) + "');";
                        stmt.execute(query);
                    }
                    // EnergyAllocatedAfter = EnergyGenerated-PowerAvailable;

                }
                else
                {
                    TurnAllOff = true;
                }

                if (TurnAllOff) {
                    // there are no units to process, or no energy available.
                    // Send a packet telling all units to turn off anyway
                    head = -1;
                    tail = -1;
                    AllOn = false;
                    SendRequest();
                    result = true;
                }


            } catch (SQLException e) {
                e.printStackTrace();
                System.out.println("SQL string:" + query);
            }
            TimerRunning = false;
        }
        return (result);
    }
}

// wind distributor class
package net.wattbox.gui;

//import java.util.Calendar;
//import java.text.DateFormat;
//import java.text.SimpleDateFormat;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;
import java.util.Timer;
import java.util.TimerTask;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.sql.*;
import java.util.zip.Adler32;
import java.util.zip.CheckedInputStream;



/**
 * @author barnes
 *This class is used to recalculate the distribution of wind energy and sends a
 *command to the ETS units using the UDP protocol.
 */
public class WindDistributor {
    static final int PacketVersion = 1;
    static final int BTF = 28;
    static final int SOH = 0x87654321;
    static final int CMD_MULTI_ON = 1; // units within a defined range turn on
    static final int CMD_MULTI_OFF = 2; // units within a defined range turn off
    static final int CMD_ALL_ON = 3; // all units turn on
    static final int CMD_ALL_OFF = 4; // all units turn off
    static final int CMD_SINGLE_ON = 5; // the identified unit turns on
    static final int CMD_SINGLE_OFF = 6; // the identified unit turns off
    static final int CMD_X_MULTI_ON = 7; // units within a defined range turn on
                                         // all others turn off
    static final int CMD_X_MULTI_OFF = 8; // units within a defined range turn
```

```
                                      // off all others turn on

boolean AllOn = false;

String DestAddress = "localhost";
int DestPort = 4536;

public int tail = -1;
public int head = -1;
public int PacketNumber = 0;
boolean TimerRunning = false;
TimerTask tick_task;
private double NetEnergy=0;
private GregorianCalendar ETSCalendar;
private GregorianCalendar EndTime;
private GregorianCalendar StartDelay;
int timescale = 600; // this is a scale factor for the timestep
int timestep = 1; // this is how often the system runs the tick_task
DatagramSocket socket;
DatagramPacket packet;
InetAddress address;

ByteArrayOutputStream ByteStream = new ByteArrayOutputStream();
// now create a data output stream so we can write primitives
DataOutputStream UDPData = new DataOutputStream(ByteStream);
TimeZone zone = TimeZone.getTimeZone("Canada/Atlantic");
GregorianCalendar HoldCalendar = new GregorianCalendar(zone);
Statement stmt;
ResultSet rs;

DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

private Connection con = null;
/**
 * Sets the time of the ETS to the passed value
 * @param aTime a long value used to set the ETS time in MSEC since 1970
 */
public void setETSTime (long aTime)
{
    ETSCalendar.setTimeInMillis(aTime);
}

/**
 * Simple constructor that calls the initialize function
 */
public WindDistributor() {
    initialize();
}

/**
 *  initializes the class variables
 */
private void initialize() {
    ETSCalendar = new GregorianCalendar(zone);
    EndTime = new GregorianCalendar(zone);
    StartDelay = new GregorianCalendar(zone);
    String dip = DestAddress;
    try {
        address = InetAddress.getByName(dip);
    } catch (UnknownHostException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
    }
    try {
        socket = new DatagramSocket();
    } catch (SocketException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

    try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            con = DriverManager.getConnection("jdbc:mysql:///WattBoxPush",
                    "root", "cuc002");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        try {
            stmt = con.createStatement();
        } catch (SQLException e3) {
            // TODO Auto-generated catch block
            e3.printStackTrace();
        }
    }

    /**
     * Builds and sends a UDP packet containing operating instructions for
     */
    private void SendRequest() {

        try {
            ByteStream.reset();

            // create a byte array output stream so we can access the data as an
            // array of bytes

            /*
             * The format for the UDP packet sent by this class is as follows
             *
             * address Name Type Notes 0x00-0x03 SOH 32UINT value = 0x87654321
             * 0x04-0x07 BTF 32UNIT number of bytes to follow (including
             * checksum) 0x08-0x0B VERSION 32UINT current version of packet (1)
             * 0x0C-0x0F CMD 32UINT command code 0x10-0x13 STARTID 32UINT start
             * id 0x14-0x17 ENDID 32UNIT end id 0x18-0x1F CHECKSUM 64UNIT Alder
             * 64 bit check sum from SOH to ENDID
             */

            try {
                UDPData.writeInt(SOH);
                UDPData.writeInt(BTF);
                UDPData.writeInt(PacketNumber);
                PacketNumber++;
                UDPData.writeInt(PacketVersion);
                // write the appropriate command
                if (tail < head)
                    UDPData.writeInt(CMD_X_MULTI_ON);
                else if (tail > head)
                    UDPData.writeInt(CMD_X_MULTI_OFF);
                else if (AllOn)
                    UDPData.writeInt(CMD_ALL_ON);
                else {
                    if (head == -1)
                        UDPData.writeInt(CMD_ALL_OFF);
                    else
                        UDPData.writeInt(CMD_X_MULTI_ON);
                }
                // write the start and stop ID's
                UDPData.writeInt(tail);
                UDPData.writeInt(head);
                System.out.println("Wind  startID:" + tail + " endID:" + head
                        + " Packet Number:" + (PacketNumber - 1));
                // append the checksum
                long HoldCRC = CalcCRC(ByteStream);
                UDPData.writeLong(HoldCRC);

            } catch (IOException e1) {
                e1.printStackTrace();
            }
            packet = new DatagramPacket(ByteStream.toByteArray(),
                    ByteStream.size(), address, DestPort);
            socket.send(packet);
        } catch (IOException io) {
        }
    }

    /**
     * @param aStream
     * @return a long value containing the Adler32 checksum of the data in the stream
     */
    private long CalcCRC(ByteArrayOutputStream aStream) {
        long result = 0;

        ByteArrayInputStream bais = new ByteArrayInputStream(
                aStream.toByteArray());
        CheckedInputStream cis = new CheckedInputStream(bais, new Adler32());
        byte[] tempBuf = new byte[aStream.size()];
        try {
            while (cis.read(tempBuf) >= 0) {
            }
            result = cis.getChecksum().getValue();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    /**
     * Resets the head and tail values to -1
     */
    public void ResetHeadTail() {
        head = -1;
        tail = -1;
    }

    /**
     * @return the lowest pushid of the ETS units
     */
    public boolean getFirst() {
        boolean result = false;

        try {
            rs = stmt
                    .executeQuery("SELECT * from storage_units ORDER BY PushID LIMIT 0,1;");
            if (rs.next()) {
                tail = rs.getInt("PushID");
                result = true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return result;
    }

    /**
     * Distributes the intermittent wind power to the ETS devices
     * @return true if there was  change in the wind power distribution
     */
    public boolean DistributeWind() {
        boolean result = false;
        double PowerProduced = 0;
        double MinLoad = 0;
        double PowerAvailable = 0;
        double EnergyGenerated = 0;
        double EnergyAllocatedBefore = 0;
        double EnergyAllocatedAfter = 0;
        double ThisAllocation = 0;
        boolean TurnAllOff = false;
        boolean DoAllocation = false;
        double MinDiff = 0;
        double MaxDiff  =0;
        double ExtTemp = 0;
        if (!TimerRunning) {
            TimerRunning = true;
            String query = "unasigned";
            ResetHeadTail();
            try {
                rs = stmt.executeQuery("SELECT SUM(output_kw) as total_energy from producers;");
                if (rs.next()) {
                    PowerProduced = rs.getDouble("total_energy");
                    EnergyGenerated = PowerProduced;
                }
                // account for the system load
                //Get the system load
                double SystemLoad = 0;
                query = "SELECT SUM(electrical_load) as total_energy from system_loads;";
                rs = stmt.executeQuery(query);
                if (rs.next())
                    SystemLoad = rs.getDouble("total_energy");

                rs = stmt.executeQuery("SELECT SUM(CurrentRecharge) as total_energy from 
storage_units;");
                if (rs.next()) {
                    EnergyAllocatedBefore = rs.getDouble("total_energy");
                }
                if ((PowerProduced - SystemLoad) > 0) {
                    if (((PowerProduced - SystemLoad) != NetEnergy)){
                        NetEnergy = (PowerProduced - SystemLoad);
                        // account for any energy that is being used that can not be
                        // turned off
                        rs = stmt.executeQuery("SELECT SUM(MinRecharge) as total_energy from 
storage_units;");
                        if (rs.next()) {
                            MinLoad = rs.getDouble("total_energy");
                            PowerAvailable = PowerProduced - MinLoad - SystemLoad;
                            EnergyAllocatedAfter = MinLoad;
                        }
                        if (PowerAvailable > 0) {
                            if (getFirst()) {
                                AllOn = false;
                                TurnAllOff = false;
                                // there is access power to distribute and devices
                                // to take it
                                rs = stmt.executeQuery("SELECT * from storage_units ORDER BY 
PushID;");
                                // power is distributed on a first come, first
                                // served basis
                                while ((PowerAvailable > 0) && (rs.next())) {
                                    DoAllocation = true;
                                    if ((PowerAvailable - rs.getDouble("MaxRecharge"))<0)
                                    {
                                        // Only turn on if we will use more wind power then we will 
have to import
                                        MaxDiff = Math.abs(PowerAvailable - 
rs.getDouble("MaxRecharge"));
                                        MinDiff = Math.abs(PowerAvailable - 
rs.getDouble("MinRecharge"));
                                        if (MinDiff < MaxDiff)
                                        {
                                            // better off not turning on, we'll import more then we 
would export
                                            DoAllocation = false;
                                            PowerAvailable = -1;
                                        }
                                    }

                                    if (DoAllocation)
                                    {
                                        head = rs.getInt("PushID");
                                        ThisAllocation = (rs.getDouble("MaxRecharge") -
rs.getDouble("MinRecharge"));
                                        PowerAvailable -= ThisAllocation;
                                        EnergyAllocatedAfter += ThisAllocation;
                                        ExtTemp = rs.getDouble("ExtTemp");

                                    }
                                    // System.out.println("Total:"+EnergyAllocatedAfter+" Allocated
"+ThisAllocation+" Min:"+rs.getDouble("MinRecharge")+" Max:"+rs.getDouble("MaxRecharge"));
                                }
                                SendRequest();
                                result = true;
                            }
                        }
                        // now record the energy usage
                        query = "INSERT energy_distribution
(EnergyProduced,EnergyUsedBefore,EnergyAllocatedAfter,ExtTemp,DateString) VALUES("
                                + EnergyGenerated
                                + ","
                                + EnergyAllocatedBefore
                                + ","
                                + EnergyAllocatedAfter
                                + ","
                                + ExtTemp
                                + ",'"
                                + dfm.format(ETSCalendar.getTime()) + "');";
                        stmt.execute(query);
                    }
                    // EnergyAllocatedAfter = EnergyGenerated-PowerAvailable;

                }
                else
                {
                    TurnAllOff = true;
                }

                if (TurnAllOff) {
                    // there are no units to process, or no energy available.
                    // Send a packet telling all units to turn off anyway
                    head = -1;
                    tail = -1;
                    AllOn = false;
                    SendRequest();
                    result = true;
                }


            } catch (SQLException e) {
                e.printStackTrace();
                System.out.println("SQL string:" + query);
            }
            TimerRunning = false;
        }
        return (result);
    }
}


// Test bed for Server-push
package net.wattbox.gui;
import java.awt.*;
import java.awt.event.*; // for ActionListener and ActionEvent
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
```

```java
import java.net.UnknownHostException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.StringTokenizer;
import java.util.TimeZone;
import java.util.zip.Adler32;
import java.util.zip.CheckedInputStream;
import java.util.Calendar;
import java.util.Collections;
import java.util.GregorianCalendar;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.*;
import net.wattbox.config.Config;
import com.toedter.calendar.JDateChooser;

/**
 * @author barnes
 *
 *This class is used has a the test bed for simulating the Server-push architecture
 */
public class SysMonMainPush {

    private String Start= Config.SimulationStart;
    private String End =  Config.SimulationEnd;

    private JFrame frame;
    private JTextArea textArea;
    List<SimETS> ETSList = new ArrayList<SimETS>();
    Connection con = null;
    private GregorianCalendar SystemCalendar;
    SimWindFarm West_Cape;
    SimWindFarm SummerSide;
    SimSystemLoad SystemLoad;
    WindDistributor Distributor;
    UDPListner aListner;
    Timer timer;
    TimerTask tick_task;
    int etsTickCount;
    Date StartTime;
    Date EndTime;
    GregorianCalendar InitTime;
    GregorianCalendar VirtualInit;
    TimeZone zone = TimeZone.getTimeZone("Canada/Atlantic");
    GregorianCalendar DistrbutionCalander = new GregorianCalendar(zone);

    WindDistributor DistributionTask;
    boolean CreateWindTable = false;
    public static String newline = System.getProperty("line.separator");
    private JDateChooser dcStart;
    private JDateChooser dcEnd;
    private JDateChooser dateChooser;
    private JTextField textField;
    private JTextField textField_1;
    private JTextField textField_2;
    private JTextField textField_3;
    private JTextField textField_4;
    private JTextField textField_5;
    private JTextField textField_6;
    private JTextField textField_7;

    ResultSet rs;
    String query;
    Statement stmt;
    DateFormat dfm;

    /**
     * @author barnes
     * This class is used to monitor a port for UDP commands sent from the server.  If the
command is validated,
     * it is passed to the ETS devices.
     */
    public class UDPListner implements Runnable
    {
        static final int PacketVersion = 1;
        static final int BTF = 24;
        static final int SOH = 0x87654321;
        static final int CMD_ON = 1;
        static final int CMD_OFF = 2;
        static final int CMD_ALL_ON = 3;
        static final int CMD_ALL_OFF = 4;
        String DestAddress = "localhost";
        String DestPort = "4536";
        DatagramSocket socket;
        private boolean PacketReceived = false;

        /**
         * @return set to true when the command is processed.  The application waits for this
parameter to be set
         * so the ETS units a re updated before the wimulation continues.
         */
        public boolean IsFinished ()
        {
            if (PacketReceived)
            {
                PacketReceived = false;
                return true;
            }
            else
                return false;
        }
        /* (non-Javadoc)
         * @see java.lang.Runnable#run()
         * the implementation waits for commands so it has to run in its own thread that can
block without blocking the application
         */
        public void run(){
            try
            {
                byte[] buffer = new byte[36];
                int port = 4536;
                try
                {
                    socket = new DatagramSocket(port);
                    while(true)
                    {
                        try
                        {
                            /*
                             * The format for the UDP packet sent by this class is as follows

                        address        Name       Type       Notes
                        0x00-0x03          SOH        32UINT     value = 0x87654321
                        0x04-0x07          BTF        32UNIT     number of bytes to follow
(including checksum)
                        0x08-0x0B          VERSION    32UINT     current version of packet (1)
                        0x0C-0x0F          CMD        32UINT     command code
                        0x10-0x13          STARTID    32UINT     start id
                        0x14-0x17           ENDID     32UNIT      end id
                        0x18-0x1F           CHECKSUM  64UNIT     Alder 64 bit check sum from SOH
to ENDID
                             */
                            //Receive request from client
                            //System.out.println("UDP listner start");
                            DatagramPacket packet = new DatagramPacket(buffer, buffer.length );

                            socket.receive(packet);
                            // create a byte array stream of the received data
                            ByteArrayInputStream ByteStream = new ByteArrayInputStream(buffer);
                            // create an input stream for primitives
                            DataInputStream UDPData = new DataInputStream(ByteStream);
                            if (UDPData.readInt() ==SOH)
                            {
                                UDPData.readInt();
                                UDPData.readInt();
                                UDPData.readInt();
                                int cmd = UDPData.readInt();
                                int startID = UDPData.readInt();
                                int endID = UDPData.readInt();
                                long PacketCS = UDPData.readLong();
                                // now compute the checksum
                                try
                                {
                                   byte[] tempBuf = new byte[28];
                                   System.arraycopy(buffer,0,tempBuf,0,28);
                                   ByteArrayInputStream bais = new ByteArrayInputStream(tempBuf);
                                   CheckedInputStream acis = new CheckedInputStream(bais, new
       Adler32());

                                   byte readBuffer[] = new byte[28];
                                   while (acis.read(readBuffer) >= 0);
                                   long holdCS = acis.getChecksum().getValue();
                                   if ( holdCS== PacketCS)
                                   {
                                       //System.out.println("CMD:"+cmd+" startID:"+startID+"
       endID:"+endID+" Packet Number:"+PacketNumber);

                                       // valid packet.  Forward to each ETS
                                       for (SimETS aETS : ETSList)
                                       {

                                           //System.out.println(aETS.DeviceID);
                                           aETS.HandleUDPCmd(cmd,startID, endID);
                                       }
                                       PacketReceived = true;
                                   }
                                }
                                catch (IOException e)
                                {
                                   e.printStackTrace();
                                }
                            }
                        }
                        catch(UnknownHostException ue){}
                    }
                }
                catch(java.net.BindException b){}
             }
             catch (IOException e){
                System.err.println(e);
             }
         }
       }

          /**
         * @param args
         * Main function for the application
         */
     public static void main(String[] args) {
         EventQueue.invokeLater(new Runnable() {
                public void run() {
                    try {
                        SysMonMainPush window = new SysMonMainPush();
                        window.frame.setVisible(true);
                    } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public SysMonMainPush() {
        initialize();
    }

    /**
     * @author barnes
     * This class is used to step through time in the simulation.  It is implemented as a timer
task.
     */
    class tick_task extends TimerTask {
        public void run()
        {
            //This event triggers all the devices in the system to update serially
            //In the real world, multiple devices could hit the servers at the same time
            long TimeDelta = 0;
            if (Config.RealTimeRun)
            {
                TimeDelta = System.currentTimeMillis() - InitTime.getTimeInMillis();
                SystemCalendar.setTimeInMillis(VirtualInit.getTimeInMillis()+TimeDelta);
            }
            else
            {
                TimeDelta = Config.TimeIncrement;
                SystemCalendar.setTimeInMillis(SystemCalendar.getTimeInMillis()+TimeDelta);
            }
            if (EndTime.after(SystemCalendar.getTime()))
            {
                // in the push implementation, all ets units run and report operating parameters
before wind is distributed.

                // sort the list of ETS units based upon when they are next serviced
                for(SimETS aETS : ETSList){
                    aETS.setETSTime(SystemCalendar.getTimeInMillis());
                }
                Collections.sort(ETSList);
                // process this time step
                for(SimETS aETS : ETSList){
                    aETS.run();
                }
                // by now each ETS has depleted based upon the temperature and reported it's
operating parameters to the server

                // update the wind farm production levels
                West_Cape.setETSTime(SystemCalendar.getTimeInMillis());
                West_Cape.run();
                SummerSide.setETSTime(SystemCalendar.getTimeInMillis());
                SummerSide.run();
                // update the system load
                SystemLoad.run();

                // now distribute the power to the ETS units
                Distributor.setETSTime(SystemCalendar.getTimeInMillis());
                if (Distributor.DistributeWind())
                {
                    // wait here until the packet is processe;
                    while (!(aListner.IsFinished()));
                }
                // now collect the statistics, we care about the wind energy that has been
redirected to ETS units vs exported.
                //Get the total energy allocated to storage units
                double EnergyAllocated=0;
                query = "SELECT SUM(CurrentRecharge) as energy_allocated from storage_units;";
                try {
                    rs = stmt.executeQuery(query);

                if (rs.next())
                    EnergyAllocated =rs.getDouble("energy_allocated");
                //Get the total energy being generated
                double EnergyProduced=0;
                query = "SELECT SUM(output_kw) as total_energy from producers;";
                rs = stmt.executeQuery(query);
                if (rs.next())
                    EnergyProduced = rs.getDouble("total_energy");

                //Get the system load
                double SystemLoad = 0;
                query = "SELECT SUM(electrical_load) as total_energy from system_loads;";
                rs = stmt.executeQuery(query);
                if (rs.next())
                    SystemLoad = rs.getDouble("total_energy");

                double EnergyAvailable = EnergyProduced - SystemLoad;
                double EnergyImported =0;
                double EnergyExported =0;
                if (EnergyAvailable > EnergyAllocated)
                {
                    EnergyImported =0;
                    EnergyExported = EnergyAvailable-EnergyAllocated;
                }
                else
                {
                    EnergyExported =0;
                    if (EnergyAvailable > 0)
                        EnergyImported = EnergyAllocated-EnergyAvailable;
                    else
                        EnergyImported = EnergyAllocated;


                }

                // determine how long it took to distribute the energy
                TimeDelta = System.currentTimeMillis() - InitTime.getTimeInMillis();
                DistrbutionCalander.setTimeInMillis(VirtualInit.getTimeInMillis()+TimeDelta);
                // now set the ETS time



                // record the time, total wind power available, wind power being imported space
heating, wind power being exported and if it is during the evening hours
                query = "INSERT INTO energy_stats (" +
                        "aDateTime," +
                        "aDistributionTime,"+
                        "WindEnergyAvailable,"+
                        "WindEnergyForSpaceHeating, "+
                        "ImportedEnergyForSpaceHeating, "+
                        "ExportedWindEnergy, "+
                        "EveningHours, "+
                        "DateString) "+
                        "VALUES ("+
                        SystemCalendar.getTimeInMillis()+","+
                        DistrbutionCalander.getTimeInMillis()+","+
                        EnergyAvailable+","+
                        EnergyAllocated+","+
                        EnergyImported+","+
                        EnergyExported+","+
                        EveningHours()+","+
                        "'"+dfm.format(SystemCalendar.getTime()) +"')";

                stmt.execute(query);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
```

```
        else
        {
            timer.cancel();
            textArea.append("Simulation complete");
        }
        //increment the system calendar
        dateChooser.setDate(SystemCalendar.getTime());
    }
}

/**
 * @return  1 if it is between 23:00 to 7:00, 0 otherwise
 */
public int EveningHours()
{
    int result=1;
    long DayStart;
    long DayEnd;
    // determine if we are in the 'auto recharge time';

    // get rid of the fractional days
    long Hour = SystemCalendar.get(Calendar.HOUR_OF_DAY);
    long Minute = SystemCalendar.get(Calendar.MINUTE);
    long Second = SystemCalendar.get(Calendar.SECOND);
    long Delta = Hour*60*60+Minute*60+Second;

    DayStart =  60 * 60 * 7;  // 7 am
    DayEnd =  60 * 60 * 23; // 11 pm

    if ((Delta > DayStart)&(Delta < DayEnd))
    {
        result = 0;
    }
    return result;
}

/**
 * Creates the database
 * @param conn
 *
 * This function creates a fresh database for the simulation.
 */
public void createDataBase(Connection conn)
{
    String query;
     Statement stmt;
     textArea.append("Creating database: ");

     try
     {
        query="create database 'test'";
         stmt = conn.createStatement();
         stmt.executeUpdate(query);
         query="use 'test'";
         stmt.execute(query);
         stmt.close();
         textArea.append("Complete \n");
     }
     catch (Exception e)
     {
        textArea.append("Exception:"+e.getMessage());
        e.printStackTrace();
     }
}

/**
 * Creates the database tables
 * @param conn
 */
public void createTables(Connection conn)
{
```

```
String query="empty";
Statement stmt;

try
{
    // delete the old table
    textArea.append("Dropping old tables:");
    query ="DROP TABLE IF EXISTS activedevices";
    stmt = conn.createStatement();
     stmt.execute(query);
     textArea.append("Complete \n");

     // create the authentication table
     textArea.append("Creating authentication table:");
    query ="DROP TABLE IF EXISTS auth_table";
    stmt.execute(query);
     // create a new copy
     query="CREATE TABLE auth_table ("+
       "DeviceID char(32) DEFAULT NULL,"+
       "Password char(32) DEFAULT NULL"+
       ") ENGINE=InnoDB DEFAULT CHARSET=latin1";
     stmt.execute(query);
     textArea.append("Complete \n");

     // create table for loads
     textArea.append("Creating system loads table:");
     query ="DROP TABLE IF EXISTS system_loads";
     stmt.execute(query);
     query ="CREATE TABLE system_loads ("+
       "DeviceID char(32) NOT NULL,"+
       "electrical_load double) ENGINE=InnoDB DEFAULT CHARSET=latin1";
     stmt.execute(query);
     textArea.append("Complete \n");
     textArea.append("Creating storage unit table:");

     // create producers table
     query ="DROP TABLE IF EXISTS producers";
     stmt.execute(query);
     query ="CREATE TABLE producers ("+
       "DeviceID char(32) NOT NULL,"+
       "output_kw double DEFAULT NULL,"+
       "PRIMARY KEY (DeviceID)"+
       ") ENGINE=InnoDB DEFAULT CHARSET=latin1";
     stmt.execute(query);
     textArea.append("Complete \n");

     // create storage_unit table
     textArea.append("Creating storage unit table:");
     query ="DROP TABLE IF EXISTS storage_units";
     stmt.execute(query);
     query ="CREATE TABLE storage_units ("+
       "DeviceID char(32) NOT NULL,"+
       "PushID int NOT NULL AUTO_INCREMENT,"+
       "energy_allocation double DEFAULT 0,"+
       "CurrentRecharge double DEFAULT 0,"+
       "MaxRecharge double DEFAULT 0,"+
       "MinRecharge double DEFAULT 0,"+
       "ExtTemp double DEFAULT 0,"+
       "PRIMARY KEY (PushID)"+
       ") ENGINE=InnoDB DEFAULT CHARSET=latin1";
     stmt.execute(query);
     textArea.append("Complete \n");

     // create emergency storage table
     textArea.append("Creating emergency recharge table:");
     query ="DROP TABLE IF EXISTS emergency_storage";
     stmt.execute(query);
     query ="CREATE TABLE emergency_storage ("+
       "aDateTime bigint,"+
       "DeviceID char(32) NOT NULL,"+
       "ResSize double,"+
      "EnergyIntensity double,"+
      "EmergencyRecharge double,"+
      "ExtTemp double) ENGINE=InnoDB DEFAULT CHARSET=latin1";
    stmt.execute(query);
    textArea.append("Complete \n");

    textArea.append("Creating wind data table:");
   if (CreateWindTable)
    {
       // create the wind data table
       query ="DROP TABLE IF EXISTS wind_data_table";
       stmt.execute(query);
       // create a new copy
       query="CREATE TABLE wind_data_table ("+
       "aDateString char (32),"+
       "aDateTime bigint,"+
       "PowerLoad double,"+
       "WestCape double,"+
       "Summerside double,"+
       "Temperature double"+
       ") ENGINE=InnoDB DEFAULT CHARSET=latin1";
       stmt.execute(query);
       textArea.append("Complete \n");
    }

// create the energy usage table
    query ="DROP TABLE IF EXISTS energy_usage";
   stmt.execute(query);
    // create a new copy
    query="CREATE TABLE energy_usage ("+
      "aDateTime bigint,"+
      "DeviceID char(32), "+
      "EnergyProduced double, "+
      "SystemLoad double,"+
     "EnergyAvailable double,"+
      "EnergyAllocatedBefore double, "+
      "EnergyAllocatedAfter double, "+
      "EnergyAllocatedToUnit double, "+
      "MinRecharge double, "+
      "MaxRecharge double, "+
      "CurrentRecharge double, "+
      "LastRecharge double, "+
      "CurrentStorage double, "+
      "CurrentDischarge double, "+
      "ExtTemp double,"+
      "DateString char (64), "+
      "RecordIndex int (4) NOT NULL AUTO_INCREMENT,"+
      "UNIQUE ( RecordIndex )"+
      ") ENGINE=InnoDB DEFAULT CHARSET=latin1";
    stmt.execute(query);
    textArea.append("Complete \n");
// create the energy usage table
    query ="DROP TABLE IF EXISTS energy_stats";
   stmt.execute(query);
    // create a new copy
    query="CREATE TABLE energy_stats ("+
      "aDateTime bigint,"+
      "aDistributionTime bigint,"+
      "WindEnergyAvailable double,"+
      "WindEnergyForSpaceHeating double, "+
      "ImportedEnergyForSpaceHeating double, "+
      "ExportedWindEnergy double, "+
      "EveningHours integer, "+
      "DateString char (64), "+
      "RecordIndex int (4) NOT NULL AUTO_INCREMENT,"+
      "UNIQUE ( RecordIndex )"+
      ") ENGINE=InnoDB DEFAULT CHARSET=latin1";
    stmt.execute(query);
    textArea.append("Complete \n");
    stmt.close();
```

```
        }
        catch (Exception e)
        {
           System.out.println(query);
           e.printStackTrace();
        }
    }

    /**Populates the database tables with values
     * @param conn
     */
    public void PopulateTables(Connection conn)
    {
        String query,str_i;
        Statement stmt;
        int count =0;
        int i;
        try
        {
           textArea.append("populating authentication table(adding ETS units):");
           count+=Integer.parseInt(textField.getText());
           count+=Integer.parseInt(textField_1.getText());
           count+=Integer.parseInt(textField_2.getText());
           count+=Integer.parseInt(textField_3.getText());
           count+=Integer.parseInt(textField_4.getText());
           count+=Integer.parseInt(textField_5.getText());
           count+=Integer.parseInt(textField_6.getText());
           count+=Integer.parseInt(textField_7.getText());


           // add devices to the auth table
           stmt = conn.createStatement();
           query = "insert into auth_table(DeviceID,Password) values ";
           for (i=0;i<count;i++)
           {
               str_i = Integer.toString(i);
               if (i!=0)
               {
                   query = query+",";
               }
               query = query+"('device"+str_i+"','password"+str_i+"')";
           }
           stmt.execute(query);
           textArea.append("complete\n");
           //now add our wind farm
           query = "insert into auth_table(DeviceID,Password) values
('SummerSide','SummerSide')";
           stmt.execute(query);
           query = "insert into auth_table(DeviceID,Password) values ('WestCape','WestCape')";
           stmt.execute(query);
           textArea.append("complete\n");
           // add the system load device

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    /**
     * Reasd the wind data from a CSV file and puts it in the database.
     * @param conn
     */
    private void ReadWindData(Connection conn)
    {
       String line = null;
       String adate = null;
       int i;
       double temperature;
       double load;
       double WestCapeOutput;
double SummerSideOutput;
Date DateTime=new Date();
String query;
 Statement stmt;
 long HoldLong =0;
 textArea.append("Loading Wind data into database:");
File file = new File("c:\\SummersideWindData.csv");
BufferedReader bufRdr;
try {
    bufRdr = new BufferedReader(new FileReader(file));
    DateFormat dfm = new SimpleDateFormat("MM/dd/yyyy HH:mm");
    dfm.setTimeZone(TimeZone.getTimeZone("Canada/Atlantic"));
    // the first 6 lines are header lines and have no data,
    try {
        for (i=0;i<=5;i++)
            line = bufRdr.readLine();
    } catch (IOException e1) {
        e1.printStackTrace();
    }
    i=0;
    //read each remaining line of text file
    try
    {
        while((line = bufRdr.readLine()) != null)
        {
            i++;
            if (i==1) textArea.append(".");
            i%=100;
            StringTokenizer st = new StringTokenizer(line,",");
            // each line has 18 values.  The first is the date and time
            adate = st.nextToken();
            try
            {
                DateTime = dfm.parse(adate);
            } catch (ParseException e)
            {
                e.printStackTrace();
            }
            // second value is the load
            load =Double.parseDouble(st.nextToken())*1000;
            // value 3 is the power output in kwh, from west cape
            WestCapeOutput = Double.parseDouble(st.nextToken())*1000;
            // value 4 is the output from Summerside
            SummerSideOutput = Double.parseDouble(st.nextToken())*1000;
            // value number 4 is the temperature
            temperature = Double.parseDouble(st.nextToken());
            // now add this data to the table
            HoldLong = DateTime.getTime();
            stmt = conn.createStatement();
            query = "Insert INTO wind_data_table(" +
                        "aDateString,"+
                        "aDateTime,"+
                        "PowerLoad,"+
                        "WestCape,"+
                        "SummerSide,"+
                        "Temperature) VALUES ('"+
                        adate+"',"+
                        Long.toString(HoldLong)+","+
                        Double.toString(load)+","+
                        Double.toString(WestCapeOutput)+","+
                        Double.toString(SummerSideOutput)+","+
                        Double.toString(temperature)+");";
            stmt.execute(query);
        }
    } catch (NumberFormatException e) {
        System.out.println("Error in line " +line);
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
```

```
            }
            textArea.append("complete\n");
            bufRdr.close();

            } catch (FileNotFoundException e2) {
                e2.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    /**
     * Creates the ETS units based on the parameters set by the user
     */
    private void CreateETSUnits()
    {
        SimETS ThisETS;
        int i;
        int aDeviceID =0;
        int aCount = 0;
        double ScaleFactor = 0;
        double ResSize =0;
        DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        dfm.setTimeZone(TimeZone.getTimeZone("Canada/Atlantic"));
        try {
            StartTime = dfm.parse(Start);
            EndTime = dfm.parse(End);
        } catch (ParseException e1) {
            e1.printStackTrace();
        }

        textArea.append("Creating ETS units");
        // we need to create ETS units based upon the size of the residence and its vintage

        // pre 1946
        ScaleFactor = 1.408;
        ResSize = 110.1;
        aCount = Integer.parseInt(textField.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 37.2,14.4,
180,ResSize,ScaleFactor, 19);
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 1946-1960
        ScaleFactor = 1.565;
        ResSize = 128.8;
        aCount = Integer.parseInt(textField_1.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 37.2,14.4,
180,ResSize,ScaleFactor, 19);
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 1961-1977
        ScaleFactor = 0.880;
        ResSize = 125.6;
        aCount = Integer.parseInt(textField_2.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 24.8,10,
120,ResSize,ScaleFactor, 19);
```

```
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 1978-1983
        ScaleFactor = 0.782;
        ResSize = 122.1;
        aCount = Integer.parseInt(textField_3.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 24.8,10,
120,ResSize,ScaleFactor, 19);
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 1984-1995
        ScaleFactor = 0.618;
        ResSize = 132;
        aCount = Integer.parseInt(textField_4.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 24.8,10,
120,ResSize,ScaleFactor, 19);
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 1996-2000
        ScaleFactor = 0.587;
        ResSize = 137.6;
        aCount = Integer.parseInt(textField_5.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 24.8,10,
120,ResSize,ScaleFactor, 19);
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 2001-2005
        ScaleFactor = 0.469;
        ResSize = 147.5;
        aCount = Integer.parseInt(textField_6.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 24.8,10,
120,ResSize,ScaleFactor, 19);
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        // 2006-2008
        ScaleFactor = 0.391;
        ResSize = 146.1;
        aCount = Integer.parseInt(textField_7.getText());
        for (i=0;i<aCount;i++)
        {
            textArea.append(".");
            ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 24.8,10,
120,ResSize,ScaleFactor, 19);
```

```
            //ThisETS = new SimETS("Device" + aDeviceID, "Password" + aDeviceID, 38.4,18.4,
240,ResSize,ScaleFactor, 19); // 4140 model
            ThisETS.InitTimeStamp(StartTime,EndTime);
            ETSList.add(ThisETS);
            aDeviceID++;
        }

        textArea.append("complete\n");
    }

    /**
     * creates the wind farms used by the simulation
     */
    private void CreateWindFarms()
    {
        West_Cape = new SimWindFarm("WestCape","WestCape");
        DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        dfm.setTimeZone(TimeZone.getTimeZone("Canada/Atlantic"));
        textArea.append("Creating wind farms:");
        try {
            StartTime = dfm.parse(Start);
            EndTime = dfm.parse(End);
            West_Cape.InitTimeStamp(StartTime,EndTime);
            textArea.append("complete\n");
        } catch (ParseException e) {
            textArea.append("Exception:"+e.getMessage()+"\n");
            e.printStackTrace();
        }
        SummerSide = new SimWindFarm("SummerSide","SummerSide");
        dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        dfm.setTimeZone(TimeZone.getTimeZone("Canada/Atlantic"));
        textArea.append("Creating wind farms:");
        try {
            StartTime = dfm.parse(Start);
            EndTime = dfm.parse(End);
            SummerSide.InitTimeStamp(StartTime,EndTime);
            textArea.append("complete\n");
        } catch (ParseException e) {
            textArea.append("Exception:"+e.getMessage()+"\n");
            e.printStackTrace();
        }

    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize()
    {
        // create the connection to the database
        try {
            Class.forName(Config.DriverClass).newInstance();
            con = DriverManager.getConnection(Config.dBaseName,
                        Config.dBaseUserName, Config.dBasePassword);
            stmt = con.createStatement();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");


        timer = new Timer();
        frame = new JFrame();
        frame.setBounds(100, 100, 593, 647);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        GridBagLayout gridBagLayout = new GridBagLayout();
        gridBagLayout.columnWidths = new int[]{10, 548, 10, 0};
        gridBagLayout.rowHeights = new int[]{10, 289, 253, 0};
        gridBagLayout.columnWeights = new double[]{1.0, 0.0, 0.0, Double.MIN_VALUE};
        gridBagLayout.rowWeights = new double[]{0.0, 1.0, 0.0, Double.MIN_VALUE};
        frame.getContentPane().setLayout(gridBagLayout);

        DateFormat HoldDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        HoldDate.setTimeZone(TimeZone.getTimeZone("Canada/Atlantic"));

        JPanel panel = new JPanel();
        GridBagConstraints gbc_panel = new GridBagConstraints();
        gbc_panel.anchor = GridBagConstraints.NORTH;
        gbc_panel.insets = new Insets(0, 0, 5, 5);
        gbc_panel.fill = GridBagConstraints.HORIZONTAL;
        gbc_panel.gridx = 1;
        gbc_panel.gridy = 1;
        frame.getContentPane().add(panel, gbc_panel);
        GridBagLayout gbl_panel = new GridBagLayout();
        gbl_panel.columnWidths = new int[]{0, 0, 21, 134, 26, 0, 82, 0};
        gbl_panel.rowHeights = new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        gbl_panel.columnWeights = new double[]{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
Double.MIN_VALUE};
        gbl_panel.rowWeights = new double[]{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, Double.MIN_VALUE};
        panel.setLayout(gbl_panel);

        JLabel lblVintage = new JLabel("Vintage");
        lblVintage.setFont(new Font("Tahoma", Font.BOLD, 11));
        GridBagConstraints gbc_lblVintage = new GridBagConstraints();
        gbc_lblVintage.insets = new Insets(0, 0, 5, 5);
        gbc_lblVintage.gridx = 1;
        gbc_lblVintage.gridy = 2;
        panel.add(lblVintage, gbc_lblVintage);

        JLabel lblNumberOfResidences = new JLabel("Number of residences");
        lblNumberOfResidences.setFont(new Font("Tahoma", Font.BOLD, 11));
        GridBagConstraints gbc_lblNumberOfResidences = new GridBagConstraints();
        gbc_lblNumberOfResidences.insets = new Insets(0, 0, 5, 5);
        gbc_lblNumberOfResidences.gridx = 3;
        gbc_lblNumberOfResidences.gridy = 2;
        panel.add(lblNumberOfResidences, gbc_lblNumberOfResidences);

        JLabel lblBefore = new JLabel("Before 1946");
        GridBagConstraints gbc_lblBefore = new GridBagConstraints();
        gbc_lblBefore.anchor = GridBagConstraints.BELOW_BASELINE_TRAILING;
        gbc_lblBefore.insets = new Insets(0, 0, 5, 5);
        gbc_lblBefore.gridx = 1;
        gbc_lblBefore.gridy = 3;
        panel.add(lblBefore, gbc_lblBefore);

        textField = new JTextField();
        textField.setHorizontalAlignment(SwingConstants.CENTER);
        textField.setText("13");
        textField.setColumns(10);
        GridBagConstraints gbc_textField = new GridBagConstraints();
        gbc_textField.fill = GridBagConstraints.HORIZONTAL;
        gbc_textField.insets = new Insets(0, 0, 5, 5);
        gbc_textField.gridx = 3;
        gbc_textField.gridy = 3;
        panel.add(textField, gbc_textField);

        JLabel lblSimulationStart = new JLabel("Simulation start");
        GridBagConstraints gbc_lblSimulationStart = new GridBagConstraints();
        gbc_lblSimulationStart.insets = new Insets(0, 0, 5, 5);
        gbc_lblSimulationStart.gridx = 5;
        gbc_lblSimulationStart.gridy = 3;
```

```
panel.add(lblSimulationStart, gbc_lblSimulationStart);
lblSimulationStart.setHorizontalAlignment(SwingConstants.LEFT);
try {
    dcStart = new JDateChooser(HoldDate.parse(Start));
} catch (ParseException e1) {
    e1.printStackTrace();
}
GridBagConstraints gbc_dcStart = new GridBagConstraints();
gbc_dcStart.fill = GridBagConstraints.HORIZONTAL;
gbc_dcStart.insets = new Insets(0, 0, 5, 0);
gbc_dcStart.gridx = 6;
gbc_dcStart.gridy = 3;
panel.add(dcStart, gbc_dcStart);
dcStart.setDateFormatString("yyyy-MM-dd HH:mm:ss");

JLabel label = new JLabel("1946-1960");
GridBagConstraints gbc_label = new GridBagConstraints();
gbc_label.anchor = GridBagConstraints.EAST;
gbc_label.insets = new Insets(0, 0, 5, 5);
gbc_label.gridx = 1;
gbc_label.gridy = 4;
panel.add(label, gbc_label);

textField_1 = new JTextField();
textField_1.setHorizontalAlignment(SwingConstants.CENTER);
textField_1.setText("3");
textField_1.setColumns(10);
GridBagConstraints gbc_textField_1 = new GridBagConstraints();
gbc_textField_1.fill = GridBagConstraints.HORIZONTAL;
gbc_textField_1.insets = new Insets(0, 0, 5, 5);
gbc_textField_1.gridx = 3;
gbc_textField_1.gridy = 4;
panel.add(textField_1, gbc_textField_1);

JLabel lblSimulationEnd = new JLabel("Simulation end");
GridBagConstraints gbc_lblSimulationEnd = new GridBagConstraints();
gbc_lblSimulationEnd.insets = new Insets(0, 0, 5, 5);
gbc_lblSimulationEnd.gridx = 5;
gbc_lblSimulationEnd.gridy = 4;
panel.add(lblSimulationEnd, gbc_lblSimulationEnd);
lblSimulationEnd.setHorizontalAlignment(SwingConstants.LEFT);
try {
    dcEnd = new JDateChooser(HoldDate.parse(End));
} catch (ParseException e1) {
    e1.printStackTrace();
}
GridBagConstraints gbc_dcEnd = new GridBagConstraints();
gbc_dcEnd.insets = new Insets(0, 0, 5, 0);
gbc_dcEnd.gridx = 6;
gbc_dcEnd.gridy = 4;
panel.add(dcEnd, gbc_dcEnd);
dcEnd.setDateFormatString("yyyy-MM-dd HH:mm:ss");

JLabel label_3 = new JLabel("1961-1977");
GridBagConstraints gbc_label_3 = new GridBagConstraints();
gbc_label_3.anchor = GridBagConstraints.EAST;
gbc_label_3.insets = new Insets(0, 0, 5, 5);
gbc_label_3.gridx = 1;
gbc_label_3.gridy = 5;
panel.add(label_3, gbc_label_3);

textField_2 = new JTextField();
textField_2.setHorizontalAlignment(SwingConstants.CENTER);
textField_2.setText("7");
textField_2.setColumns(10);
GridBagConstraints gbc_textField_2 = new GridBagConstraints();
gbc_textField_2.fill = GridBagConstraints.HORIZONTAL;
gbc_textField_2.insets = new Insets(0, 0, 5, 5);
gbc_textField_2.gridx = 3;
gbc_textField_2.gridy = 5;
panel.add(textField_2, gbc_textField_2);
```

```
JLabel label_4 = new JLabel("1978-1983");
GridBagConstraints gbc_label_4 = new GridBagConstraints();
gbc_label_4.anchor = GridBagConstraints.EAST;
gbc_label_4.insets = new Insets(0, 0, 5, 5);
gbc_label_4.gridx = 1;
gbc_label_4.gridy = 6;
panel.add(label_4, gbc_label_4);

textField_3 = new JTextField();
textField_3.setHorizontalAlignment(SwingConstants.CENTER);
textField_3.setText("5");
textField_3.setColumns(10);
GridBagConstraints gbc_textField_3 = new GridBagConstraints();
gbc_textField_3.fill = GridBagConstraints.HORIZONTAL;
gbc_textField_3.insets = new Insets(0, 0, 5, 5);
gbc_textField_3.gridx = 3;
gbc_textField_3.gridy = 6;
panel.add(textField_3, gbc_textField_3);


JLabel label_5 = new JLabel("1984-1995");
GridBagConstraints gbc_label_5 = new GridBagConstraints();
gbc_label_5.anchor = GridBagConstraints.EAST;
gbc_label_5.insets = new Insets(0, 0, 5, 5);
gbc_label_5.gridx = 1;
gbc_label_5.gridy = 7;
panel.add(label_5, gbc_label_5);

textField_4 = new JTextField();
textField_4.setHorizontalAlignment(SwingConstants.CENTER);
textField_4.setText("15");
textField_4.setColumns(10);
GridBagConstraints gbc_textField_4 = new GridBagConstraints();
gbc_textField_4.fill = GridBagConstraints.HORIZONTAL;
gbc_textField_4.insets = new Insets(0, 0, 5, 5);
gbc_textField_4.gridx = 3;
gbc_textField_4.gridy = 7;
panel.add(textField_4, gbc_textField_4);

JLabel label_6 = new JLabel("1996-2000");
GridBagConstraints gbc_label_6 = new GridBagConstraints();
gbc_label_6.anchor = GridBagConstraints.EAST;
gbc_label_6.insets = new Insets(0, 0, 5, 5);
gbc_label_6.gridx = 1;
gbc_label_6.gridy = 8;
panel.add(label_6, gbc_label_6);

textField_5 = new JTextField();
textField_5.setHorizontalAlignment(SwingConstants.CENTER);
textField_5.setText("6");
textField_5.setColumns(10);
GridBagConstraints gbc_textField_5 = new GridBagConstraints();
gbc_textField_5.fill = GridBagConstraints.HORIZONTAL;
gbc_textField_5.insets = new Insets(0, 0, 5, 5);
gbc_textField_5.gridx = 3;
gbc_textField_5.gridy = 8;
panel.add(textField_5, gbc_textField_5);

JLabel label_7 = new JLabel("2001-2005");
GridBagConstraints gbc_label_7 = new GridBagConstraints();
gbc_label_7.anchor = GridBagConstraints.EAST;
gbc_label_7.insets = new Insets(0, 0, 5, 5);
gbc_label_7.gridx = 1;
gbc_label_7.gridy = 9;
panel.add(label_7, gbc_label_7);

textField_6 = new JTextField();
textField_6.setHorizontalAlignment(SwingConstants.CENTER);
textField_6.setText("7");
textField_6.setColumns(10);
```

```
        GridBagConstraints gbc_textField_6 = new GridBagConstraints();
        gbc_textField_6.fill = GridBagConstraints.HORIZONTAL;
        gbc_textField_6.insets = new Insets(0, 0, 5, 5);
        gbc_textField_6.gridx = 3;
        gbc_textField_6.gridy = 9;
        panel.add(textField_6, gbc_textField_6);

        JLabel label_8 = new JLabel("2006-2008");
        GridBagConstraints gbc_label_8 = new GridBagConstraints();
        gbc_label_8.anchor = GridBagConstraints.EAST;
        gbc_label_8.insets = new Insets(0, 0, 5, 5);
        gbc_label_8.gridx = 1;
        gbc_label_8.gridy = 10;
        panel.add(label_8, gbc_label_8);

        textField_7 = new JTextField();
        textField_7.setHorizontalAlignment(SwingConstants.CENTER);
        textField_7.setText("4");
        textField_7.setColumns(10);
        GridBagConstraints gbc_textField_7 = new GridBagConstraints();
        gbc_textField_7.fill = GridBagConstraints.HORIZONTAL;
        gbc_textField_7.insets = new Insets(0, 0, 5, 5);
        gbc_textField_7.gridx = 3;
        gbc_textField_7.gridy = 10;
        panel.add(textField_7, gbc_textField_7);

        JLabel lblSimulationTime = new JLabel("Simulation time");
        lblSimulationTime.setHorizontalAlignment(SwingConstants.LEFT);
        GridBagConstraints gbc_lblSimulationTime = new GridBagConstraints();
        gbc_lblSimulationTime.insets = new Insets(0, 0, 5, 5);
        gbc_lblSimulationTime.gridx = 5;
        gbc_lblSimulationTime.gridy = 10;
        panel.add(lblSimulationTime, gbc_lblSimulationTime);

        dateChooser = new JDateChooser((Date) null);
        dateChooser.setDateFormatString("yyyy-MM-dd HH:mm:ss");
        GridBagConstraints gbc_dateChooser = new GridBagConstraints();
        gbc_dateChooser.insets = new Insets(0, 0, 5, 0);
        gbc_dateChooser.fill = GridBagConstraints.BOTH;
        gbc_dateChooser.gridx = 6;
        gbc_dateChooser.gridy = 10;
        panel.add(dateChooser, gbc_dateChooser);

        JPanel panel_2 = new JPanel();
        GridBagConstraints gbc_panel_2 = new GridBagConstraints();
        gbc_panel_2.anchor = GridBagConstraints.NORTH;
        gbc_panel_2.fill = GridBagConstraints.HORIZONTAL;
        gbc_panel_2.gridwidth = 2;
        gbc_panel_2.gridx = 0;
        gbc_panel_2.gridy = 2;
        frame.getContentPane().add(panel_2, gbc_panel_2);
        GridBagLayout gbl_panel_2 = new GridBagLayout();
        gbl_panel_2.columnWidths = new int[]{16, 46, 99, 0, 104, 167, 0, 0};
        gbl_panel_2.rowHeights = new int[]{0, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0};
        gbl_panel_2.columnWeights = new double[]{0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
Double.MIN_VALUE};
        gbl_panel_2.rowWeights = new double[]{0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
Double.MIN_VALUE};
        panel_2.setLayout(gbl_panel_2);
        textArea = new JTextArea();
        GridBagConstraints gbc_textArea = new GridBagConstraints();
        gbc_textArea.gridwidth = 6;
        gbc_textArea.gridheight = 8;
        gbc_textArea.insets = new Insets(0, 0, 5, 5);
        gbc_textArea.fill = GridBagConstraints.BOTH;
        gbc_textArea.gridx = 1;
        gbc_textArea.gridy = 0;
        panel_2.add(textArea, gbc_textArea);
        textArea.setEditable(false);

        JButton btnStart = new JButton("Start");
        btnStart.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                // Initialize the database
                createTables(con);
                PopulateTables(con);

                // the wind data must be parsed from a comma separated file
                if (CreateWindTable) ReadWindData(con);

                // Create the ETS units
                CreateETSUnits();

                // create the wind farm
                CreateWindFarms();

                // create the system load
                SystemLoad = new SimSystemLoad();

                // create a wind distributor
                Distributor = new WindDistributor();

                DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                dfm.setTimeZone(TimeZone.getTimeZone("Canada/Atlantic"));
                textArea.append("Creating wind farms:");

                try {
                    StartTime = dfm.parse(Start);
                    EndTime = dfm.parse(End);

                    SystemLoad.InitTimeStamp(StartTime,EndTime);
                    TimeZone zone = TimeZone.getTimeZone("Canada/Atlantic");
                    SystemCalendar  = new GregorianCalendar(zone);
                    SystemCalendar.setTime(StartTime);
                    SystemCalendar.setTimeInMillis(SystemCalendar.getTimeInMillis()-
Config.TimeIncrement);
                    InitTime = new GregorianCalendar(zone);
                    VirtualInit= new GregorianCalendar(zone);
                    VirtualInit.setTime(StartTime);
                    textArea.append("complete\n");
                } catch (ParseException ex) {
                    textArea.append("Exception:"+ex.getMessage()+"\n");
                    ex.printStackTrace();
                }
                aListner = new UDPListner();
                new Thread(aListner). start ( );

                timer.schedule(new tick_task(), 10000,50);
            }
        });
        GridBagConstraints gbc_btnStart = new GridBagConstraints();
        gbc_btnStart.anchor = GridBagConstraints.EAST;
        gbc_btnStart.insets = new Insets(0, 0, 5, 5);
        gbc_btnStart.gridx = 5;
        gbc_btnStart.gridy = 8;
        panel_2.add(btnStart, gbc_btnStart);
    }
}

// servlet class used for storage device registration and reporting
package net.wattbox.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
//import java.text.DateFormat;
import java.text.DecimalFormat;
//import java.text.SimpleDateFormat;
import java.util.GregorianCalendar;
import java.util.List;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class GetState
 */
@WebServlet(description = "Gets the state of an ETS device from the server", urlPatterns = {
"/GetState" })
public class GetState extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    private static Connection con = null;

    PrintWriter out;
    public GetState() {
        super();
    }

    /**
     * @see Servlet#init(ServletConfig)
     */
    public void init(ServletConfig config) throws ServletException {
        System.out.println("Storage registration :Init");
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
             con = DriverManager.getConnection("jdbc:mysql:///WattBoxPush","root", "cuc002");
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
/**
 * @param d
 * @return rounds a decimal number to two decimal points
 */
double roundTwoDecimals(double d) {
    DecimalFormat twoDForm = new DecimalFormat("#.##");
    return Double.valueOf(twoDForm.format(d));
}
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE html>\n" +
            "<html>\n" +
            "<head><title>GetState</title></head>\n" +
            "<body bgcolor=\"#fdf5e6\">\n" +
            "<h1>Test</h1>\n" +
            "<p>Applet running</p>\n" +
            "</body></html>");
}

/**
```

```
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    boolean valid_payload = true;

    response.setContentType("text/html");
    out = response.getWriter();

    if(Debug.ON)   out.println("Storage Registration start");
    // read the HTML payload data
    String deviceID = request.getParameter("deviceid");
    String DevicePW = request.getParameter("password");
    String maximum_recharge = request.getParameter("MaxRecharge");
    String minimum_recharge = request.getParameter("MinRecharge");
    String current_recharge = request.getParameter("CurrentRecharge");
    String last_recharge = request.getParameter("LastRecharge");
    String current_storage = request.getParameter("CurrentStorage");
    String current_discharge = request.getParameter("CurrentDischarge");
    String ext_temp = request.getParameter("ExtTemp");
    String action = request.getParameter("action");
    String aTime = request.getParameter("PostTime");

    if(Debug.ON)   out.println("Validating payload");
    // validate the payload
    ParamValidation test = new ParamValidation();
    if (!test.validate_double(maximum_recharge,"MaxRecharge")) valid_payload = false;
    if (!test.validate_double(minimum_recharge,"MinRecharge")) valid_payload = false;
    if (!test.validate_double(last_recharge,"LastRecharge")) valid_payload = false;
    if (!test.validate_double(current_recharge,"CurrentRecharge")) valid_payload = false;
    if (!test.validate_double(current_storage,"CurrentStorage")) valid_payload = false;
    if (!test.validate_double(current_discharge,"CurrentDischarge")) valid_payload = false;


    if (!valid_payload)
    {
        // an error occurred during validation.  Return the validation results
        List<String> ThisList = test.getStringOutput();
        for (String error_string: ThisList)
        {
            out.println(error_string);
        }
    }
    else
    {
        // we have a valid payload, authenticate the device
        //double LastRecharge = roundTwoDecimals(Double.parseDouble(last_recharge));
        //double MinRecharge = roundTwoDecimals(Double.parseDouble(minimum_recharge));
        //double MaxRecharge = roundTwoDecimals(Double.parseDouble(maximum_recharge));
        double CurrentRecharge = roundTwoDecimals(Double.parseDouble(current_recharge));
        //double CurrentStorage = roundTwoDecimals(Double.parseDouble(current_storage));
        //double CurrentDischarge = roundTwoDecimals(Double.parseDouble(current_discharge));
        int PushID = -1;
        // lock the database

        String query = "SELECT password FROM auth_table WHERE deviceid = '"+deviceID+"' and
password = '"+DevicePW+"';";
        try
        {
            Statement stmt = con.createStatement();
            Statement astmt = con.createStatement();
            //synchronized(this)
            {
                //stmt.execute("LOCK TABLE auth_table WRITE, storage_units WRITE;");
                ResultSet rs = astmt.executeQuery(query);
                if (rs.next())// authenticated
                {
                    if(Debug.ON)   out.println("Authenticated");
                    if (action.equals("register"))
                    {
```

```
                    query = "SELECT PushID from storage_units where DeviceID =
'"+deviceID+"';";
                    rs = stmt.executeQuery(query);
                    if (rs.next())
                    {
                        // device exits, update the current record
                        PushID =rs.getInt("PushID");
                        query = "UPDATE storage_units SET MinRecharge="+minimum_recharge+","+
                                                      "MaxRecharge="+maximum_recharge+","+
                                                      "CurrentRecharge="+CurrentRecharge+","+
                                                      "ExtTemp = "+ext_temp+" where DeviceID =
'"+deviceID+"';";

                        stmt.executeUpdate(query);
                    }

                    else
                    {
                        // this unit does not exist in the table, add it.
                        query = "INSERT storage_units
(DeviceID,PushID,energy_allocation,MinRecharge,MaxRecharge,CurrentRecharge,ExtTemp)
VALUES('"+deviceID+"',0,0,"+minimum_recharge+","+maximum_recharge+","+CurrentRecharge+","+ext_tem
p+");";
                        stmt.execute(query);
                        query = "SELECT PushID from storage_units where DeviceID =
'"+deviceID+"';";
                        rs = stmt.executeQuery(query);
                        if (rs.next())
                            PushID =rs.getInt("PushID");
                    }

                    // record energy stats for analysis
                    query = "INSERT energy_usage (aDateTime," +
                     "DeviceID," +
                     "EnergyAllocatedBefore," +
                     "EnergyAllocatedAfter,"+
                     "EnergyAllocatedToUnit," +
                     "MinRecharge," +
                     "MaxRecharge," +
                     "CurrentRecharge,"+
                     "CurrentStorage,"+
                     "CurrentDischarge,"+
                     "EnergyProduced," +
                     "SystemLoad,"+
                     "EnergyAvailable,"+
                     "LastRecharge,"+
                     "ExtTemp,"+
                     "DateString)" +
                    " VALUES" +
                    "("+new GregorianCalendar().getTimeInMillis()+","+
                     "'"+deviceID+"',"+
                     0+","+
                     0+","+
                     CurrentRecharge+","+
                     minimum_recharge+","+
                     maximum_recharge+","+
                     CurrentRecharge+","+
                     current_storage+","+
                     current_discharge+","+
                     0+","+
                     0+","+
                     0+","+
                     0+","+
                     ext_temp+",'"+
                     aTime+"')";

                    stmt.execute(query);
                }
                else
                {
                    query = "DELETE from storage_units where DeviceID = '"+deviceID+"';";
```

```
                        stmt.execute(query);
                    }
                }
                stmt.execute("commit;");
                //stmt.execute("UNLOCK TABLES;");
                rs.close();
            }

            stmt.close();
            astmt.close();
            out.println("PushID="+PushID);
            //System.out.println("SetRecharge="+EnergyToAllocate);
        } catch (SQLException e) {
            if(Debug.ON) out.println("SQLException");
            e.printStackTrace();
        }
    }
    if(Debug.ON) out.println("Registration End");

    //if (out!=null) out.close();
    }
}
```