

# **Design and Implementation of a Cloud-Based Building Energy Monitoring System**

A THESIS SUBMITTED TO THE GRADUATE DIVISION IN PARTIAL  
FULFILLMENT OF THE UNIVERSITY OF HAWAI'I AT MĀNOA  
REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

DECEMBER 2012

By

Hao Liu

Thesis Committee:

Xiangrong Zhou, Chairperson  
Yingfei Dong  
Zhengqing Yun

Keywords: building energy monitoring, cloud computing, embedded Linux,  
wireless sensor network

## **Abstract**

Energy efficiency and renewable energy are said to be the twin pillars of a sustainable energy policy, both of which should be developed simultaneously in order to reduce carbon dioxide emissions. In this thesis, we design and integrate a cloud based energy monitoring system for buildings so that people such as commercial managers or residential building occupants can make policies accordingly regard to improving the efficiency of energy usage.

This system consists of three major components: wireless sensor network, embedded Linux gateway and cloud computing component. The wireless sensor network achieves the function of retrieving energy consumption data on the socket level without comprising the existing power supply infrastructure. The embedded Linux gateway reads data from to the wireless sensor network and delivering them to the cloud. The cloud process the raw energy data, store the processed data to a database and displaying the power consumption using a web server for visualization.

**Keywords:** energy monitoring, cloud computing, embedded Linux, wireless sensor network.

## Table of Contents

Abstract.....	i
List of Figures.....	iii
Chapter 1 Introduction.....	1
Chapter 2 Related work.....	4
Chapter 3 System Design.....	7
3.1 Energy usage information acquisition design.....	7
3.2 Energy data processing and visualization design.....	9
3.3 Connectivity design between a wireless sensor network and a cloud.....	10
3.4 Overall System Architecture.....	11
Chapter 4 System Implementation.....	13
4.1 Implementation of data acquisition subsystem.....	13
4.2 Implementation of application specific gateway.....	17
4.2.1 Hardware platform.....	17
4.2.2 Operating System-Ångström.....	18
4.2.3 System Configurations and Application Program.....	19
4.2.3.1 Java Runtime Environment.....	19
4.2.3.2 Serial Port Listener.....	20
4.2.3.3 TCP Client Socket.....	22
4.2.3.4 SSH tunneling.....	23
4.3 Implementation of cloud computing subsystem.....	24
4.3.1 Cloud Computing Platform.....	24
4.3.2 System Set up and Application program.....	25
4.3.2.1 LAMP.....	25
4.3.2.2 Server Socket Programming.....	26
4.3.2.3 Web Page Design.....	29
Chapter 5 System test.....	32
5.1 Short-term functionality test.....	32
5.2 Robustness functionality test.....	34
Chapter 6 Conclusion and Future Work.....	37
References.....	38

## List of Figures

<b>Figure</b>	<b>Page</b>
Figure 1.1 Percentage share of total U.S. electricity generation by various resources.....	1
Figure 3.1 System Architecture for building energy monitoring system.....	13
Figure 4.1 ACme Application Component Illustration.....	13
Figure 4.2 Hardware overview of beagleboard-Xm.....	13
Figure 4.3 Sample output of the Java Listener Figure 3.1 System Architecture for building energy monitoring system.....	22
Figure 4.4 EC2 Management Console.....	25
Figure 4.5 Illustration of database table construction.....	27
Figure 4.6 Illustration of database table record.....	29
Figure 4.7 Web interface for short-time power monitoring.....	30
Figure 4.8 Web interface for daily power monitoring.....	31
Figure 5.1 Short-term power data of GR-820 GOLDSTAR freezer.....	13
Figure 5.2 Short-term power data of Kenmore microwave oven.....	33
Figure 5.3 Short-term power data of Gateway E series desktop computer.....	33
Figure 5.4 Daily Power data of Gateway E series desktop on 01/17/2012.....	34
Figure 5.5 Daily Power data of Gateway E series desktop on 01/18/2012.....	35
Figure 5.6 Daily power data of GR-820 GOLDSTAR refrigerator on 01/20/2012.....	35
Figure 5.7 Daily power data of ThinkPad T420 laptop on 01/17/2012.....	36
Figure 5.8 Daily power data of application level router on 01/18/2012.....	36

## Chapter 1 Introduction

With the fast increasing worldwide population and the subsequent increasing demand for energy which will be primarily met by fossil fuel for economic and technological reasons, the limited fossil fuel will be depleted soon. Several studies have estimated that the crude oil reserves will come to an end between year 2050 [1], the coal reserves will reach an end by year 2230 and the natural gas reserves will be depleted by year 2166 [2]. In order to meet continuous development needs, efforts must be made to slow down the fossil fuel depletion. To efficiently decrease the fossil fuel usage, attention should be focused on the major consuming factor that consumes the most fossil fuel.

More than two third of the fossil fuel such as coal, natural gas and oil currently consumed worldwide are utilized to generate electricity, and hence electricity generation is the key to deal with the fossil fuel depletion crisis. There are generally two methods to reduce the usage of fossil fuel consumed to generate electricity.

One method is to substitute the fossil fuel to non-fossil fuel resources such as nuclear, hydropower and renewable energy. Figure 1.1 below shows the percentage share of U.S. electricity generation by different resources since year 2004. From the figure, we can see that the percentage share of nuclear and hydropower electricity generation remains steady over years. Renewable energy electricity generation achieves an increasing over the past few years, but still accounts for a trivial share of the total energy generation. Around 70% of the total electricity generation is still produced by traditional fossil fuel.

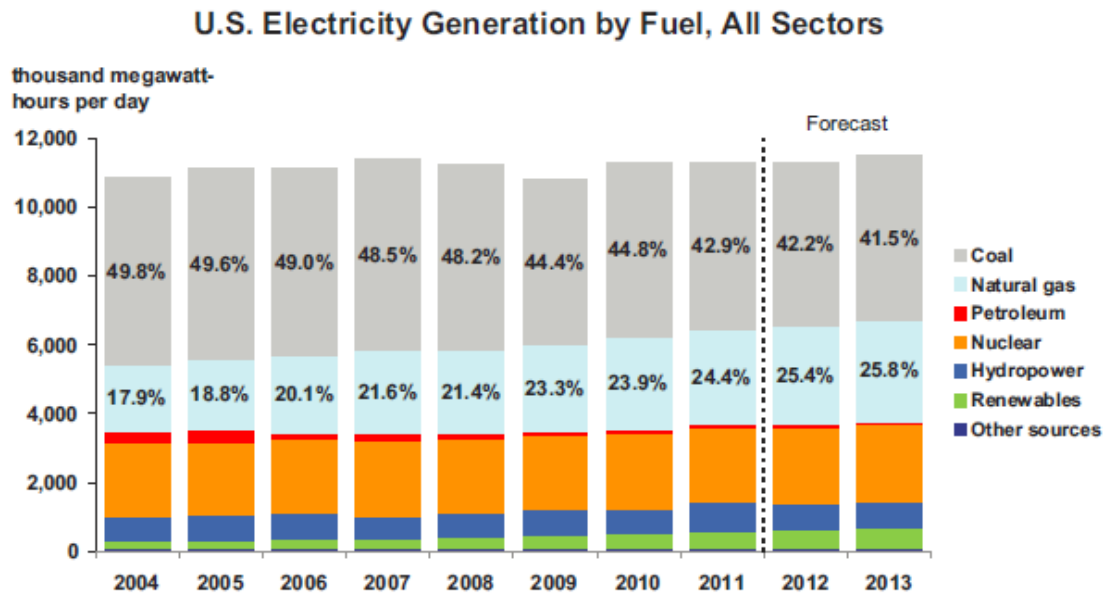


Figure 1.1 Percentage share of total U.S. electricity generation by various resources.

Another method is to reduce the overall electricity generation and hence the fossil fuel consumption will be decreased accordingly. To reduce the overall electricity generation, attention should be paid to the primary electricity energy consuming component. According to U.S. Department of Energy Electricity Consumption report, 72% of the total U.S. electricity consumption occurs in residential and commercial buildings [3] and 30% of energy consumed in both commercial and residential buildings is wasted [4]. The wasted energy consumption in commercial and residential buildings accounts for 27% of the total U.S. electricity consumption, which if reduced effectively, will greatly cut down the overall electricity generation needed and hence tremendously decrease the fossil fuel consumption.

One of the most effective methods to cut down the waste in electricity energy consumption in buildings is to employ a building energy monitoring system. A building energy monitoring system can make the detailed energy consumption data available to

building occupants and building managers, so that they can probe into the details of energy usage pattern and detect where and when energy waste occurred. Then people can make policy accordingly to improve their energy usage efficiency. The preliminary results from a “pilot program” carried by IBM show that, by utilizing the real-time power consumption data, the strongly engaged residents in the City of Dubuque, Iowa, could save energy up to 11% [5]. Also, a research shows that the introduction of visual feedback power monitoring systems to college dormitory resulted in a 32 percent reduction in electricity use when the college students are educated and incented [6].

In this thesis, we have built a cloud based building energy monitoring system. We primarily contributed to the design, integration and implementation of the energy monitoring system.

In Chapter 2, we will review the related work. Chapter 3 shows the design of the energy monitoring system. Chapter 4 illustrates the integration and implementation of the monitoring system. Chapter 5 demonstrates the system’s functionality and robustness. Chapter 6 concludes this thesis and discusses future work.

## Chapter 2 Related work

There are generally two approaches to design the building energy monitoring system. One is the traditional top-down approach which involves putting the sensing equipment at the root of the power distribution network. Algorithm is utilized to increase the visibility of the power consumption for individual types of appliances by analyzing an aggregated load/root of the power distribution network. Some utility companies have adopted this approach to offer the “bill disaggregation” service which breaks down user’s monthly bill by extracting the energy consumption for different types of loads from the aggregated energy source. This approach was first proposed by Hart in 1995. He proposed disaggregating individual electrical loads by conducting real and reactive power measurements [7]. However, this approach is only suitable and feasible for a small amount of loads that the power factors are distinguishable from each other. Even though more algorithms are developed and have shown improvement with respect to better disaggregating the individual electrical loads from aggregated energy source, this top-down approach is generally less effective in some environments where the load factor of appliances is similar.

Recently intensive research and industrial endeavors have been conducted to increase the visibility of the power information. Instead of the traditional top-down approach, recently researches have proposed a bottom-up approach by attaching sensors to individual appliances to monitor the detailed power profile of each appliance, which bears more capability of power information visibility [8, 9]. Some start-up companies such as EnergyHub [10] and Greenbox [11] have already put the energy monitoring solutions with enhanced power information visibility into commercial market. These

newly conducted research and commercial solutions deploy wireless sensor network technology to perform detailed monitoring of selective individual appliances. To display the power information retrieved by the sensor network, setting up a dedicated display is a straightforward approach. For example, the WattBott residential electricity monitoring system [12] adopts the dedicated display to offer the visualization service. However, a dedicated display is not cost effective and restricts users to a certain geographical locations to access the power profile. Some academic research upon energy monitoring system design has proposed and discussed utilizing a web server approach to display the energy information [13], which solves the problem of geographical restriction. However it is not cost effective either. The server's computational and storage resources is either wasted if the working load is relatively small or its computational and storage resources is below the required needs if the working load is too large. An alternate solution is proposed by Sentilla Corporation [14]. Sentilla introduces a data center energy monitoring solution using wireless plug-load meters, which features cloud computing. Cloud computing bears the ability of load balancing so that if the computational requirement is high, a cloud can always distribute the work load across multiple computers or computer clusters by loading balancing. But there is still one critical problem associated with the cloud based solution, which is network security. Energy data is commonly sent to the cloud by either TCP or UDP over IP network, which is vulnerable to security attacks such as eavesdropping, data modification, identity spoofing and man-in-the middle attack.

In this thesis, we designed and integrated a cloud based building energy monitoring system using the bottom up approach. To deal with the network security issues, Secure

Shell Tunneling protocol [15] is adopted in this system to encapsulate the original TCP packet to perform the secure delivery of the energy data to the cloud.

## Chapter 3 System Design

In this work, we designed our own building energy monitoring system which consists of mainly three components: The first component performs the acquisition of building energy usage information. The second component achieves the function of processing raw energy data and displaying the processed energy data. The third component accomplishes delivering the energy data acquired from the first component to the second component. In this chapter, we will discuss the design of each component and proposes an overall architecture for the building energy monitoring system. In the next chapter, we will discuss the system implementation in details.

### *3.1 Energy usage information acquisition design*

The purpose of this component is to collect the building energy consumption data for later processing by the cloud computing component. This subsystem consists of both embedded system software and embedded system hardware.

As mentioned before, the traditional top down approach to acquire building energy usage information involves monitoring the aggregate load/root of the power distribution network, and then an algorithm is applied to disaggregate different types of electronic appliances according to the diverse power factors. This top-down approach lacks visibility of the detailed energy usage information and even fails when the power factors of the electronic appliances are similar. To acquire more visibility of the power usage information, applying dedicated energy sensors to perform power-socket level energy monitoring is a good choice. Two general methods are adopted with respect to harvesting the power-socket level energy sensor readings. One is the wired method which involves setting up dedicated wires/buses. This method requires a lot of mechanical work to set up

the wired connection and hence needs extensive amount time for system set up. Also this approach lacks the ability of system scalability, for example, when thousands of sensors are required to be installed, setting up the wires alone can be a very difficult job. Another method is by utilizing the wireless sensor network technique. Energy sensor data is harvested by deploying the wireless technology and hence do not need additional mechanical work to set up the system which greatly reduce the system set up time. Also, by utilizing a wireless sensor network for energy data harvesting, the system is more scalable.

In our system, we utilized the wireless sensor network approach to conduct the energy data acquisition. Each sensor in the Wireless sensor network must achieve two main functions: power sensing and wireless communication with other sensors. Also, in order to accomplish the multitasking requirement between the power sensing and wireless routing, and meet even more tasking handling requirements in the future, adopt an operating system for the sensor nodes to perform task scheduling is a good choice. However, confined by the low cost and low power limitation of existing sensors, the operating system for the sensors must be light weight without unnecessary features to meet the computational and storage limitation. Further, our energy sensor involves many wireless I/O transmissions, of which if adopting a synchronous/blocking mechanism OS, the performance will be greatly deteriorated. So we should adopt a non-blocking/asynchronous OS which allows other processing to proceed before I/O operations committed. This non-blocking mechanism prevents the waste of CPU time waiting for I/O operations to be completed.

### ***3.2 Energy data processing and visualization design***

In order to display the energy data, several methods can be deployed. The first method is to set up dedicated display inside buildings. However, the dedicated display method bears the following drawbacks. First of all, the acquired energy data are always raw energy data which needs further processing. However, the dedicated display barely provides energy data processing. Secondly, it is not cost effective. In order to enable building tenants to view the energy usage information conveniently, at least each dedicated display should be placed in each apartment in the building. So, it needs many dedicated displays which add more costs buying and setting up equipment. Thirdly, and most importantly, the dedicated display method can't offer remote visibility. The energy data can only be viewed by utilizing the display. People can't access the data remotely by using web browsing in personal computers or smart phones.

To overcome the drawbacks of the dedicated display approach, a method involves setting up a centralized computer server to process the energy data as well as to offer Word Wide Web accessibility is always adopted by industry. This method offers stronger processing ability and provides remote accessibility. However, this centralized method also bears immense drawbacks. First of all, it is not scalable. With the increasing needs for building energy monitoring, more and more energy data is required to be processed. Also more and more users' accesses to the web interface are expected. These increasing needs now exceed or will exceed the limited computational and storage resources provided by the centralized computer server. Second, it is not reliable or robust with respect to security. Because of its centralized location, it is more vulnerable to physical attacks from terrorists and criminals. Further, it is vulnerable to cyber security attacks.

Once the centralized server is under dedicated security attacks, the visualization service will be compromised and may even completely lose the ability of visualization. Lastly, it is not cost-effective either. Technical personnel are needed to keep the maintenance of the computer server, which increases the overall cost of the monitoring system.

To deal with the problems mentioned above, we choose the cloud computing method to meet the needs of data computing and web hosting. A cloud computing method has the following major advantages: Cloud offers scalability and elasticity via dynamic/on-demand provision of resources on a fine-grained, self-service basis near real-time, without users having to engineer for peak loads [16, 17]. Also, security of the cloud is often as good as or better than other traditional systems, in part because providers are able to devote resources to solving security issues that many customers can not afford [18]. Further, the maintenance of cloud computing applications is easier, because they do not need to be installed on each user's computer and can be accessed from different places.

So, in our system, we adopt a cloud computing subsystem to be the component for energy data processing as well as web hosting.

### ***3.3 Connectivity design between a wireless sensor network and a cloud***

After determining the wireless sensor network method for energy data acquisition and adopting the cloud computing approach for energy data processing and web visualization, the problem now lies in how to connect the two subsystems together.

One popular approach that connects the two subsystems together is by deploying a general-purpose IP router which routes the IP packets in the wireless sensor networks to the IP network outside. However, to enable an IP router to be the connecting device for

the two subsystems, each sensor in the wireless sensor network much has a complete network stack to support IPv6 protocol. Till now, only a small amount of wireless sensor network support this feature.

One more issue associated with the general-purpose IP router is the data transmission security over the Internet. The data routed by the general-purpose IP router is usually a TCP or UDP packet, which is vulnerable to security attacks such as eavesdropping, data modification, identity spoofing, and man-in-the middle attacks. To enhance the security feature, rather than using the standard IP router, we designed an application specific embedded Linux gateway to deliver the energy data to the cloud subsystem.

The embedded Linux gateway retrieves the energy data from the root sensor of the wireless sensor network. Then it utilizes tunneling protocol to encapsulate the original TCP or UDP energy packets as payload. By using tunneling we can carry the energy payload over an incompatible delivery-network, or provide a secure path through an untrustworthy network. To be more specific, we utilize secure shell (SSH) tunneling which consists of an encrypted tunnel created through a SSH protocol connection. SSH tunnels are set up between the application-level embedded Linux gateway and the cloud computing server to transfer unencrypted traffic over Internet through an encrypted channel.

### ***3.4 Overall System Architecture***

As discussed above, the overall building energy monitoring system consists of three major parts which is illustrated in Figure 2.1 below. The first one is energy data acquisition subsystem, which comprises a wireless sensor network (WSN) with the

function of sensing and acquiring the energy consumption information. The sensing data is routed within the wireless sensor network and finally to the gateway sensor which connects to the embedded Linux gateway.

The second subsystem is the application level embedded Linux router, which routes the sensing data to the cloud. The embedded Linux router first reads the data from the gateway sensor and then routes them to the cloud computing unit over the Internet. The TCP socket programming technique is employed to ensure the reliability of transmission over the Internet. Secure Shell Tunneling protocol [15] is utilized to encapsulate the original TCP energy packet to securely deliver the data to the cloud computing server over the Internet.

The third one is the cloud computing component, which includes cloud processing the raw data, storing the processed data into a database and displaying the near real-time and long term power usage data via a cloud web server. People who are interested with these power data such as building tenants or building energy managers can easily acquire the energy consumption data on both long term and real-time level without setting up an extra database server or installing dedicated real-time display

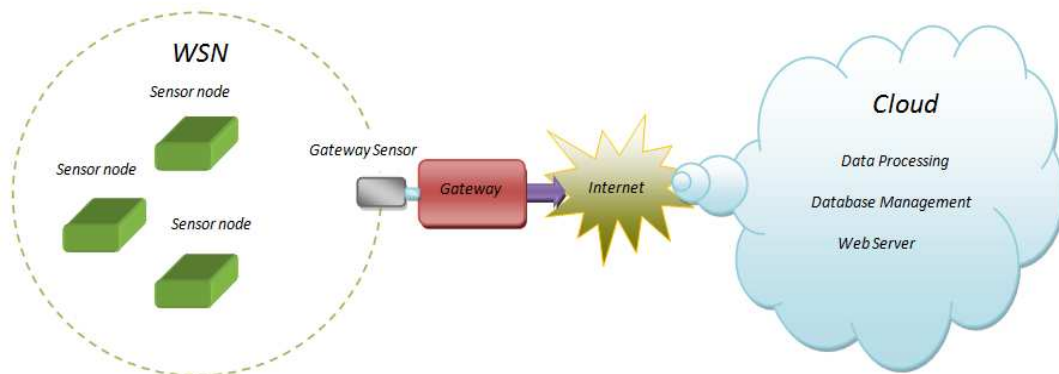


Figure 3.1 System Architecture for building energy monitoring.

## Chapter 4 System Implementation

We have discussed the system design in the previous chapter. This chapter illustrates in details how the system is implemented. We will discuss the detailed implementation of each subsystem in this chapter and will show the overall system test in the following chapter.

### *4.1 Implementation of data acquisition subsystem*

As mentioned in the previous chapter, in order to measure the power on power-socket level without adding redundant electrical wires or compromising the existing infrastructure, a wireless sensor network (WSN) approach is adopted. Each sensor in the WSN must achieve two main functions: power sensing and wireless communication with other sensors. An ACme board which is released under BSD License is a dedicated power sensing board with an IEEE 802.15.4 wireless interface aimed at forming a WSN. We adopt ACme boards as the sensing hardware for the wireless sensor network.

TinyOS is an open source Operating System dedicated to support embedded devices in wireless sensor networks. It is completely non-blocking/asynchronous which allows other processing to proceed before I/O operation committed [19]. Also, TinyOS is a light weight OS which fulfills the onboard memory and computing resource limitation of the ACme sensor as well as supports the corresponding hardware drivers of ACme sensors. So, the TinyOS operating system is adopted to fulfill the multitasking and concurrent requirement of ACme's power sensing and wireless communication.

The ACme application is written in nesC, a dialect of C language dedicated to support wireless sensor networks. nesC features command and event split-phase which

mimics hardware parallelism and supports the TinyOS asynchronous mechanism to avoid CPU idle time waiting for the sub process to be completed [20].

nesC have well defined components which should be considered as hardware abstraction. Component usually provides interfaces for its upper level component and uses interfaces provided by its lower level component expect that top layer component only uses interfaces provided by lower components and lowest layer component only provides interfaces to its upper level component. In order to include a lower level component to an upper level component, it is a necessity to connect the interfaces provided by the lower level component to interfaces used by the upper level component. Then the upper level component could call a command to the lower level component through the connected interface and the lower level component could fire an event to the upper level component through the connected interface.

Figure 4.1 below illustrates the ACme application component. The top layer component calls a sensing command through its interface connected to the sensing component, during which the sampling interval is designated. The sensing component then launches via the interface connected to the Timer Component, a timer command which specifies the designated sampling interval. Each time the Timer reaches the sampling interval, the Timer Component triggers a timer event via the connected interface to its upper level sensing component which then triggers a sensing command to its lower level sensor component. Each time the sensor component's sensing is done, it triggers an event to its up level sensing component which then triggers a sensing done event to the top layer component. When the top layer component receives a sensing done event, it reads the sensing results and calls a command to its lower level Active Message

Component to transmit the data across the wireless sensor network. The Active Message component then calls the command to its corresponding lower level component and so on, which finally put the data into the hardware transmission FIFO. When the actual data transmission is committed, corresponding events are generated to their father components via the connected interfaces and finally a routing done event is generated from the Active Message Component to the top layer component.

The Collection Tree Routing Component in the ACme application uses the collection tree routing algorithm which configures the wireless sensor network into a tree topology. Each ACme node in the collection tree is a tree node and the gateway sensor is the root. We can configure multiple gateway sensors to be the tree roots which form distinct trees. Each tree node has a unique path to its tree root and the data is routed one step closer to the root each time by the ACme sensor node along the path. Each time the lowest level Radio Hardware Component receives byte data from other ACme sensor, events are generated from the lowest layer to the top layer, then the top layer uses collection tree component to route the Active Message to the next node along the path to the root.

Crossbow's Telosb mote is utilized to be the gateway sensor in the wireless sensor network. It is IEEE802.15.4 compliant, equipped with a USB interface and is supported by TinyOS operating system [21]. In the TinyOS utility, a base station application is available to configure the Telosb mote to be a gateway sensor. The base station application bridge the Active Message received by the IEEE 802.15.4 interface to the USB interface so that it is convenient to retrieve the wireless sensor network sensing data by connecting the gateway sensor to our application level router via the USB interface.

The Active Message has the following format, of which the payload area could be defined according our own application needs.

- Active Message Indication (1 byte)
- Destination Address (2 byte)
- Link Source Address (2 bytes)
- Message Length (1 byte)
- Group ID (1 byte)
- Active Message Handler type (1 byte)
- Payload (6 bytes for our application )

An example Active Message for our application is

“00 FF FF 00 02 06 00 08 00 00 00 00 0E 22”. Its destination address is “FF FF” specifying the root address of our collection tree. Its Link source address is “00 02” which is the node ID specified at compile time in our application. Its payload length is “06” bytes and group ID is “00” which indicates the collection tree it belongs to. Its active message handler type is “08” and bears the six bytes power data payload “00 00 00 00 0E 22”.

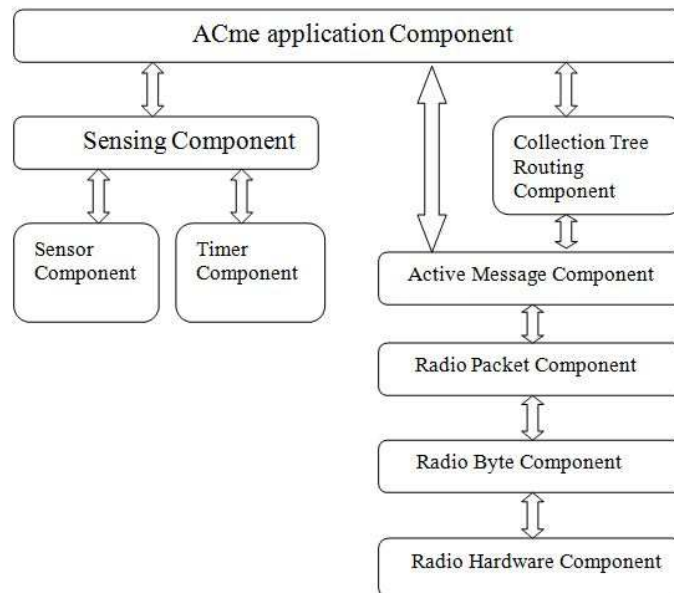


Figure 4.1 ACme Application Component Illustrations.

## ***4.2 Implementation of application specific gateway***

We use BeagleBoard-xM as our gateway hardware platform, and adopt Ångström, a Linux distribution which can be applied to a variety of embedded devices, as our operation system for the router. A TCP client socket program is developed by using Java programming language to ensure the delivery of the Active Message retrieved from the connected gateway sensor to the cloud. Also, a Java TCP server socket is developed which runs on the cloud to establish a communicational connection with the Java TCP client socket over Internet. We will talk about the server socket later in the next section.

### ***4.2.1 Hardware platform***

The Beagle Board is a low-cost, fan-less single-board computer based on TI's OMAP3 device family, with all of the expandability of today's desktop machines, but without the bulk, expense, or noise [22]. It is equipped with TI's OMAP3530 processing platform which includes a 1GHz ARM Cortex-A8 CPU [23]. Its onboard memory/RAM size is 512Mb but is not equipped with on board NAND which in turn requires the boot loader, the Linux kernel and File System to be stored into a microSD card. Its primary I/O interface includes a microSD/MMC card slot, an Ethernet port, a RS-232 port, and 4 USB ports. Please refer to Figure 4.2 below for the hardware overview.

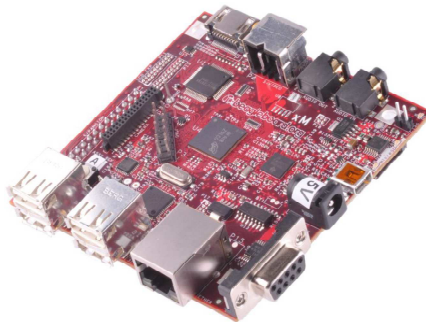


Figure 4.2 Hardware overview of beagleboard –Xm.

### ***4.2.2 Operating System-Ångström***

Linux is always considered and chosen to be the operating system running on embedded devices because it is open source, free and customizable. By using Linux, we can customize the desirable features we want and reduce the unnecessary redundancy to meet the limited computing and storage limitation of the embedded devices.

Ångström is a popular Linux distribution which is verified to run smoothly and fast on beagleboard and was initiated by open source embedded developers who worked on the OpenSimpad, OpenZaurus and OpenEmbedded projects [24]. We adopt Ångström Linux distribution as the operation system for BeagleBoard hardware.

As mentioned above, BeagleBoard xM does not include on-board NAND, which requires a separate SD card to boot operating system from. In order to make a bootable SD card for BeagleBoard xM, a special geometry in the partition table should be set up to comply with the OMAP3 (BeagleBoard xm processing platform) ROM. We need to split the SD into two partitions, with one “Fat32” partition holding the boot files and one “Ext3” partition holding the Linux root file system. Formatting utility in Linux is utilized to partition the SD card into the desired formats needed.

We customize a fairly basic Ångström image for BeagleBoard without additional application packages. After building the Ångström, four files required to boot the operating system are generated: MLO, u-boot.bin, uImage, and the root file system: Angstrom-BeagleBoard-xxx. MLO is the first stage boot loader which fits entirely into the on-chip memory of the OMAP3530. Upon booting, the on board ROM code loads MLO which configures the external memory for use and then loads the second stage boot loader, u-boot.bin. Constrained by the size of the internal memory of the OMAP3530, the

two stage boot loaders are separate; otherwise they can be combined into one boot loader. The second stage boot loader, u-boot.bin then loads the kernel image, uImage. Then the Linux kernel takes over control and the operating system can boot and run normally.

Because the on board ROM code loads the MLO by reading the first physical address of the SD card, the first three files should be placed to the Fat32 partition of the SD card in a strict order and the fourth root file system should be placed into the Ext3 partition of the SD card.

### ***4.2.3 System Configurations and Application Program***

Now, we have a basic Ångström distribution bootable from the SD card on the BeagleBoard, the following will discuss all the system configurations and application programs needed to make BeagleBoard an application level router.

#### ***4.2.3.1 Java Runtime Environment***

Our application programs such as the client and server socket programs are needed to run on hosts with different hardware/software platforms such as Ångström /ARM and Ubuntu/amd64, so it is desirable to utilize cross-platform programming language for the purpose of uniformity and simplicity. Java is among the most developed and stable cross-platform programming language and hence it is deployed to be the primary programming language in our system development.

JamVm is an extremely compact Java Virtual Machine which conforms to JVM specification version 2 and supports a variety of OS/Architectures including Linux/BeagleBoard (ARM Cortex A8) [25]. We use JamVM bundled with GNU classpath to be our Java Runtime Environment (JRE). Since Ångström has Java in its repository, so a single terminal command “opkg install jamvm” will automatically

download and install the JamVM as well as GNU classpath APIs. Then Java bytecode can be run on our BeagleBoard using the JRE.

#### **4.2.3.2 Serial Port Listener**

The first application program needed to run on top of Ångström is a serial port listener. It reads the energy data from forwarded by the gateway sensor which is connected to the BeagleBoard using the USB Virtual COM Port. (Recall that, the firmware in the gateway sensor collects all the energy data from the wireless sensor network and forward the data to the USB Virtual COM/Serial port).

In order for the Java Listener to listen to the serial port, it is a necessity to load the corresponding LKM (Loadable Kernel Module) to invoke the USB to Serial driver because the angstrom does not usually load this driver automatically after booting. In Ångström, upon booting, the system first uses the file `/etc/inittab` to identify the default run level (run level is the point the system enters) and find, under the corresponding run level directory e.g. `/etc/rc5.d/`, the symbolic links to all the start up scripts which lie in the central repository `/etc/init.d /`. Then the system will execute the actual start up scripts according to the order specified in the symbolic link name. In order to load the drivers automatically in the booting process, we write a start up script which installs the corresponding drivers/kernel modules and place it in the central repository. Then, under the appropriate run level directory, we create a symbolic link pointing to the script.

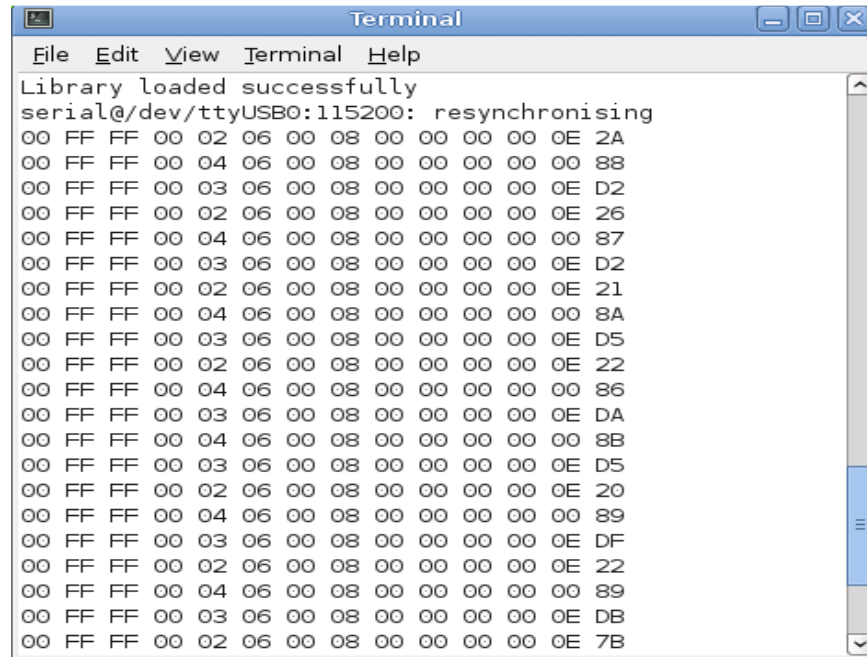
After loading the USB to serial driver, the gateway sensor's USB Virtual COM port which connects to the BeagleBoard appears as an extra serial port to the BeagleBoard and should be mounted as `/dev/ttyUSBx` in the root file system (x is a Arabic numeral and is usually 0).

TinyOS developing utility offers a Java Listener of which the function is to simply stream the input from serial port to the Linux terminal. This Java Listener is platform dependant because it uses the Java Native Interface. The **Java Native Interface (JNI)** is a programming framework that enables Java code running in a JVM to call and to be called by native applications (programs specific to hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly [26]. So, the Java Listener specific to the Ångström and BeagleBoard platform is adopted. The Java CLASSPATH environmental variable should be configured to include the absolutely path to the Java Listener Jar file so that Java can always find, within the Jar file, the Java Listener main class and all its dependant classes. For example, in order to set up the CLASSPATH automatically each time after system boots, the command

"export CLASSPATH=/`<path>`/listener\_name.jar " should be placed within the file "/etc/profile" in the file system.

After all the work is done as mentioned above, it is ready to acquire the energy data from the wireless sensor network and printing them on them on the terminal. Java listener is invoked by the following command:

"java net.tinyos.tools.Listen -comm serial@/dev/ttyUSBx:115200", which takes the mount point of the Gateway sensor's USB Virtual COM port as its first argument and the baud rate "115200" as its second argument. A sample output is showed in Figure4.3 of which each line in the terminal is a complete Active Message.



```
Terminal
File Edit View Terminal Help
Library loaded successfully
serial@/dev/ttyUSB0:115200: resynchronising
00 FF FF 00 02 06 00 08 00 00 00 00 0E 2A
00 FF FF 00 04 06 00 08 00 00 00 00 00 88
00 FF FF 00 03 06 00 08 00 00 00 00 0E D2
00 FF FF 00 02 06 00 08 00 00 00 00 0E 26
00 FF FF 00 04 06 00 08 00 00 00 00 00 87
00 FF FF 00 03 06 00 08 00 00 00 00 0E D2
00 FF FF 00 02 06 00 08 00 00 00 00 0E 21
00 FF FF 00 04 06 00 08 00 00 00 00 00 8A
00 FF FF 00 03 06 00 08 00 00 00 00 0E D5
00 FF FF 00 02 06 00 08 00 00 00 00 0E 22
00 FF FF 00 04 06 00 08 00 00 00 00 00 86
00 FF FF 00 03 06 00 08 00 00 00 00 0E DA
00 FF FF 00 04 06 00 08 00 00 00 00 00 8B
00 FF FF 00 03 06 00 08 00 00 00 00 0E D5
00 FF FF 00 02 06 00 08 00 00 00 00 0E 20
00 FF FF 00 04 06 00 08 00 00 00 00 00 89
00 FF FF 00 03 06 00 08 00 00 00 00 0E DF
00 FF FF 00 02 06 00 08 00 00 00 00 0E 22
00 FF FF 00 04 06 00 08 00 00 00 00 00 89
00 FF FF 00 03 06 00 08 00 00 00 00 0E DB
00 FF FF 00 02 06 00 08 00 00 00 00 0E 7B
```

Figure 4.3 sample output of the Java Listener.

#### 4.2.3.3 TCP Client Socket

Socket Programming is also called computer network programming, which enables processes to communicate with other processes across a computer network. The process starting the communication is called a client and the process waiting for the client to establish a communicational connection is called a server. There are two types of communication between a client and a server. One is connection oriented, which usually uses the TCP (Transmission Control Protocol) as the Transport layer protocol and the other is connectionless which usually uses the UDP as the Transport layer protocol. TCP handles the issues such as resending the packets in case of packets lost and hence guarantees the delivery of the communicational data, so we prefer TCP Socket Programming to ensure the reliability. Java programming language is utilized to implement the sockets.

The Client socket program takes the output of the Java Listener as input, which can be done using pipe "|" command in Linux. Each active message (one line in the terminal) is delivered from the Client socket process running on the router to the Server socket process running on the cloud. In order to achieve the function above, a client socket object should be created by taking the IP address of the cloud host and the port number of the Server socket process as parameters. For example, the following Java statement "Socket clientSocket = new Socket ("107.20.169.101", 50000)" creates a client socket object by designating "107.20.169.101" as its destination IP address and port "50000" as its destination port number. Within the client socket process, each active message piped from Java Listener is passed to the client socket object which then transmits the data to the server socket process hosted on the cloud.

#### ***4.2.3.4 SSH tunneling***

SSH tunneling is created using SSH protocol, which comprises a secure tunnel between a client and a server. To guarantee network security, the SSH tunneling technique is employed to deliver the data from the gateway's Client socket process to the cloud's Server Socket process over the Internet.

By deploying SSH tunneling, we can configure to forward a specific local port to a designated port on the cloud so that any data sent by the client socket to the specific local port will be forwarded to the designated port on the cloud. For example, the command below

"ssh -L 50000:localhost:123456 -i securitykey.pem ubuntu@107.20.169.101" forwards any data sent to local port "50000" to the port "123456" of the cloud host with the IP

address 107.20.169.101. The identity file "securitykey.pem" is the security key used to identify the local host's legitimacy while accessing the cloud host.

### ***4.3 Implementation of cloud computing subsystem***

The Cloud Computing component accomplishes three major tasks. Firstly, a server program is implemented in Java on the cloud to receive active message from the Client program hosted on the router. Upon receiving, the active message is parsed, processed and then written to the cloud relational database. Secondly, an Apache HTTP Server is set up and web pages are designed to display the long term and real-time active power data.

#### ***4.3.1 Cloud Computing Platform***

Amazon Elastic Compute Cloud (EC2) is a cloud computing platform provided by Amazon Web Service (AWS), which allows users to run their own computer applications. Users can launch an Amazon Machine Image (AMI) to create a cloud computing unit with the desired features. Amazon EC2 platform is utilized to implement our cloud computing component and Amazon Machine Image of which the OS is Ubuntu-maverick-10.10 is adopted to be our cloud computing unit.

AWS provides a friendly graphical user interface (GUI) for users to easily manage their AMI image such as creating, erasing an image instance, booting & rebooting and stopping a image instance. Figure 4.4 below is an example of the AWS GUI console for EC2 to manage our cloud image. We can obtain and attach a public IP and domain name to our instance and users can access to our web interface using the public IP or domain name. Our public IP is "107.20.169.101".

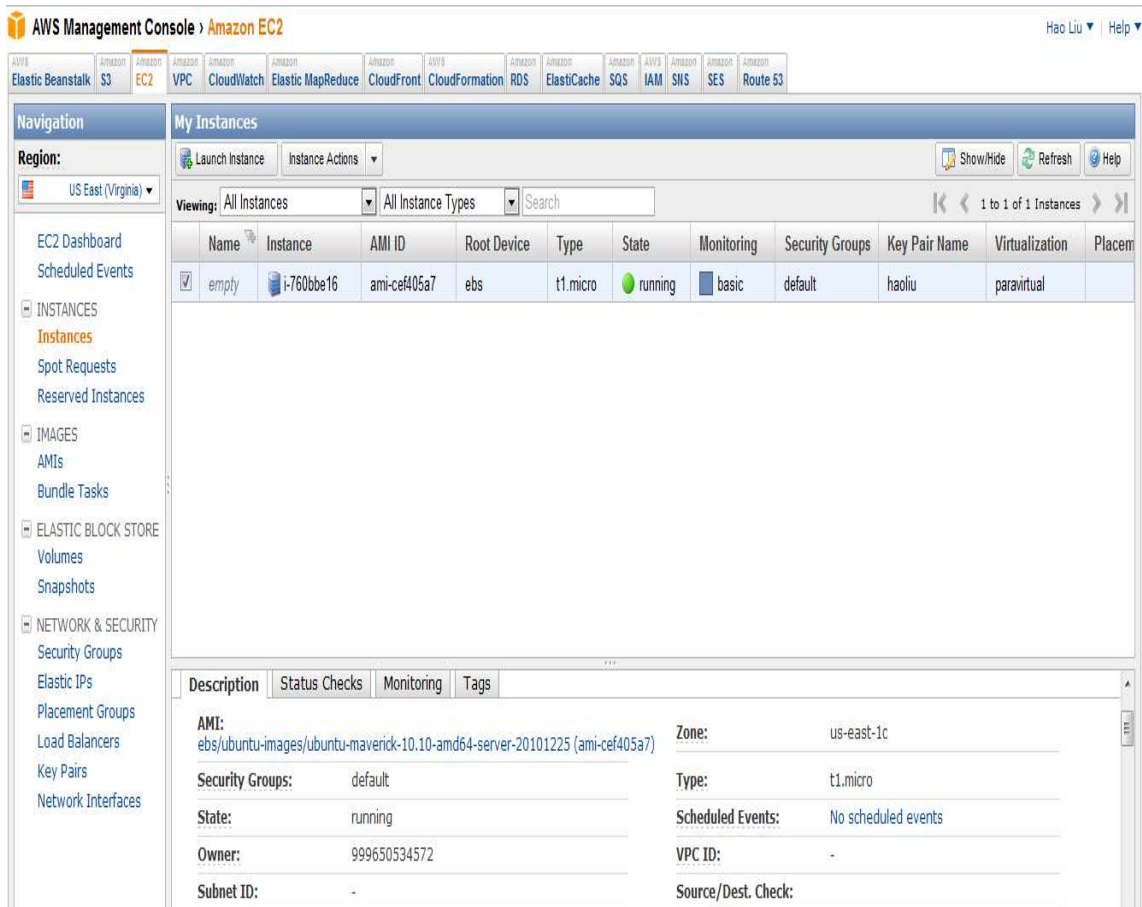


Figure 4.4 EC2 Management Console.

### 4.3.2 System Set up and Application program

#### 4.3.2.1 LAMP

We use open source LAMP which is the acronym for the free, open source software stack constituted by Linux, Apache, Mysql and PHP to meet the needs of web hosting and data processing.

LAMP is set up and configured to run automatically upon booting as the daemon software on our cloud image to support HTTP requests, Database management and server side manipulation.

#### ***4.3.2.2 Server Socket Programming***

As mentioned in Chapter 3, the Client Socket in the router delivers the active message to the Server Socket process. The Server Socket process keeps on listening to the port designated by the client socket to establish connections and receive data. Upon receiving, the Server Socket program extracts from the active message the ID of the sensor node, the raw power data and processes the raw power data. Then along with the ID and the processed data, it adds additional timestamp to create a new data set ready to insert into database.

The Active Message has the format of “00 FF FF 00 01 06 00 08 00 00 00 00 00 05” which has 14 bytes in total. The server socket program checks the length of the Active Message it receives to verify the integrity of the data. After the verification successfully done, server socket first extracts the sensor’s ID and convert it from hexadecimal to decimal. Then, it extracts the last six bytes (the payload), converting them from hexadecimal to decimal and using the offset and gain to calculate the real power data.

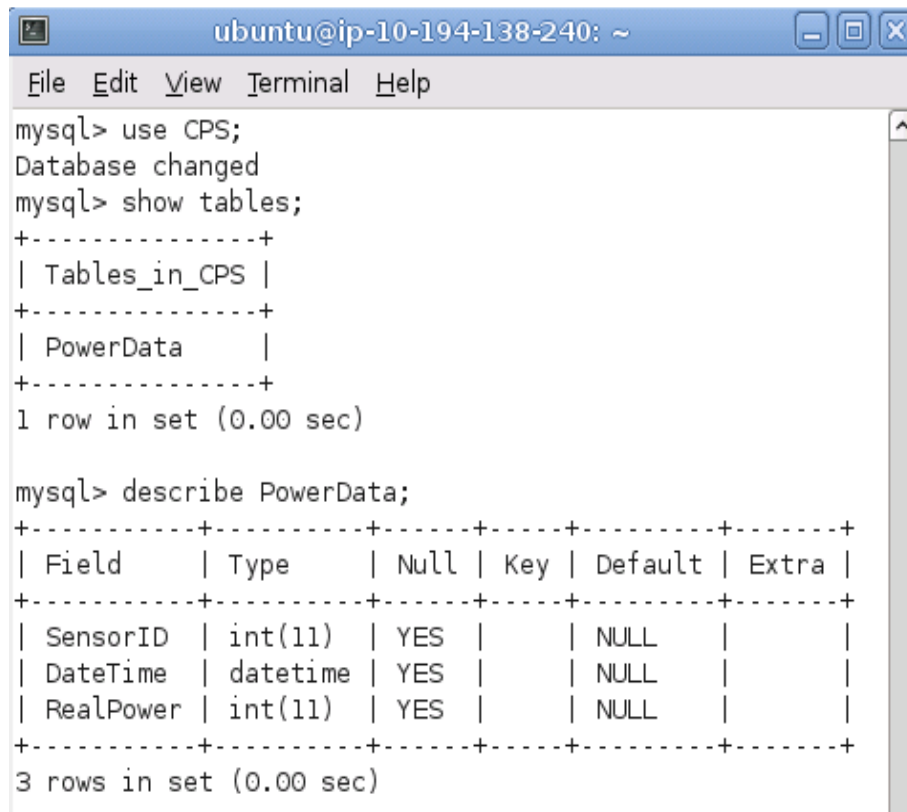
Then a timestamp is created and combined with the ID as well as the calculated power to form a new record ready to be inserted into the row of the database table.

JDBC, unofficially referred to as Java Database Connectivity offers an API (Application Programming Interface) for Java programming language to access and manage the relational database. Our Server Socket program utilizes JDBC to manage our Mysql relational database management system (RDMS).

First of all, a database called “CPS” is created within Mysql RDMS, and within the “CPS” database, a table named “PowerData” is created with the following fields:

- **SensorID:** This is the field holding the ID of the sensor which acquires the power consumption information. The data type for this field is “Int”.
- **DateTime:** This is the field holding the arriving date and time of the active message to our cloud. The data type for this field is “datetime” which has the “YYYY-MM-DD hh:mm:ss” format. Based on our test, the Active message arrives to our cloud from the router within less than one second which complies with the real time power monitoring requirement.
- **RealPower:** This is the field holding the real power data and the data type is “Int”.

Figure 4.5 below is the illustration of the database and table created for our system.



```

mysql> use CPS;
Database changed
mysql> show tables;
+-----+
| Tables_in_CPS |
+-----+
| PowerData      |
+-----+
1 row in set (0.00 sec)

mysql> describe PowerData;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| SensorID   | int(11)   | YES  |     | NULL    |       |
| DateTime   | datetime  | YES  |     | NULL    |       |
| RealPower  | int(11)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

Figure 4.5 Illustration of database table construction.

Secondly, in order to use JDBC to connect to our Mysql RDBMS, after loading the JDBC driver into our server socket program, we should register the JDBC driver for MySQL by the following java code:

```
Class.forName("com.mysql.jdbc.Driver");
```

Then the URL of database server for mysql database named "CPS" on the "localhost" with the default port number "3306" should be defined by the following java code:

```
String url = "jdbc:mysql://localhost:3306/CPS";
```

Then a connection object to the database for a user with the password can be obtained by the following:

```
Connection con=DriverManager.getConnection (url, "username", "password");
```

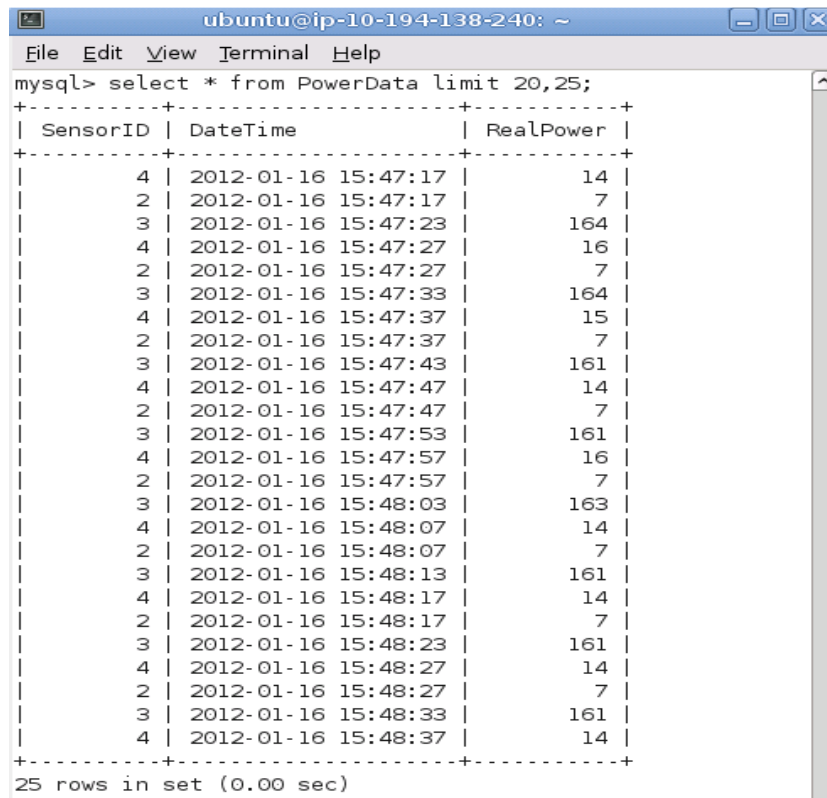
Till now, by calling the "createStatement" method of the connection object mentioned above, a statement object is created which includes the "executeUpdate" method taking standard SQL language as parameter to update the table in our database.

For example:

```
stmt=con.createStatement();
```

```
stmt.executeUpdate("INSERT INTO PowerData ( SensorID, DateTime,  
RealPower ) VALUES (" + Integer.toString(ID) + "," + timestamp+ "," +  
Integer.toString(Power) + ")");
```

Figure 4.6 below illustrates the table records inserted by our Server Socket Program using JDBC: Sensor with ID 2 monitors the power usage of our application level router power. Sensor with ID 3 monitors the power usage of a Gateway E Series Desktop. Sensor with ID 4 monitors the power usage of an IBM T420 Laptop.



The screenshot shows a terminal window titled 'ubuntu@ip-10-194-138-240: ~'. The terminal displays a MySQL query: 'mysql> select \* from PowerData limit 20,25;'. The result is a table with three columns: SensorID, DateTime, and RealPower. The data is as follows:

SensorID	DateTime	RealPower
4	2012-01-16 15:47:17	14
2	2012-01-16 15:47:17	7
3	2012-01-16 15:47:23	164
4	2012-01-16 15:47:27	16
2	2012-01-16 15:47:27	7
3	2012-01-16 15:47:33	164
4	2012-01-16 15:47:37	15
2	2012-01-16 15:47:37	7
3	2012-01-16 15:47:43	161
4	2012-01-16 15:47:47	14
2	2012-01-16 15:47:47	7
3	2012-01-16 15:47:53	161
4	2012-01-16 15:47:57	16
2	2012-01-16 15:47:57	7
3	2012-01-16 15:48:03	163
4	2012-01-16 15:48:07	14
2	2012-01-16 15:48:07	7
3	2012-01-16 15:48:13	161
4	2012-01-16 15:48:17	14
2	2012-01-16 15:48:17	7
3	2012-01-16 15:48:23	161
4	2012-01-16 15:48:27	14
2	2012-01-16 15:48:27	7
3	2012-01-16 15:48:33	161
4	2012-01-16 15:48:37	14

At the bottom of the terminal output, it says '25 rows in set (0.00 sec)'.

Figure 4.6 illustration of database table record.

#### 4.3.2.3 Web Page Design

HyperText Markup Language (HTML) is adopted to design the graphical user interface of our web page and PHP is embedded into the HTML code functioning as a server side script to perform database query.

Users who are interested with the power information could indicate some keywords such as Sensor ID and time in the form defined by the HTML in the web interface and by submitting the form, user's query information is sent via HTTP browser to our server side PHP page. The PHP code then queries the database accordingly and echo back the power information to the browser.

Two simple web interfaces are designed for our system. One is dedicated to display the short-term active power consumption information. Users who are interested with the

power information could indicate the sensor ID and then this web interface will display from the last 1 minute to current near real-time power consumption data. Please see figure 4.7 for the short-term power web interface (the sensor is set up for monitoring the microwave oven).

Another web interface is designed to display the daily power usage information which could be used by building manager or tenants to identify the power usage pattern in a day. The power information depicted in Figure 4.8 below shows the 24 hours power usage information of a freezer on 01/17/2012.

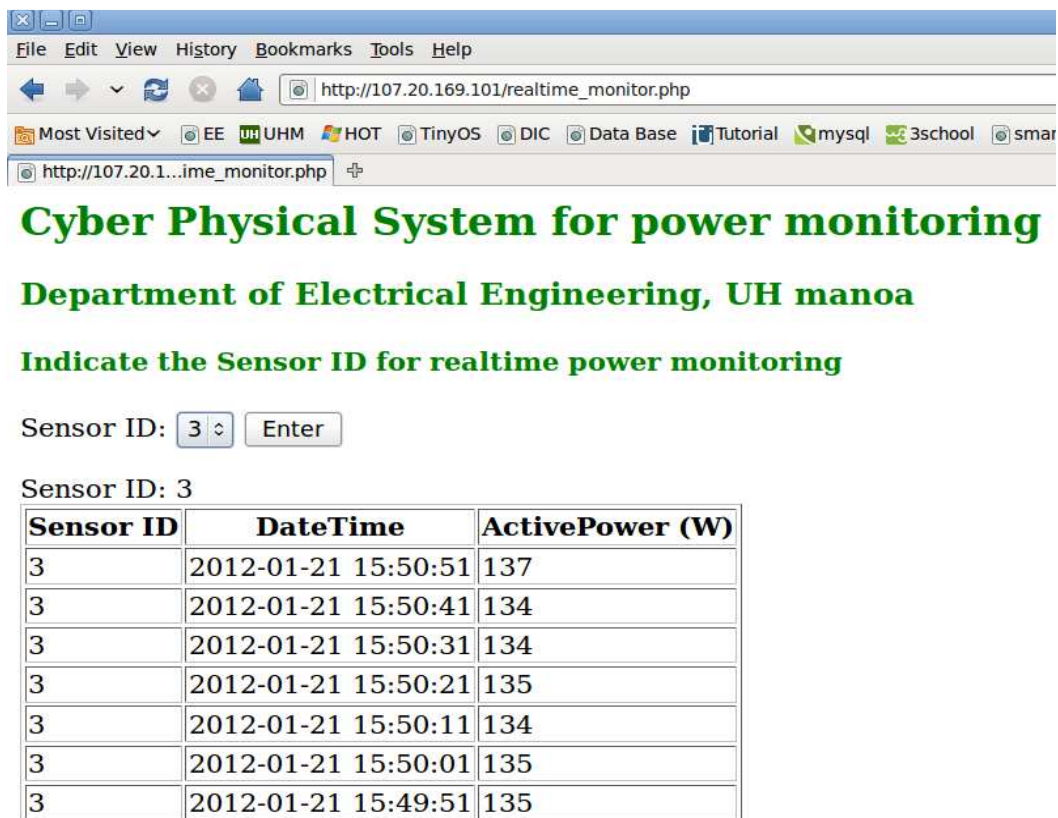


Figure 4.7 web interface for short-term power monitoring.

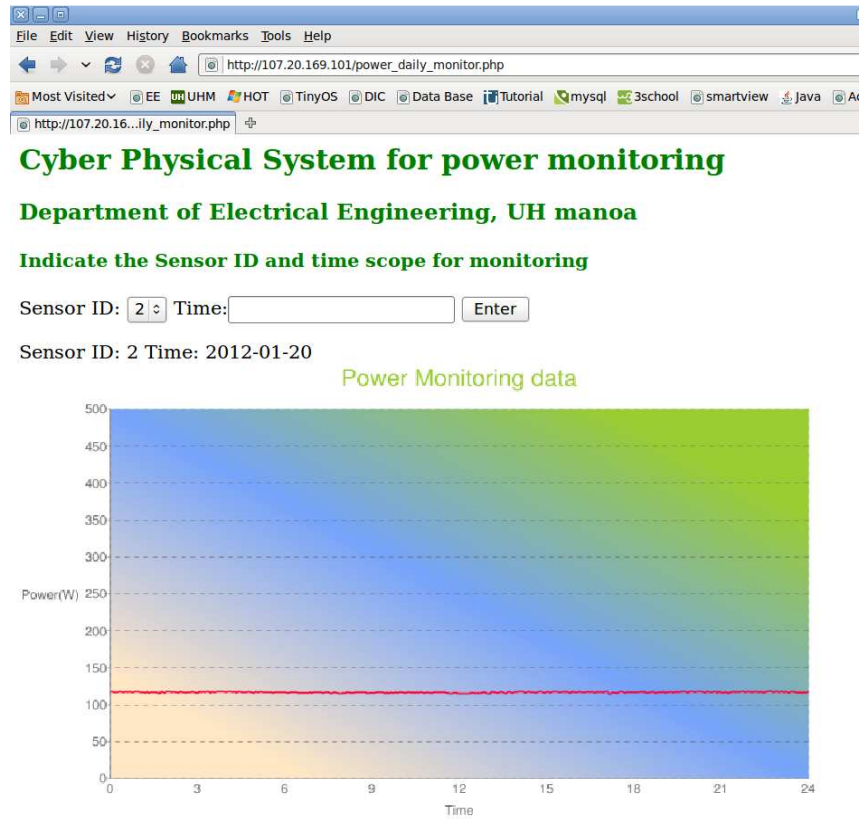


Figure 4.8 web interface for daily power monitoring.

## Chapter 5 System test

### 5.1 Short-term functionality test

To test the short-term functionality, we utilize PCs, microwave ovens and freezers as our test cases and adopt the short-term web interface to show results of the active power consumption data.

Figure 5.1 below shows the short-term power data of the GR-820 GOLDSTAR freezer.

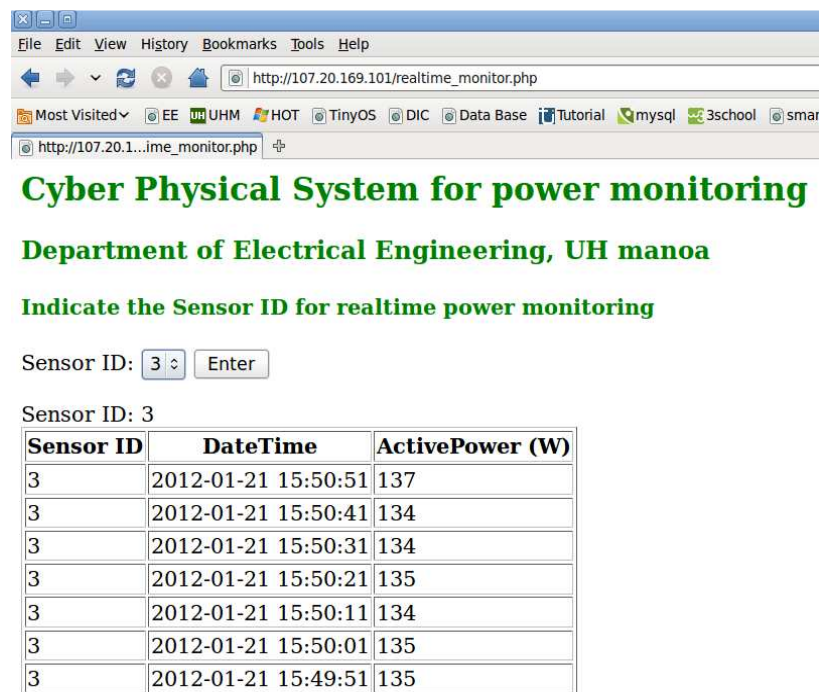


Figure 5.1 Short-term power data of the GR-820 GOLDSTAR freezer.

Figure 5.2 below shows the short-term power data of the Kenmore microwave oven with model no. 565.8738612.

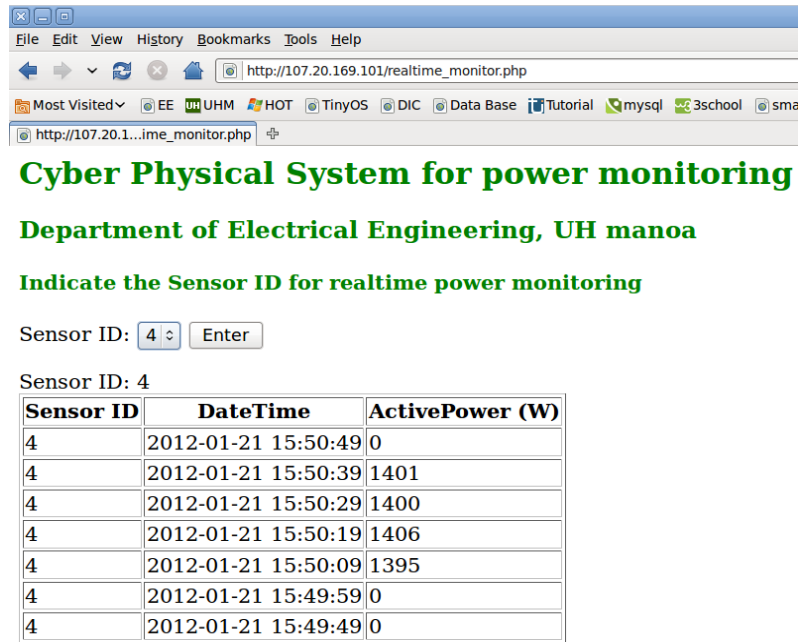


Figure 5.2 Short-term power data of the Kenmore microwave oven.

Figure 5.3 shows the short-term power data of the Gateway E series desktop computer with Pentium 4 processor and 16 inches LCD display.

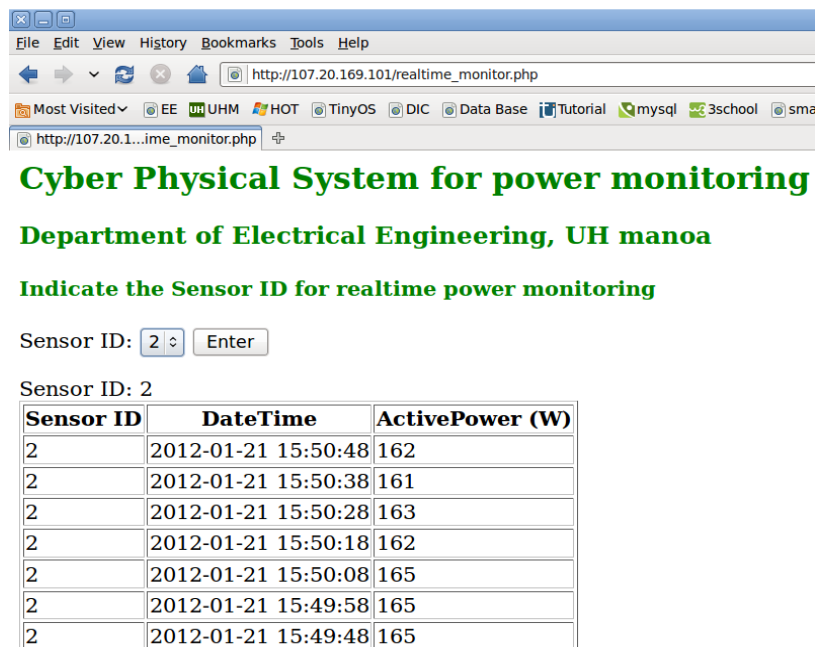


Figure 5.3 Short-term power data of Gateway E series desktop computer.

## 5.2 Robustness functionality test

To test the robustness of the monitoring system, we set up the system to work continuously for a couple of days and use the daily monitor web interface to display the results.

We choose several electrical appliances including desktop computers, laptop computers, freezers and our embedded Linux routers to be our test cases.

The first test case is a Gateway E series desktop computer with Pentium 4 processor and 16 inches LCD display. We use our CPS system to continuously monitor it for two days, and Figure 5.4 and Figure 5.5 shows the power usage information of the computer on 01/17/2012 and 01/18/2012. The figures below shows that the computer was used on regular working hours and the user may forgot to turn it off on 01/17/2012. The power of the computer fluctuates between 150 to 250 watts depending on the work load when running.

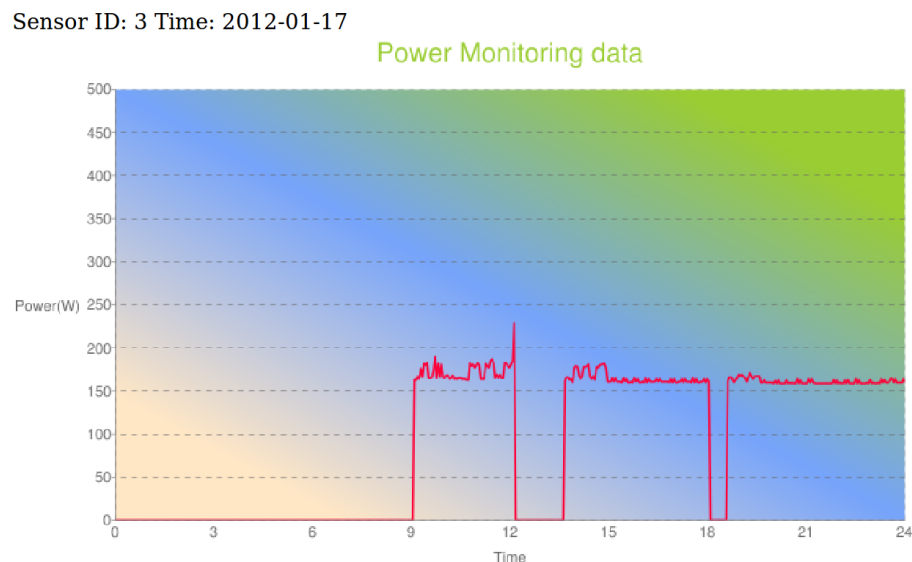


Figure 5.4 Daily Power data of Gateway E series desktop on 01/17/2012.

Sensor ID: 3 Time: 2012-01-18

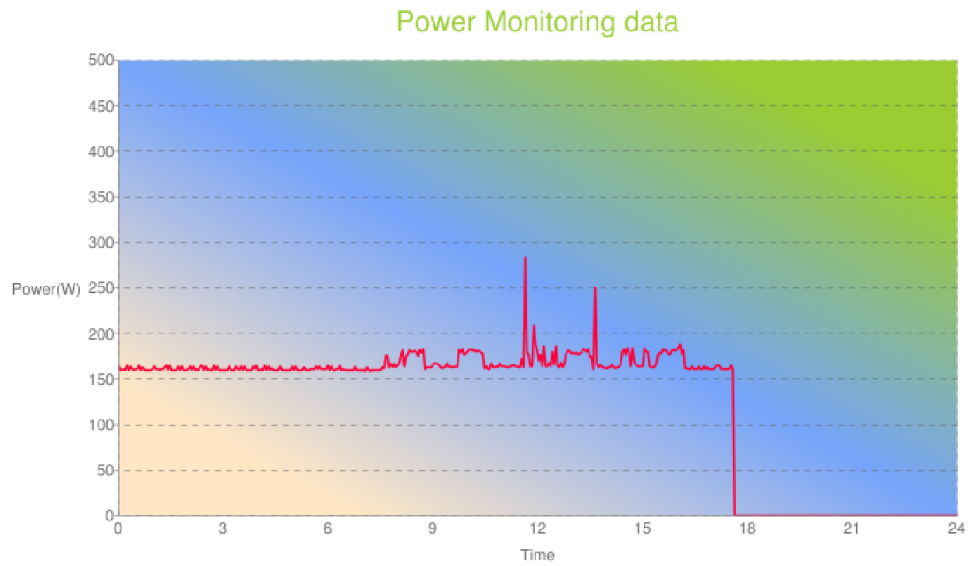


Figure 5.5 Daily Power data of Gateway E series desktop on 01/18/2012.

The second test case is the GR-820 GOLDSTAR refrigerator, figure 5.6 shows the power usage pattern of the freezer on 01/20/2012.

Sensor ID: 2 Time: 2012-01-20

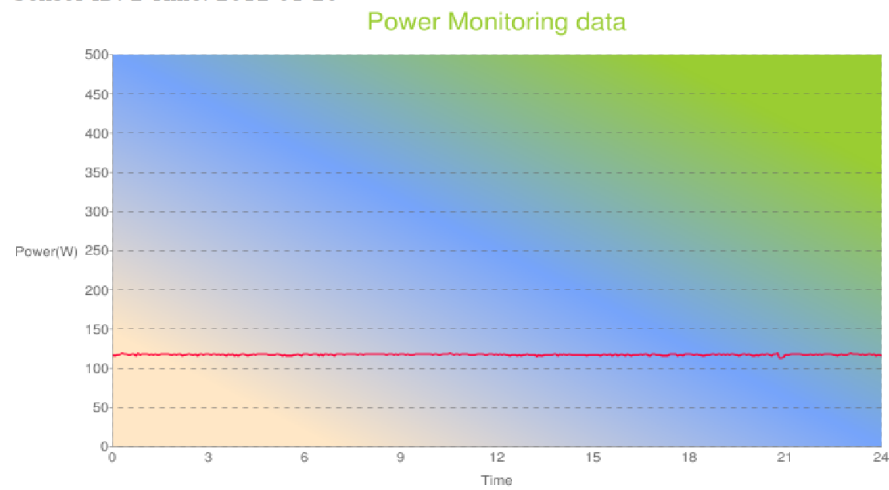


Figure 5.6 Daily power data of GR-820 GOLDSTAR refrigerator on 01/20/2012.

The third test case is the ThinkPad T420 laptop with Core i5 processor and 14 inches LCD display, figure 5.7 below shows the power usage information on 01/17/2012.

We can see that the laptop is used between around 11:00 am to 8 pm and the power is around 18 watts.

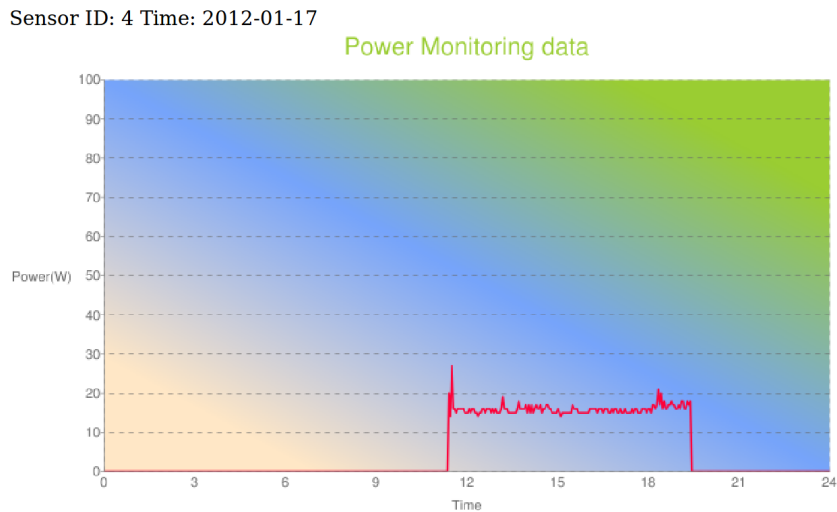


Figure 5.7 Daily power data of ThinkPad T420 laptop on 01/17/2012.

The last test case is our application level Linux gateway which consumes around 8 watts when working.

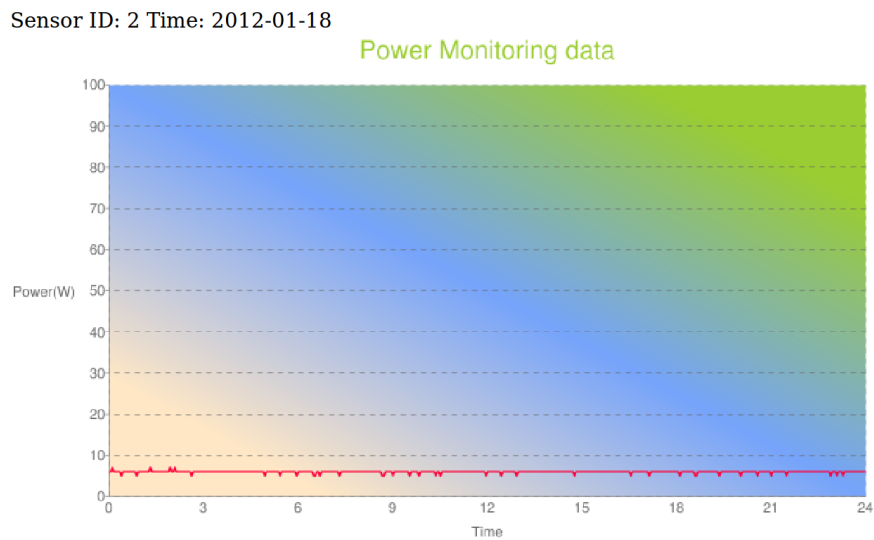


Figure 5.8 Daily power data of application level gateway on 01/18/2012.

## **Chapter 6 Conclusion and Future Work**

In this thesis, we designed and implemented a cloud based building energy monitoring system. Several test cases are adopted to test the functionality and robustness of this system. Through our web interfaces, users can easily get the near real-time as well as the long term energy usage information to help them make according policies regards to reducing the energy waste.

Future work can be done to enlarge the system to support mobile devices. Mobile Applications can be developed for smart phones so that people can access the energy information via the phone's application anytime and anywhere.

## References

- [1] Walsh, J.H., 2000, Projection of Cumulative World Conventional Oil Production, Remaining Resources and Reserves to 2050, March 2000.
- [2] U.S. Department Of Energy, International Energy Annual 1999 (IEA 1999): World Energy Overview.
- [3] U.S. Department Of Energy. Buildings Energy Data Book, 2008. <http://buildingsdatabook.eere.energy.gov/>.
- [4] U.S. Department Of Energy. Energy Information Administration. Commercial buildings energy consumption survey, 2003. <http://www.eia.doe.gov/emeu/cbecs/>.
- [5] “Pilot Program” <http://www.google.com/powermeter/about/>
- [6] J. Petersen, V. Shunturov, K. Janda, G. Platt, and K. Weinberger. Dormitory residents reduce electricity consumption when exposed to real-time visual feedback and incentives. *International Journal of Sustainability in Higher Education*, 8(1):16–33, 2007.
- [7] G. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 1992.
- [8] X. Jiang, M. Van Ly, J. Taneja, P. Dutta, and D. Culler. Experiences with a high-fidelity wireless building energy auditing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 113–126.
- [9] X. Jiang, S. Dawson-Haggerty, P. Dutta, D. Culler. Design and implementation of a high-fidelity AC metering network. In *Information Processing in Sensor Networks*, 2009. IPSN 2009.
- [10] “Energy Hub”. <http://www.energyhub.net/>.
- [11] “Green Box”. <http://www.getgreenbox.com/>.
- [12] D. Petersen, J. Steele, and J. Wilkerson. WattBot: a residential electricity monitoring and feedback system. In *Proc. of the 27th international conference extended abstracts on CHI*, 2009.
- [13] N. Motegi, M. A. Piette, S. Kinney, K. Herter. Web-based energy information systems for energy management and demand response in commercial buildings. Lawrence Berkeley National Laboratory, LBNL Report 2003.
- [14] “Sentilla”. <http://www.sentilla.com/>.

- [15] SSH Tunneling protocol. [http://en.wikipedia.org/wiki/Tunneling\\_protocol](http://en.wikipedia.org/wiki/Tunneling_protocol).
- [16] Defining and Measuring Cloud Elasticity. KIT Software Quality Departement.  
Retrieved 13 August 2011.
- [17] Economies of Cloud Scale Infrastructure. Cloud Slam 2011. Retrieved 13 May 2011.
- [18] Mills, Elinor (2009-01-27). Cloud computing security forecast: Clear skies. CNET News. Retrieved 2010-08-22.
- [19] “TinyOS” <http://www.tinyos.net/>
- [20] “TinyOS Programming” <http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf>
- [21] “Crossbow’s Telosb datasheet” [http://www.willow.co.uk/TelosB\\_Datasheet.pdf](http://www.willow.co.uk/TelosB_Datasheet.pdf)
- [22] “BeagleBoard” <http://beagleboard.org>
- [23] “Features, OMAP3530, Texas Instrument” <http://www.ti.com/product/omap3530>
- [24] “The Ångström Distribution” <http://www.angstrom-distribution.org/>
- [25] “JamVM” <http://jamvm.sourceforge.net/>
- [26] “The Java Native Interface Programmer's Guide and Specification”  
<http://java.sun.com/docs/books/jni/download/jni.pdf>