Street Based Mote

Members

Brian Matthews
borg286@gmail.com

RashinBolkameh
rashinbe@yahoo.com

Raheem alhamdani
r.alhamdani@utah.edu

Group e-mail
null-pointers@googlegroups.com

Group website
http://www.eng.utah.edu/~alhamdan/Mote

April 23, 2008

Abstract

We seek to create a dynamic mesh network of wireless sensor nodes, or motes, to reduce traffic and accidents and keep drivers aware of traffic conditions on their route. Motes will be installed on street lights and in cars, all communicating with each other and reporting to a central computer that can monitor and track weather and traffic received from the mote to help keep traffic to a minimum.

Project summary

We want to design a traffic sensor network that will monitor traffic and provide local information to motes based in cars within range. These motes will be attached to street lights powered by the city and creating a mesh network through the city. We plan to demonstrate its functionality with 3 Motes in a straight line in the MEB parking lot with a car based mote traversing the length. We will also write software that retrieves maps from google.maps.com and converts them to graphs. We will then partition the city around street lights so that each mote would be able to provide the car based mote with local street information such as road graphs, and dream features such as gas stations, restaurants, motels, etc.

Our original hardware component will include a dipole antenna that we hope to upgrade to a bi-cone for greater range and noise reduction. The motes will be able to gather weather information as well as local traffic, and send these back to a central computer for analysis. The motes have the capability to control the lights and thus enable the central computer to control the lights for a faster commute.

Tasks

- **Write mesh network software**

A mesh network is similar to a graph, but enables for dynamic updating of nodes. We will be using open source code found on the web as a starting point for our network software. The main difference is that our software will have to be able to handle very dynamic configurations of the network such as passing car based motes. Car based motes will be coming in and out of the street based mote's range which we will need to be able to handle very quickly. All this information needs to be reported efficiently and streamlined so that a central computer can make sense of it all. We will create a graph representation that overlapping graphs can be joined with minimal computation.

- **Write Google maps software/Central Computer program**

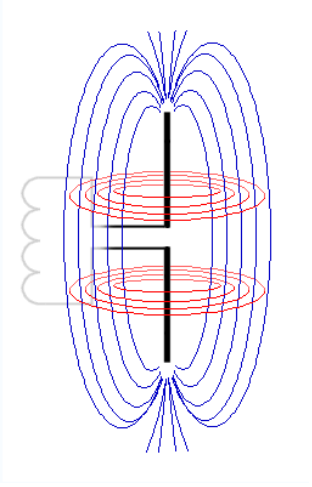
By writing the Google Maps software we will be able to offer the Google services like restaurants, gas stations, motels..., and we will be able to extract out road graphs. This resource will be part of a program that will be always running on a central computer that will get all the packets from the motes and organize them for analysis to the user. If new information is available to google, the motes will receive them shortly thereafter, information like traffic, weather, events, etc. Thus we will maintain the most up-to-date traffic information available to the user/driver.

- **Minimize response time**

This will be software intensive. We don't know how fast we can transmit and receive data (handshaking) with the mote as we have never used them before. We assume with their 250 kbps, High Data Rate Radio we can make sure handshaking time is kept to a minimum. We also hope that interference will not be an issue. We will try and create interference for testing in an attempt to create the worst case scenario, but we can only do so much.

- **Maximize range with custom antenna (Dipole)**

Using custom antennas (Dipole for example) improves the reception for the mote and directs the signals that the mote transmits and recieves. Therefore, this improvement provides benefits in terms of greater range and increased data transfer speeds. A dipole antenna is an antenna with a center-fed driven element for transmitting or receiving radio frequency energy, it can provide up to 2.14 dBi of peak antenna gain, which will be perfect in our project.



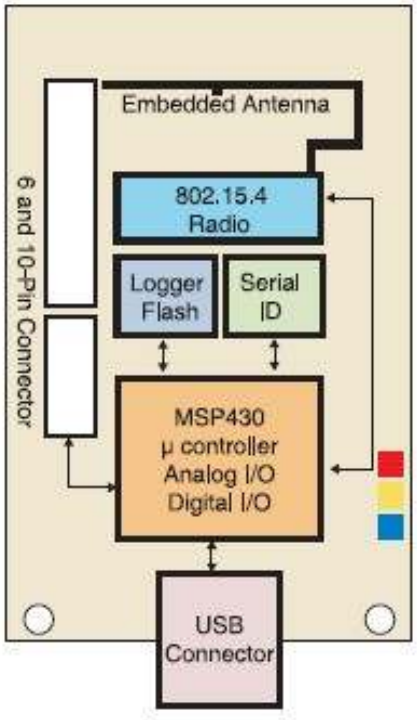
Specific Task Interfaces

As per the Hardware interfaces there is very little to be done. We had initially planned on using sensors for weather, and temperature, but due to the advanced state of the motes we are using, those sensors come prebuilt on the mote. We had also wanted to use a solar panel in conjunction with batteries and manage power with a power unit, but that idea got superseded by the fact that we have city power to use from the street lights themselves. In light of this we will be simply using a standard household multi-voltage transformer and set it to the voltage for optimal transmission range. We will assure that the transformer bought will have a current that is compatible with the mote. The only other hardware piece is the antenna. The only interfacing we need is to program the mote to know that it now has an external antenna with different impedance than its internal antenna. The band we will be communicating on is 2.4 to 2.4835 GHz, a globally compatible ISM band. This will lead to an antenna about 7 cm in length.

The main part of interfacing that is to be done is in software with the other team. We will need to know specifics of how 802.15.4 works so as to minimize the time spent in communication, thus reducing the power consumed by the transmitter. We plan on using a form of XML to transfer data and messaging. This gives us lots of freedom with what and how we communicate. Currently we have 3 types of messages: update, request, special. Almost all data fields (tags) are optional, and will be supported by both teams. Handshaking will prove to be difficult because we don't want just anyone telling our motes what to do or what to send. Encryption will take a good part of protocol development.

TelosB mote Specifications

- Open-source Operating System
- IEEE 802.15.4 Compliant
- 250 kbps, High Data Rate Radio
- TI MSP430 Microcontroller with 10kB RAM
- Integrated Onboard Antenna which we will connect to an external antenna
- Data Collection and Programming via USB Interface
- Integrated Temperature, Light and Humidity Sensor
- Visible Light Sensor Range, 320 nm to 730 nm
- Visible to IR Sensor Range, 320 nm to 1100nm
- Temperature Sensor Range, -40°C to 123.8°C
- Humidity Sensor Range, 0-100% RH



Testing and Integration Strategy

Testing will be heavy on the software side; agreeing on protocol, testing that the motes can hear each other and understand each other is key. NesC(native language of TinyOS, covered later) code examples will be helpful in getting our feet wet, and helping us get to our feet and off to a good start. There are examples like how to make the lights blink, and sending a “hello world” message. Initial testing will be to see if we can get the lights to blink, then send a basic “hello world” message across, then create a network, then try and cut off a central node and see if the other motes can handle it, then predict speed and trajectory and know when the car mote will leave range. These will be key milestones to show our progress.

Group communication plan

As per communication for out group we plan on using Google Groups. Google groups let us upload code changes and send the entire group notifications. The other mote team will also be able to observe what's going on with our team. One of our members will attend every other of group meeting of the other mote team so as to keep both teams on the same page.

As per source control, a rudimentary method of downloading and uploading a zipped version of the code is what we'll use. WinMerge will be used when multiple checkouts are done. WinMerge is a free program that can be used to analyze 2 documents and lets the user merge the 2 files together. We will investigate other means like Google Documents so everyone has the most up-to-date version of the code. The nice thing about Google Documents is that when a member makes changes to the document, that change is reflected wherever the document is available, be it on our Google groups page, presentation, e-mail attachments, so forth. The only problem is that one would need to extract the code from the document and paste it into the program, which is a little inconvenient. Whichever is optimal as found through testing will be used.

Project Mote Schedule

Members : Brian Matthews, Rashin Bokameh, Raheem Alhamdan

Hardware & Software Tasks	Completed by:	Apri	May	Jun	July	Aug	Sept	Octo	Nov	Dec	Janu	Febr	Mar	Apri	May
Get TinyOS up and running.	Raheem														
Write mesh network software	Al														
Write Google maps/Central computer software	Rashin + Other team														
Order 3 motes, antenna,power converter	Rashir														
Investigate antenna configurations	Rashir														
Agree on message protocol/data rep.	Al + Other team														
Test program to blink lights and read sensors	Raheem and Rashin														
Test range w/o antenna	Al														
Minimize response time	Brian														
Maximize range with custom enterna	Rashir														
Test motes with car based motes	Al + Other team														
Test + Debug	Al														
Demo	Al + Other team														

The Operating System

The most widely used operating system for motes is TinyOS, there are several ways to download and install TinyOS on a PC or a Mac. I found that using Vmware Server console to creating a XubunTOS virtual machine within it is one of the simplest ways. From a single desktop the Vmware server can creates and manages Multiple virtual machines in parallel. So now we have VMware Player and we can run a pre-configured XubunTOS virtual machine image inside it.

There are also several editors and IDE's for TinyOS. XubunTOS comes with an Emacs editor, but Eclipse is better editor with great plug-ins. It has numerous features that could be essential to writing better code for our motes. It also has what is called an example wizard that uses the examples provided with the TinyOS.

A quick lesson is included to introduce the basic concepts of TinyOS and the nesC language in which the system is written. It includes a quick overview of the nesC language concepts and syntax to help getting started with programming in this environment.

TinyOS and nesC

NesC is a new C-like syntax language. It's used for programming structured component-based applications. Components are how TinyOS is designed and structured. It includs libraries, system files and applications. NesC language was created for embedded systems such as the motes we will be working on, but it must support the TinyOS concurrency model. Structuring, naming and linking application together is the idea behind this component based applications running on TinyOS. It gives application designers the freedom to mesh applications into each other and compose a complete, concurrent system. At the same time still perform extensive checking at compile time. TinyOS has few concepts that are expressed in nesC, such as components with bidirectional interfaces, they use the same model of concurrency based on tasks defined in the model and hardware event handlers,. It also detects data races at compile time.

Milestones

Some basic milestones will be getting TinyOS up and running. We hope to then get the lights to blink. The Hello World example will also be a big achievement. Getting a network created and wirelessly reported to a computer will be our next goal. After that more complex messages will show we are on our way. If our antenna can get 300' we will have achieved desired range. Our final milestone is when a car based mote can traverse from one end of a network to the other.

Risks

Some risks we foresee are an insufficient gain from the antenna, thus limiting our range, meaning more motes per square mile, if this were to be deployed in a city. As we are unfamiliar with the functionality provided us through TinyOS we might need to delve deeper into nesC, the programming language for TinyOS, upon finding programming stumbling blocks. We assume that the motes will be able to provide us with received signal strength, from which we can extrapolate distance to neighboring motes. Even if this functionality is not provided with the mote we've chosen we will find alternative means of getting that information. The car based motes will be equipped with GPS, from whom we can get coordinates. We also predict that reaction time (hand shaking time) might be too long for practicality.

Another risk is weather proofing. We will only demo in nice weather, but the motes need to be able to withstand rain, snow, earthquake…. Along with weather is the reduction of signal strength. We don't know how much our range will be reduced. It could be so much that a connected network is impossible.

BOM and Vendors

TELOSB MOTE w/ SENSOR SUITE	$ 110.00	Willow Tech. +44 (0) 1342 835234 CrossBow (408) 865-3300
Antenna (di-pole)	$0.50	The Radio works, Jim (804) 484-1040
Variable power supply	$5	Big Lots (801) 596-8611

Conclusion

We hope this project will give us a better understanding of motes and how to see a project through to its completion. We hope to be able to investigate nd pioneer a new technique for handling traffic. Currently stop lights are soley based on timing, and each light is independent of the rest Our method would create an interdependent network that can dynamically adjust for unexpected turns. Instead of getting to the light for it to realize you're there (weight sensor) the light would expect you coming and, in future versions, change the light you, before you get there. No more rolling out the red carpet, but rolling out the green carpet of green lights.

http://www.willow.co.uk/html/telosb_mote_platform.html
http://www.willow.co.uk/TelosB_Datasheet.pdf
http://www.tinyos.net/scoop/
http://www.dcg.ethz.ch/~rschuler/img/roundtrip/play.htm
http://www.intel.com/research/sensornets/
http://www.xbow.com/Home/HomePage.aspx
http://computer.howstuffworks.com/mote.htm
http://www.eng.utah.edu/~alhamdan/Mote/ref.pdf
http://en.wikipedia.org/wiki/Dipole_antenna
http://www.radioshack.com/product/index.jsp?productId=2062691
http://maps.google.com/
http://groups.google.com/
http://documents.google.com/
http://toilers.mines.edu/Public/XubunTOS

Appendix

The Blinking LED example

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

StdControl.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

BlinkM.nc

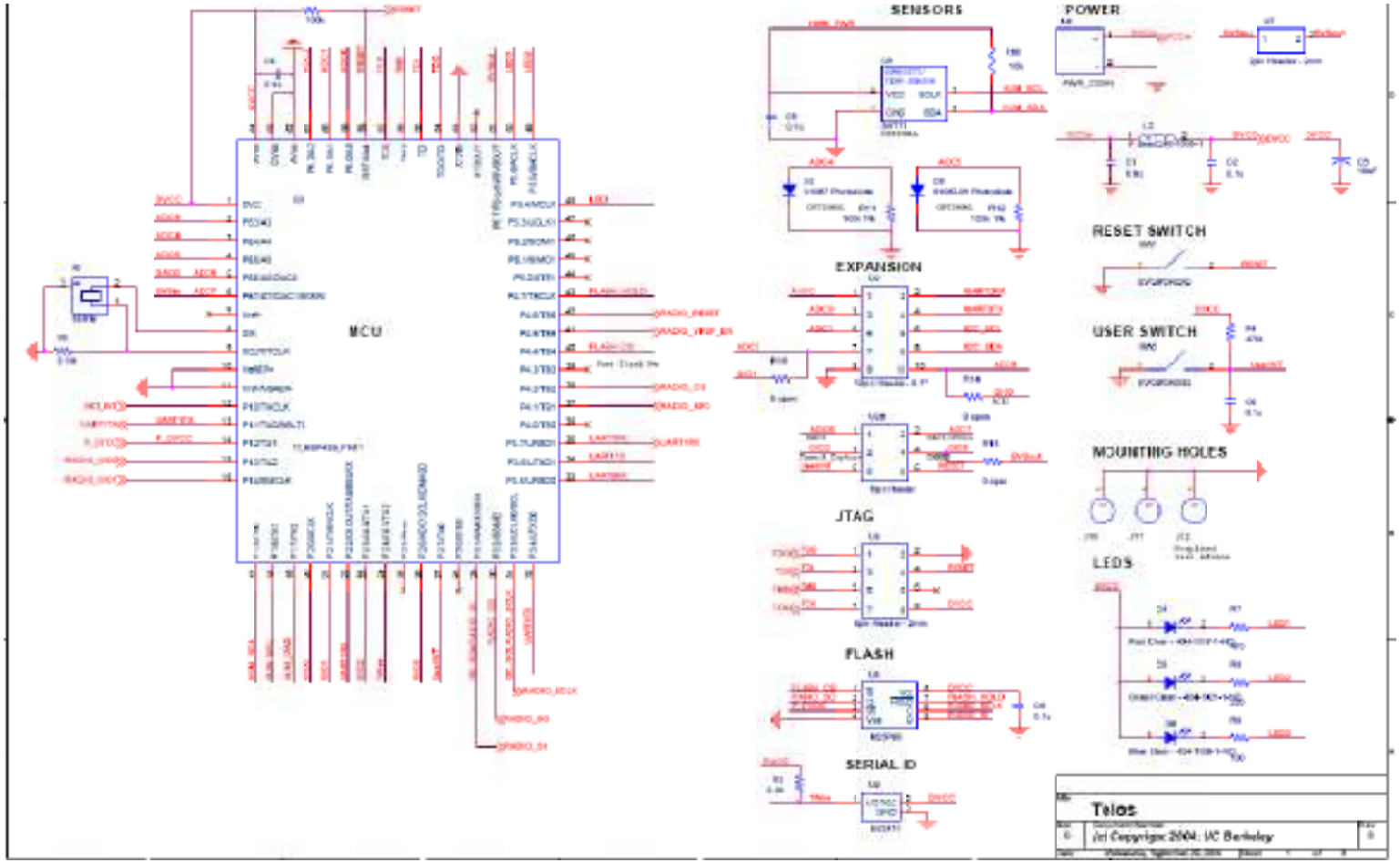
```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
// Continued below...
```

Timer.nc

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
}
```

BlinkM.nc, continued

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```


SENSORS
POWER
MCU
EXPANSION
RESET SWITCH
USER SWITCH
MOUNTING HOLES
JTAG
LEDS
FLASH
SERIAL ID
Telos
(c) Copyright 2004: UC Berkeley