Smart Home Using Sensor Motes

Eric Olsen
Department of Computer Science Engineering,Binghamton University
eolsen4@binghamton.edu

Nitish Mehta
Department of Electrical and Computer Engineering,Binghamton University nmehta4@binghamton.edu

Junyang Shi
Department of Computer Science Engineering,Binghamton University
jshi28@binghamton.edu

ABSTRACT

Home automation or a 'Smart Home' environment is defined as building automation for the home and will likely have a variety of applications in the future. Smart Homes involve the control and automation of lighting, heating, ventilation, air conditioning, and security, as well as home appliances such as washers and dryers. Wi-Fi is often used for remote monitoring and control, but devices must consume large amount of energy in order to work under Wi-Fi. In our paper, we try to use IEEE 802.15.4 technical standard which defines the operation of low data rate wireless personal area networks. For our constrained sensor motes in our Smart Home, using low power network protocol is essential, and we successfully build a low power consumption network in this project.

In our implementation, we use two different devices, TelosB and SensorTag for sensing. The Raspberry Pi can collect this data using a serial port which is connected with a base station mote. Through an Ethernet cable, the Raspberry Pi can directly forward data to our Laptop, then a user interface displays temperature, light and door status information to users.

CONCEPTS

• **Wireless Sensor Networks → Smart Home Environments**

KEYWORDS

Wireless Sensor Network, Smart Home, telosB, SensorTag,

1 INTRODUCTION

As Wireless Sensor Networks continue to make their way into mainstream technologies, it is only natural that they will occupy an integral place in the household. From providing temperature and lighting data to home security, that tasks which can be automated through the use of sensors are extensive and varied. To this end our group approached creating a Smart Home environment with a specific goal in mind: creating a network that could display important home information in a clear and concise manner. To accomplish this we brought together a variety of technologies, including TelosB sensor motes, a TI SensorTag, a Raspberry Pi, and Linux, TinyOS, and Contiki operating systems to create a robust and responsive network.

Section 2 describes our implementations of various technologies used the Sensor Network.

2 Design and Implementation

2.1 Overall Design

The overall design of the system is based around a star topography, with the various telosB motes feeding into a singular base station. These motes then send temperature and lighting sensing data to the base station at regular intervals, to be sent and used further down the line.

Originally, we had linked this base station directly into the serial port of the user device. This provided the most responsiveness for the system, however portability was subsequently limited. Due to this, we offloaded base station support to a Raspberry Pi running the tinyOS toolchain on top of raspbian which then itself was linked to the user device, either through Ethernet or Wi-Fi. We then found that the most effective method of extracting and sending the packet data was to run a SerialForwarder on the Pi, which could then be hooked into by a second SerialForwarder on the user device. This is preferable to sending the fully rendered GUI from the Pi for two reasons. First, latency is greatly decreased, as there is significantly more overhead in sending a fully rendered image than there is sending a stream of individual packets. Second, by opening a pair of SerialForwarders, other applications can be linked to the packet stream on both the Pi and user device, allowing the health of the system to be monitored.

Finally, it is left to the user device to render the GUI image, however this is trivial as the interface is minimal and far below the abilities of modern devices.

2.2 TI SensorTag

The multi-standard SensorTag Kit, based on the SimpleLink ultra-low power CC2650 wireless MCU is being used as Magnet Sensor to know whether the door is "Open" or "Close".

ContikiOS: an open source, popular operating system for Internet of Things low-power, low-memory devices, is used for the SensorTag. The C language is used as the programming language supported by ContikiOS.

cc26xx-magnet-sensor.c: The event-driven thread of ContikiOS is used to read values from a magnet sensor. The reed-relay, acting as a magnet sensor, gives a binary value as "Door Open" or "Door Closed".

Although the SensorTag supports Zigbee and Bluetooth to communicate wirelessly, the interoperability limitations between the base station running TinyOS and Contiki OS, as well as limited support for Bluetooth in Contiki, forced us to use the UART interface to display the state of the door. We came to know about such limitations during the project development phase,

where we also learned that packets sent by the TelosB's to the SensorTag over Zigbee can be read, but the inverse doesn't work.

2.3 TelosB Sensor Motes

In this project, we used TinyOS for Telosb programming. The programming language inherent to TinyOS is NesC, a dialect of standard C. NesC's key feature is event-driven programming. So for the base station and different sensors, they are all driven by different kinds of events.

The TelosB which is connected with a Raspberry Pi runs standard base station code in TinyOS. For our base station, there are two main functions. One is the collection of data from different kinds of sensors, and the other is to forward data to the Raspberry Pi. The two necessary interfaces are:

```
interface Receive as RadioReceive[am_id_t id];
interface AMSend as UartSend[am_id_t id];
```

For sensor part, TelosB runs with several different stages. Hard devices need booting first. After this event, we can call Radio Control start and Serial Control start. After devices have turned on their own radio, timer can start periodically to trigger more events. The timer is essential part for sensing. Every time when timer is triggered, mote will collect data into device's buffer. In this project, we improve the operation efficiency by packing 10 data into one sending. To achieve different types of sensor, we just need to change some components in TinyOS configuration file. This is an efficient way to integrate different types of sensors.

2.4 User Interface

The user interface is built off of the Java Swing and AWT libraries. On the back end, the program hooks into a SerialForwarder running on the local machine on port 9002. The code is an extension of the User Interface provided by the Oscilloscope application native to tinyOS, and as such, provided an effective baseline from which we could create an interface for our own program. The code of the UI is split across 5 java files, each handling a key piece of processing. They are:

ColorCellEditor.java: A simple object used to modify the graph color associated with each temperature node

Data.java: An object that stores the data being sent in by each sensor mote. It stores the data by sensor type and ID, as well as providing important functions such as determining if old data was stale, determining if incoming data is valid, and providing the newest values to the graphic rendering objects. *Graph.java*: As this particular form of graph is not native to java Swing/AWT, it had to be created from scratch. The bulk of this was done by tinyOS developers, however several modifications had to be made. Support for multiple motes to be graphed at once needed to be added, as well as general formatting changes. Additionally, minor bug fixes were necessary, such as the graph occasionally not centering itself on the data being displayed.

Node.java: An abstraction of the sensor motes which are sending data into the UI. This is done to allow pertinent mote data to be stored as a package, and is actually the object type that Data.java is storing. In addition to storing mote information, the Node class has the ability to erase and refresh mote info if the sensor is removed and readded to the network, or if the mote is rebooted.

SensorApp.java: The main driver of the program. This is where instances of Window and Data are stored, as well as where events are triggered, such as receiving a packet or having a new mote added to the network.

Window.java: The object actually rendering the UI. This stores an instance of Graph, as well as a pair of subclasses, MoteTableModel and LightTableModel, which display the Temperature motes and the Lighting motes, respectively. Additionally, Window provides user control to change the graph format and clear the UI.

3 EVALUATION

We tested our sensor network against a variety of scenarios to ensure that the sensors would be able to deal with various inputs. For our temperature sensors, we set up a test wherein the ambient room temperature would be displayed alongside the temperature of hot and cold objects. The objects that we measured against were metal cups alternately filled with hot and cold water. This setup was used due to the effective heat conduction of metal, and the fact that water will hold a constant temperature for a relatively long time, allowing for sufficient testing to be performed. During this, we were able to increase the temperature of the sensor next to the cup by 10 degrees Celsius, while the sensor recording ambient temperature remained unchanged. When cold water was placed next to the cup, the sensor's temperature was decreased by 7 degrees Celsius, showing that the TelosB motes are more sensitive to heat increases than decreases.

One issue we encountered when running this test is the relative lack of sensitivity of the TelosB. In order to register any appreciable change in temperature, the mote had to be placed on its side, with the sensor in direct contact with the cup. This, however, was a hardware limitation rather than a flaw in the system.

For our lighting sensors, we performed testing in a well-lit room during the day, a shaded room during the day, and a room at night. At night, the sensors responded accurately and quickly, as the threshold between the ambient lighting in the room and the lighting with an overhead light on was considerable. In the shaded room, tests would register accurately, but the threshold that we used had to be modified depending on the time of day, as there was much less difference between ambient lighting and actually having lights on in the room. We initially used a threshold of 0x75 lumens, however, this was registering false positives in a moderately lit room. At a threshold of 0x95 lumens, the sensors would respond accurately in a shaded room, but less so at night. Finally, we found that between a room at nighttime and a shaded room, 0x85 lumens returned the best results.

This being said, our lighting sensors still faced considerable issue in a room that was well-lit to begin with, as the addition of overhead lights often made a trivial difference, and the sensors were not sensitive

enough to register the change with any degree of accuracy.

For our magnetic sensor, which we used to determine if a door was open or not, we fashioned a simulated door and wall out of a cardboard box. A flap was cut out of the side of the box which became the 'door' and to which we fashioned a magnet. On the inside of the box, along the 'wall' we attached our magnetic sensor. This test proved to be quite effective. Despite the fact that we had both a low-sensitivity sensor and a low-power magnet, we were able to record almost perfect accuracy in determining whether the door was open or not. The only times when this setup would record false positives is in the case when the door is opened and shut very quickly, as the sensor does not have time to register the change, send the information, and register another change. This being said, so long as the user maintains the position of the door for at least 2 seconds, there were no inaccurate readings.

4 CONCLUSION

The goal of this project was to create a heterogeneous sensor network which could act as a smart home environment. By the end of development, we had successfully accomplished the majority of our goals, starting with the understanding and implementation of the TelosB sensor platform. Included in this was development using the NesC language, the programming language inherent to TinyOS. This was used for both the Temperature and Light sensors. With this, we also learnt about Ti SensorTag and ContikiOS to develop code for a magnet sensor. Further in this, we developed a robust and user friendly graphical user interface using java.

For communication between the base station and sensor motes, we used star network topology. We were successful in creating an ad-hoc network between the base station and all the TelosB motes. With this, we also got hands-on experience with the limitations present in communicating between devices running two different operating systems: TelosB running TinyOS and SensorTag running ContikiOS, by attempting to use example code provided by the OSes to communicate between them.