CSM 1

Real-Time Electricity Monitoring Wireless Sensor Network

Field session 2009

CLIENTS:
Dr. Qi Han
Alan Marchiori

WRITTEN BY:
Charles Davis
Casey Franek

Submitted June 18, 2009

1. **Abstract**

Rising energy costs and concerns over global warming have caused some people to question their own personal energy usage habits. The goal of this project is to develop a system by which a household would be able to collect data about their personal consumption using wireless sensors. The members of the household will also be able to view their consumption in real time by accessing the information using a webpage.

The information in this project lays out how the information gets from the sensor to the end user. To accomplish this multiple technologies were used, including TinyOS, nesC, Java, SQL Server, PHP, AJAX, and amCharts.

2. **Introduction**

Wireless sensor networks are an increasingly popular way to gather accurate information about the world around us in real time. They are portable, easy to set up, and convenient to use. Since almost any kind of information can be gathered, the possibilities for developing wireless sensor networks are almost boundless.

The specific wireless sensor network being developed in this project will be used to gather information about electricity consumption. This will be accomplished by using wireless nodes to collect information from sensors connected in series with electrical lines as well as sensors that can be externally attached. The information will be gathered into a central database and will be accessible via a webpage so that clients can see their power consumption habits.

The overall goal of this project is for people to be able to assess their power consumption habits and see what they are willing to change, leading to a more efficient use of electricity. Another use for the information gathered may be to set up a system where some appliances, such as air conditioning units, could alter their power draw based on the consumption of the rest of the house in an attempt to reduce draw at peak usage times.

3. **Requirements**

3.1 Functional Requirements

1. Develop sensors to collect electrical current information
2. Develop hardware and software interface from sensor to node
3. Develop wireless network of sensors
4. Transform raw sensor data into meaningful values
5. Gather information in a database
6. Centralize configuration information for maintenance
7. Develop web page to display data
8. Allow multi-node sensor networks

The first six functional requirements are the most important. Collecting information from the sensors for analysis is the ultimate goal, as any method of data access is variable depending on the needs of the client, and can therefore be developed separately. The client informed us that having a multi-node sensor network was low priority, but it would

make the project more meaningful if a sensor group supported more then a single sensor node.

3.2. Non-Functional Requirements

1. Use TinyOS (Version 2.1.0) on the nodes and base station
2. Use nesC to program the nodes and base station
3. Use Java to convert raw sensor data and make data submissions
4. Develop a SQL Server 2005 database to collect data
5. Use Ajax, PHP, and a Flash-based chart to access data for display

The client has specified that we need to use TinyOS for the nodes and base station to meet hardware limitations, and the use of the nesC language is necessary when working on a TinyOS platform in order to interface with the system and the sensor hardware. nesC, short for Network Embedded Systems C, is an extension of the C language specifically designed for small event-driven systems, particularly wireless sensor nodes or *motes*, as they are often called. TinyOS itself is of an entirely novel design. It consists of a number of modules written in nesC, and is almost entirely event-driven, managing concurrency primarily with interrupts and with tasks. Tasks run atomically to one another, but interrupts may preempt tasks. Like nesC, it is designed to operate within the constraints of a wireless sensor node.

The use of Java was preferred because the Java versions of the TinyOS client tools and code libraries were the most thoroughly tested, resulting in a large pool of existing stable code for us to use.

The client decided to use SQL Server 2005 as the database backend because a server he gave us already had the software installed. He also suggested the use of Ajax so the data could be displayed in a dynamic webpage, without the need to constantly refresh the page. For the chart itself, he suggested amCharts, a collection of Flash-based charts free for general use.

4. **High-level design**

Our system begins with the sensor node, which polls a sensor and passes the information to a base station, which in turn forwards the packet to the processing client through a serial port. The processing client converts the raw data into power usage information, and submits it to a database. A client can view the data stored in the database through a web browser, and the information will be displayed in a graph showing the energy consumption along a timeline. See Figure 1 for a pictorial description.

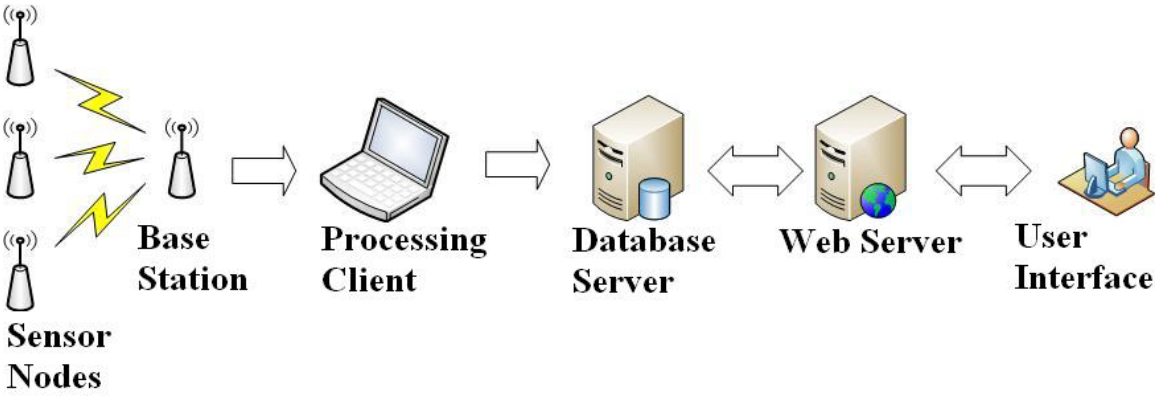

Figure 1

5. **Detail design**

5.1 Sensor nodes and base station

The sensor node begins by polling the sensor and collecting readings over a period of time, by default five seconds. The sensor node polls sensor data directly from the Analog-to-Digital Converter (ADC). The rate at which this occurs is also configurable, and defaults to 3125 Hz.

The sensor node is also capable of dumping the data it collects to its built-in serial port. A small button on the node, labeled “user,” initiates collecting a trace of the current and voltage sensors over a few seconds. After it has filled its internal buffer, the node will then break the buffer into pieces, wrap each piece in a packet, and send it to the serial port, where it can be analyzed. To initiate another trace, the user must press the “user” button again.

After five seconds of data have been collected, the sensor node will send the data to the base station (see Figure 2), using the Collection Tree Protocol (CTP).

In order to balance sampling the sensor and sending the data, the sensor node calculates a running average of all readings since the last send. This setup minimizes network traffic, as well as power usage in the sensor node. If, however, the node is currently capturing a trace because its “user” button was pressed, it will also save each reading in its internal buffer until it is full.

The base station listens for packets from the sensor nodes. When a packet is received it forwards the information over the serial port and returns to listening for more packets.

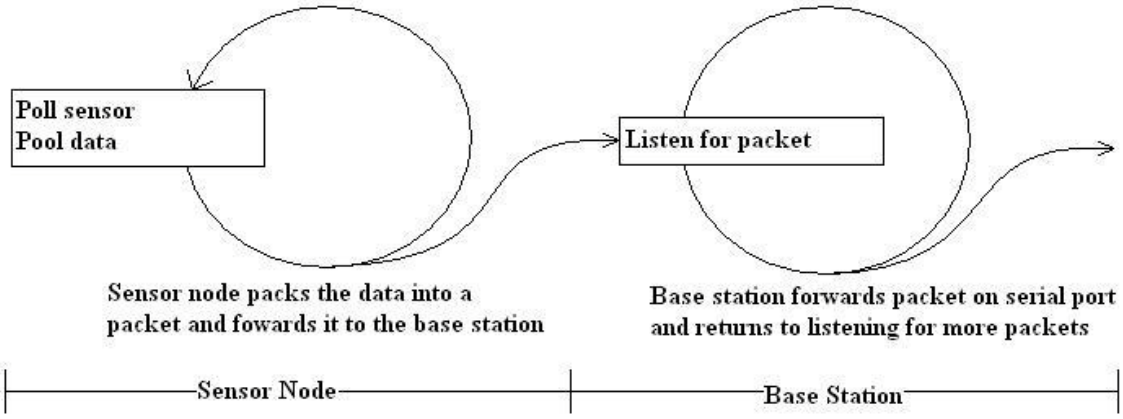

Figure 2

5.2 Processing client

The processing client is written in Java and is split up into two distinct threads. The first thread listens to the serial port, and when a packet is received it parses the packet and inserts the information it extracts into a queue. After the information has been added to the queue it returns to listening for packets on the serial port (see Figure 3).

The second thread pulls information out of the queue and turns the raw sensor data into power information. Afterwards a connection to the database is established and the converted data is submitted to the database.

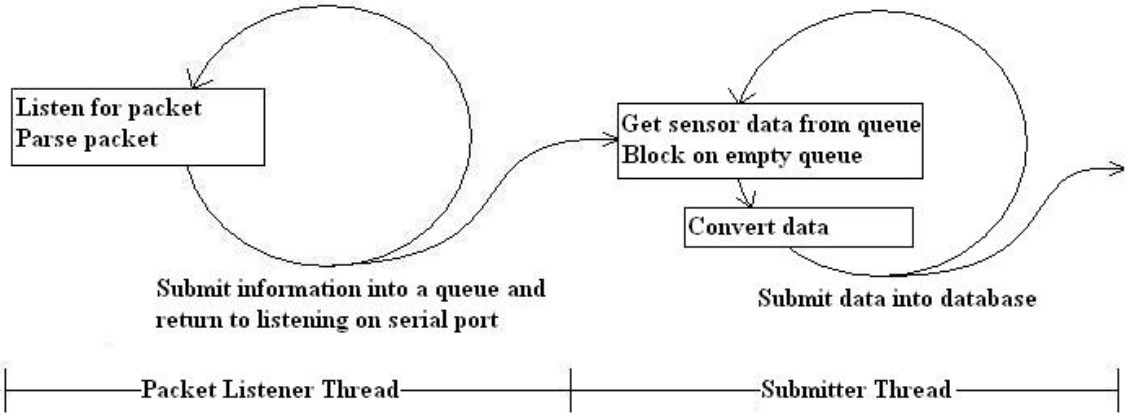

Figure 3

5.3 Database design

The database schema is shown in Figure 4.

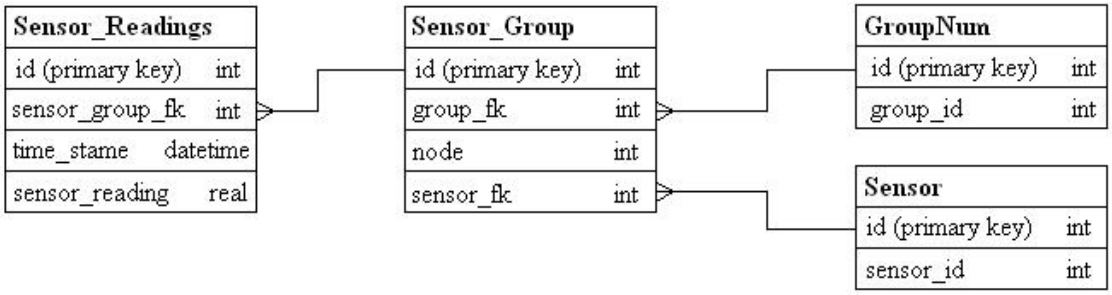

Figure 4

The database consists of four normalized tables used to hold the sensor reading information. The first table is the GroupNum table. It has two columns. The first is the "id" column, which is the primary key. The other column is the "group_id" column. A single group id is intended to be a set of sensor nodes that are logically related to each other, such as all the sensors in a single house or apartment.

The second table is the Sensor table. Similar to the GroupNum table, the Sensor table only has an "id" column and a "sensor_id" column. Originally, the idea behind this table was to allow the sensor nodes to be able to use multiple kinds of sensors; currently there is only one kind of sensor used. In the future, however, the client could add more types of sensors; this table would then allow us to identify the readings coming from each sensor.

The third table is the Sensor_Group table. The Sensor_Group table has four columns. As always, the "id" column acts a unique identifier and is the primary key of the table. The "group_fk" column stands for "group foreign key," and is used to form a relationship between the Sensor_Group table and the GroupNum table. Similarly, the "sensor_fk" column forms a relationship between the Sensor_Group table and the Sensor table. Finally, the "node" column represents a single node. Where a group id in the GroupNum table represented a single house, the node column would represent a single outlet.

The last table is the Sensor_Readings table. It has four columns. The first column is the "id" column, and again it serves as the primary key. The "sensor_group_fk" column forms a relationship between the Sensor_Readings table and the Sensor_Group table. The "time_stamp" column holds the date and time of the reading, and the "sensor_reading" column holds the amount of power consumed at that time as returned by the sensor node.

To simplify accessing and maintaining the database, we wrote stored procedures. These procedures reside on the server itself and are called instead of using Structured Query Language (SQL) statements directly, which should allow for fewer changes to the processing client and web server if the database schema is ever changed.

5.4 Webpage

The web server allows clients to view their power usage in real time. In this design, the web server is actually the same computer as the database server. We made this decision

for the sake of efficiency and compatibility.

The web page itself contains a script that makes requests to the server. The data are returned in Extensible Markup Language (XML) format, and the script can use the XML Document Object Model (DOM), a standard for representing XML data as a tree of objects, to manipulate the data.

The script actually makes three types of requests to the server. In the first type, the script asks for a list of known groups from the database. In the second type, the script has selected a group, and asks for a list of nodes that belong to the group. In the third type, the script has selected a node in a group, and continually queries every five seconds for power usage data from the database.

The web server responds to requests from the web page script by executing PHP Hypertext Processor (PHP) scripts. These scripts query the database for the desired information using a stored procedure, and emit an XML document that is returned to the web page script.

When the web page script receives power usage data, it feeds the XML document into a Flash-based chart. This chart and the data it displays are controlled by various inputs on the web page. The script is aware of these inputs, and adjusts the requests and the operation of the chart accordingly.

These inputs are as follows. The first set of inputs is located above the graph. It allows the user to select a group and a node inside that group. When the user selects a new group, the script automatically refreshes the list of nodes inside that group. When the user selects a new node, the chart is then reloaded with the data from the new node.

The second set of inputs is located below the graph. The user can use these inputs to specify a span of time containing the data to request from the server. The user can specify a specific date, or any time relative to now.

6. **Implementation Details and Results**

The choice of what to use was largely determined for us by what we already had available. The client specified that we were to use TinyOS and nesC on the sensor nodes. The use of Java to develop the processing client, though not specified, was a natural decision because of the amount of existing, stable, heavily tested code. The processing client itself uses TinyOS's standard Serial Forwarder to listen to the serial port, and Microsoft's own freely available SQL Server JDBC driver to connect to the database and insert the data.

For the database, the client preferred the use of SQL Server 2005 because he already had it installed on his server computer. For the web page, the client also suggested the use of Ajax, a technique that uses JavaScript and XML data to implement a dynamically updated web page. For the chart, he referred us to amCharts, a collection of Flash-based scriptable charts free for general use.

For the web server, we chose PHP because of its availability and reputation, and also for the similarity of its language to C/C++. The web server itself, which runs Windows Server 2008, is the same computer that hosts the database, since this made the task of connecting to the database from PHP far easier. To actually serve the web page, we used Windows Server 2008's built-in functionality, Internet Information Services (IIS).

Another issue we encountered involved the PHP scripts. The script that returned the chart data emitted them in the wrong format. Specifically, a required element was missing from the output XML document. For this reason, the chart software refused to display the data. The fix was to add this element.

7. **Future Directions**

What we have provided our client is not a final solution, but a building block for future development. With that in mind we have made some decisions to allow easy changes to code.

A. The processing client has the code to calculate power isolated, so that it can be changed easily without searching through the rest of the code.

B. The database uses stored procedures to access information instead of having programs use raw SQL.

C. The GroupNum, Sensor, and Sensor_Group tables were designed with the future intention of adding another column that would hold a text description of each group or node or sensor.

D. The work performed by the Ajax script is separated into functions.

This project is actually part of a much larger project that our client has undertaken. The goal of his project is to analyze and reduce power consumption through the use of wireless sensor networks such as this. Collecting data on power consumption is a key step in both analyzing and reducing power consumption. For if people do not know they are using more power than they should, how can we expect them to reduce their consumption? This is a good first step.

8. **Glossary**

ADC – Analog-to-Digital Converter, a device that accepts a varying analog voltage input, and produces a digital integral output. This output is suitable for use in a computer program, whereas the analog input is not. To be of any use, the raw output must be converted back to a voltage, and from there to whatever unit the sensor itself measures.

Ajax – Asynchronous JavaScript And XML, a technique that combines the two named technologies, primarily used to create dynamic web applications.

CTP – Collection Tree Protocol, the standard protocol in TinyOS for sending data from any mote to a designated root.

DOM – Document Object Model, a standard for representing HTML and XML documents as a tree of objects.

Foreign key – A column in a table that refers to a primary key in another table. Foreign keys are used to form a relationship between two tables so that two tables can be joined during a query. While a foreign key is usually not unique the value it the other table must be.

HTML – HyperText Markup Language, the *de facto* and *de jure* standard for publishing documents over the Internet.

Java – a common cross-platform environment, with associated programming language, for developing applications. It is extremely popular in web-based applications.

JDBC – Java Database Connectivity, a Java framework for connecting to and interacting with databases.

Mote – a wireless sensor node. Refers to any piece of hardware that is designed to operate as part of a wireless sensor network.

nesC – Network Embedded Systems C, a language designed for small real-time systems. It is an extension of the C language specially designed for TinyOS's feature set. Theoretically, it could be extended to other applications as well.

PHP – A recursive acronym for PHP Hypertext Processor, a server-side technology that preprocesses HTML and XML documents before a client receives them. HTML and XML documents that can be preprocessed contain sections of PHP script.

Primary key – A column within a table that can be used to uniquely identify each row in a table.

SQL – Structured Query Language, the *de facto* standard language used to query a relational database, such as Microsoft SQL Server or MySQL or Oracle, for data. The acronym is often pronounced "sequel."

Wireless sensor network – a network of extremely small, embedded computers. They are designed primarily for collecting data and/or distributed control over certain devices.

XML – eXtensible Markup Language, an increasingly popular language for describing languages that in turn are used to describe ("mark up," hence the name) data.

9. **References**

1. TinyOS Enhancement Proposal 113: Serial Communication, ://www.tinyos.net/tinyos-2.1.0/doc/html/tep113.html

2. TinyOS Enhancement Proposal 116: Packet Protocols, ://www.tinyos.net/tinyos-2.1.0/doc/html/tep116.html

3. TinyOS Enhancement Proposal 118: Dissemination, ://www.tinyos.net/tinyos-2.1.0/doc/html/tep118.html

4. TinyOS Enhancement Proposal 123: Collection Tree Protocol (CTP), ://www.tinyos.net/tinyos-2.1.0/doc/html/tep123.html

5. The XMLHttpRequest Object, ://www.w3.org/TR/XMLHttpRequest/

6. Document Object Model (DOM) Level 2 Core Specification, ://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

7. Document Object Model (DOM) Level 2 HTML Specification, ://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/

8. Extensible Markup Language (XML) 1.0 (Fifth Edition), ://www.w3.org/TR/2008/REC-xml-20081126/

9. amCharts Bundle: Line & Area, Column & Bar, Pie & Donut, Scatter & Bubble charts, ://amcharts.com/docs/v.1/bundle

10. PHP: PHP Manual – Manual, ://www.php.net/manual/en