Dan Weinberg,
Jie Zhang

16.671 -Advanced Computer Architecture
Project #4 – Sensor Wireless Network
Final Report

Introduction

Sensor networks are becoming a reality in different applications, such as medical, commercial, industry, and so on. Low power consumption sensor networks can be used for months to years without changing the battery, witch brings significant convenience to users. High data rate makes the data collection more fast and the networks work more efficiently.

In this project we built a low power consumption high data rate wireless sensor network using Moteiv's Tmote wireless sensing system, which include a Tmote Sky module and integrated light, temperature, sounds…etc. sensors. The wireless sensor network was comprised of two basic nodes. One or more SENSOR nodes, and a single DISPLAY node that was connected to the PC as a base station to collects and format data from other nodes. The SENSOR nodes were put in different places to collect the temperature and communicate with the base node via radio in an ad-hoc network. The PC provides GUI, display and control.

Tmote Sky

Tmote Sky is a mote platform for extremely low power, high data-rate sensor network applications. It has integrated sensors, radio, antenna, microcontroller and programming capabilities.

The low power operations of the Tmote Sky module is due to the ultra low power TI MSP430 F1611 microcontroller. This 16-bit RISC processor features extremely low active and sleep current consumption. In order to minimize power consumption, it is in sleep mode during majority of the time, wake up as fast as possible to process, then return to sleep mode again.

It uses a USB controller from FTDI to communicate with the host computer and features the Chipcon CC2420 radio, which is an IEEE 802.15.4 compliant radio providing reliable

wireless communication, for wireless communications. The radio provides fast data rate and robust signal. It is controlled by the microcontroller through the SPI port and can be shut off for low power duty cycled operation.

The internal Inverted-F microstrip antenna is a pseudo omni directional antenna that may attain 50-meter range indoors and up to 125-meter range outdoors.

The key feature of the Tmote sky is listed below:

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- Interoperability with other IEEE 802.15.4 devices
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption
- Fast wakeup from sleep (<6μs)
- Hardware link-layer encryption and authentication
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector
- TinyOS support : mesh networking and communication implementation
- Complies with FCC Part 15 and Industry Canada regulations

Tmote Sky Temperature Sensor

The humidity/temperature sensor is manufactured by Sensiron AG, produced using a CMOS process and coupled with a 14-bit A/D converter. The calibration coefficients are stored in the sensor's onboard EEPROM. The temperature accuracy is 0.5°C.

TinyOS

TinyOS was used as the development environment in this is project, which is an event-driven operating system intended for sensor networks with limited resources. The TinyOs development environment directly supports a variety of device programmers and permits programming each device with a unique address attribute without having to compile the source code each time. The TinyOS system, libraries and applications are

written in nesC, a Version of C that was designed for programming embedded systems. In nesC, programs are built out of components that are wired together to form whole program. The components are linked together by their interfaces. These interfaces are bidirectional and specify a set of functions to be implemented by their providers and users. NesC expects the code to be generated by whole program compilers and is based on run-to-completion tasks and interrupt handlers that may interrupt tasks and each other.

Mote/PC Communication:

We established communication between the Mote and the PC over the COM port using a modified application apps/TOS_Base. It was modified to run on the base mote to echo UART messages to the radio and radio messages to the UART and maintain UART and radio message buffers.

The java application net.tinyos.sf.SerialForwarder is a java-based applet that creates a TCP socket to allow data to be shared with other PC applications. It takes the incoming messages from the UART, and forwards it over an internet connection so that PC based applications can use the data. SerialForwarder also works in another direction. It forwards the request packet to the UART, and the attached mote forwards the message over the radio.

Display Readings on PC

A java tool modified based on the Oscilloscope tool come with TinyOS as $TOSDIR/tools/java/net/tinyos/oscope/Oscilloscope was used to display the readings on PC. It was modified for our project to include custom controls and display panes. It is responsible for sending mote commands via USB link with base node, graphing incoming sensor data and displaying current temperature readings along with mote status information

Figure below shows the set up of the communication:

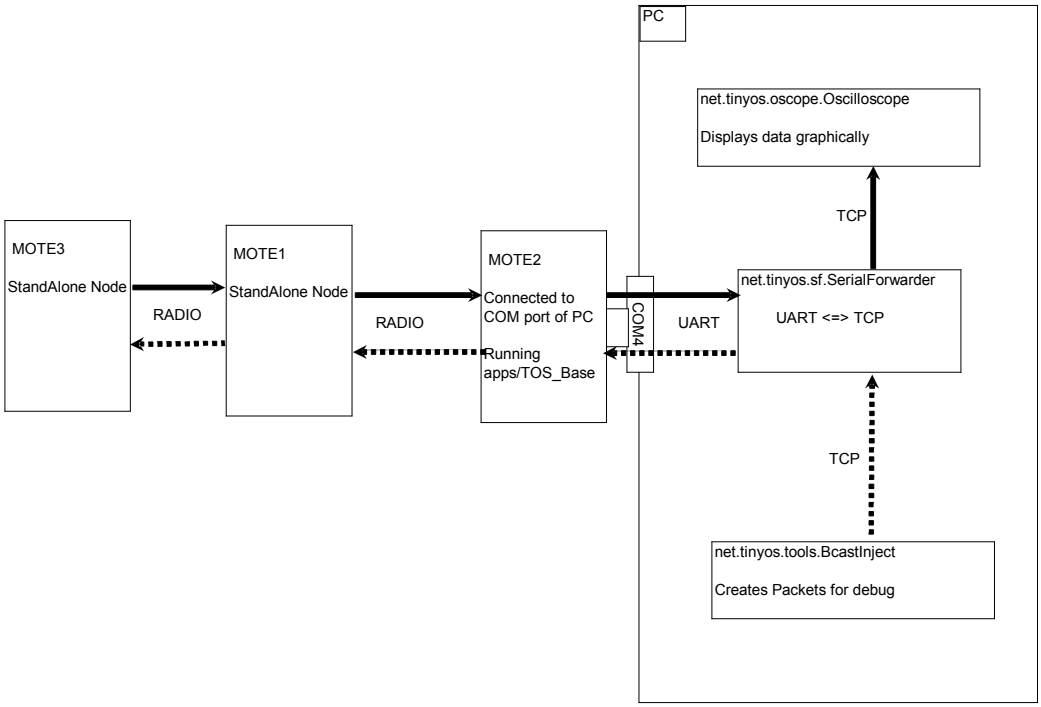

Figure 1. Set up of communications

Networking scheme

In a sensor network, some nodes are too far from the sink. The information being transferred has to travel through other nodes to reach its final destination. Therefore, multi-hop wireless networks were used in our project. The nodes of these networks can collaborate with their neighbors to perform functions including routing, localization, etc.

There are several routing algorithm used for multi-hop wireless networks. One of the algorithms is called “flooding”, in which the source node sends the information to all of its neighbors, no matter where the destination is. Each of the nodes that received this information sends the same packet out to each of its neighbor again. This is repeated until the information packet has reached the destination. This algorithm is simple but it generates a lot of inefficient network behavior.

Shortest path algorithm

The algorithm we used is the shortest path routing algorithm, which eliminates the inefficient network behavior of the previous one by directing a packet down one single path.

The goal of shortest path algorithm is to guide the packets from one node to its destination through the least amount of hops possible in a wireless network. In this algorithm, a list was used to store the shortest path of sending the message back to the base node. The path was built by adding the nodes' number into the table when the message passes a node. A hop count called "Number of issues" was also incremented as well. A minimum path length was saved in the remote motes. When a remote mote get a request message it compares its minimum path length with the hop count in order to find the shortest path for its data to get back.

The detail of the scheme is described below:

Each mote on powerup will listen for a PATHCAL message from the base. A PATHCAL message can only originate from the base, but can be repeated by other motes. A PATHCAL message will be issued by the base at regular intervals.
PATHCAL Message format:

Byte	Data
0	PATHCAL code
1	Number of issues (N)
2->N+1	ID of all motes, in order that repeated the message

Ex. Base mote (ID = 255) issues:
{PATHCAL,1,255}

If node 6 repeats this message, it adds its own ID and repeats as :
{PATHCAL,2,255,6}

If node 3 repeats this message, it adds its own ID and repeats as :

{PATHCAL,3,255,6,3}

Etc.

This way, any mote receiving the message can see an established path back to the base, by looking at the ID string.

A mote keeps the following data:

Byte Inbound_Mote_ID;

Byte Min_path_length;

When a mote receives the PATHCAL message compares the number of issues to its own Min_Path_Length. If the number is less than its current Min_Path_Length, then this path is a shorter path to the base. In this case, the mote will

1) store the new Min_Path_Length
2) Read the last ID from the string, and set that ID as the new Inbound_Mote_ID
3) Append its own ID onto the string, then Rebroadcast the message

If the number of issues is equal to the Min_Path_Length, and the last ID in the string is equal to the current Inbound_Mote_ID, then this message is a confirmation of an existing path. The mote will not change its own internal data, but will append its own ID and Rebroadcast the message. This will allow the message to propegate to any new motes that may be in range.

Messages:

If a mote receives a PATHCAL message, it will act as above. If a mote receives a DATA message, it will repeat the message addressed to its Inbound_Mote_ID.
A mote will send its own DATA messages addressed to its Inbound_Mote_ID, which is defaulted to the address of base mote on power on. While the value of base mote's address is present, the mote will only wait for a PATHCAL message. Once a mote gets a valid Inbound_Mote_ID from a PATHCAL message, it will begin to transmit its temprature at regular intervals.

Example Below:

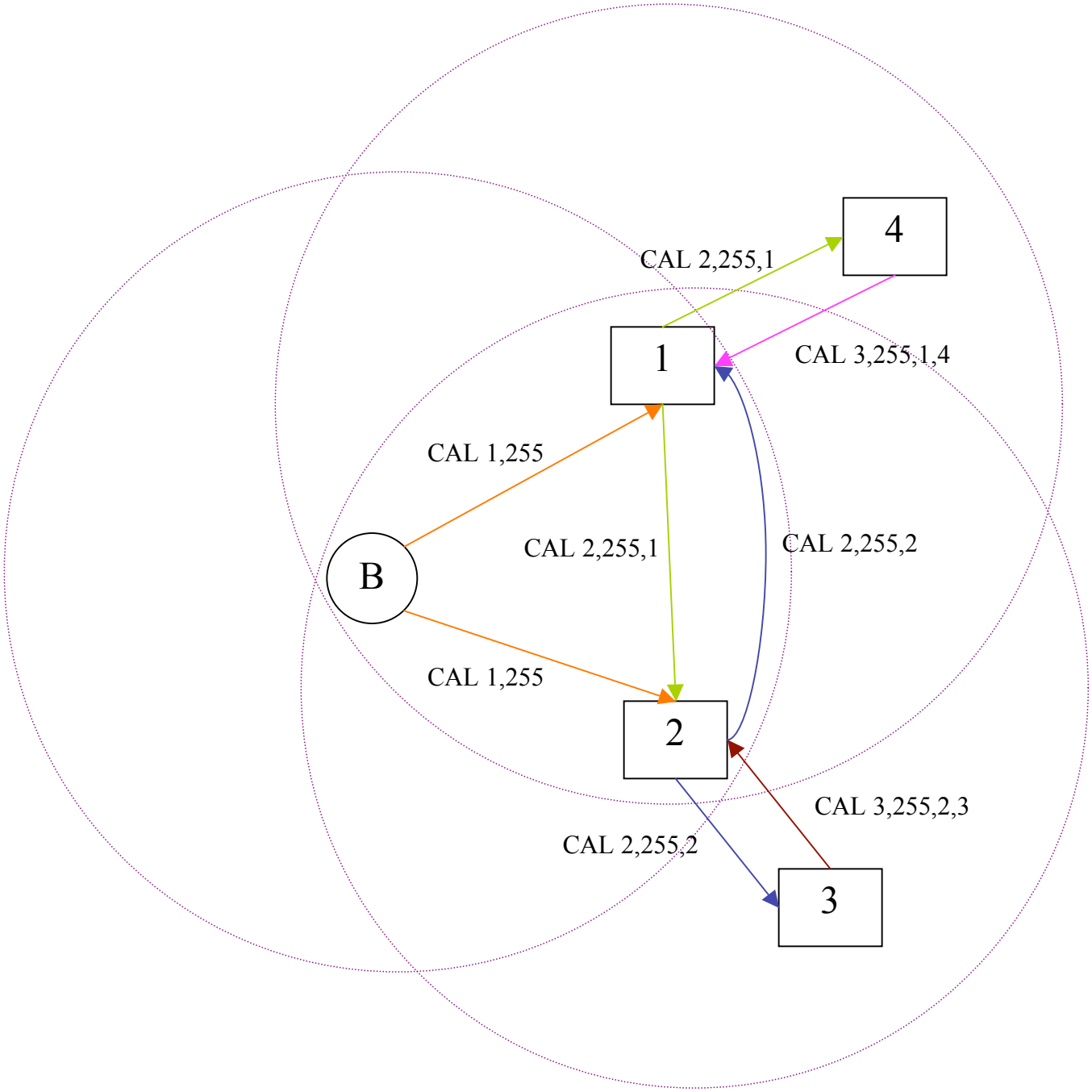

Figure 2. Example of networking scheme.

In the above example, Motes 1 and 2 are within the broadcast range of the Base (Mote 255), and each other. Mote 3 is in range only of Mote 2, and Mote 4 is in range only of Mote 1.

The base issues the PATHCAL message { CAL,1,255 }.

Motes 1 and 2 receive the message with number = 1, which is less than the default value (99) of Min_Path_Length, so motes 1 and 2 set their own Min_Path_Len to 1, and assign 255 as their Inbound_Mote_ID. Motes 1 and 2 also add their own IDs to the message and rebroadcast messages with number =2. Motes 1 and 2 will receive each other's rebroadcast, but the number (2) is greater than their current Min_Path_Length (1) so no action is taken.

Motes 3 and 4 also receive the rebroadcast from mote 2 and 1, with number = 2. This is less than the default(99) so Motes 3 and 4 set their Min_Path_Length to 2, and set their Inbound_Mote_ID to 2 and 1, respectively. Motes 3 and 4 also add their own IDs to the message and rebroadcast messages with number =3.

Motes 2 and 1 will each receive rebroadcasts from Motes 3 and 4, but the number (3) is greater than their current Min_Path_Length (1) so no action is taken.

Example below shows when more motes are added:

At some time later, Mote 5 comes on line, within range of only Mote 3. Soon, another PATHCAL command is issued by the base. Mote 3 receives a rebroadcast from Mote 2 {CAL,2,255,2} with number = 2. This is equal to Min_Path_Length, and the last ID (2) is the Inbound_Mote_ID. Therefore, Mote 3 appends and rebroadcasts the message, where it is received by Mote 5. Number (3) is less than the default (99) so Mote 5 sets Inbound_Mote_ID to 3, and Min_Path_Length to 3. No other action is taken by the network.

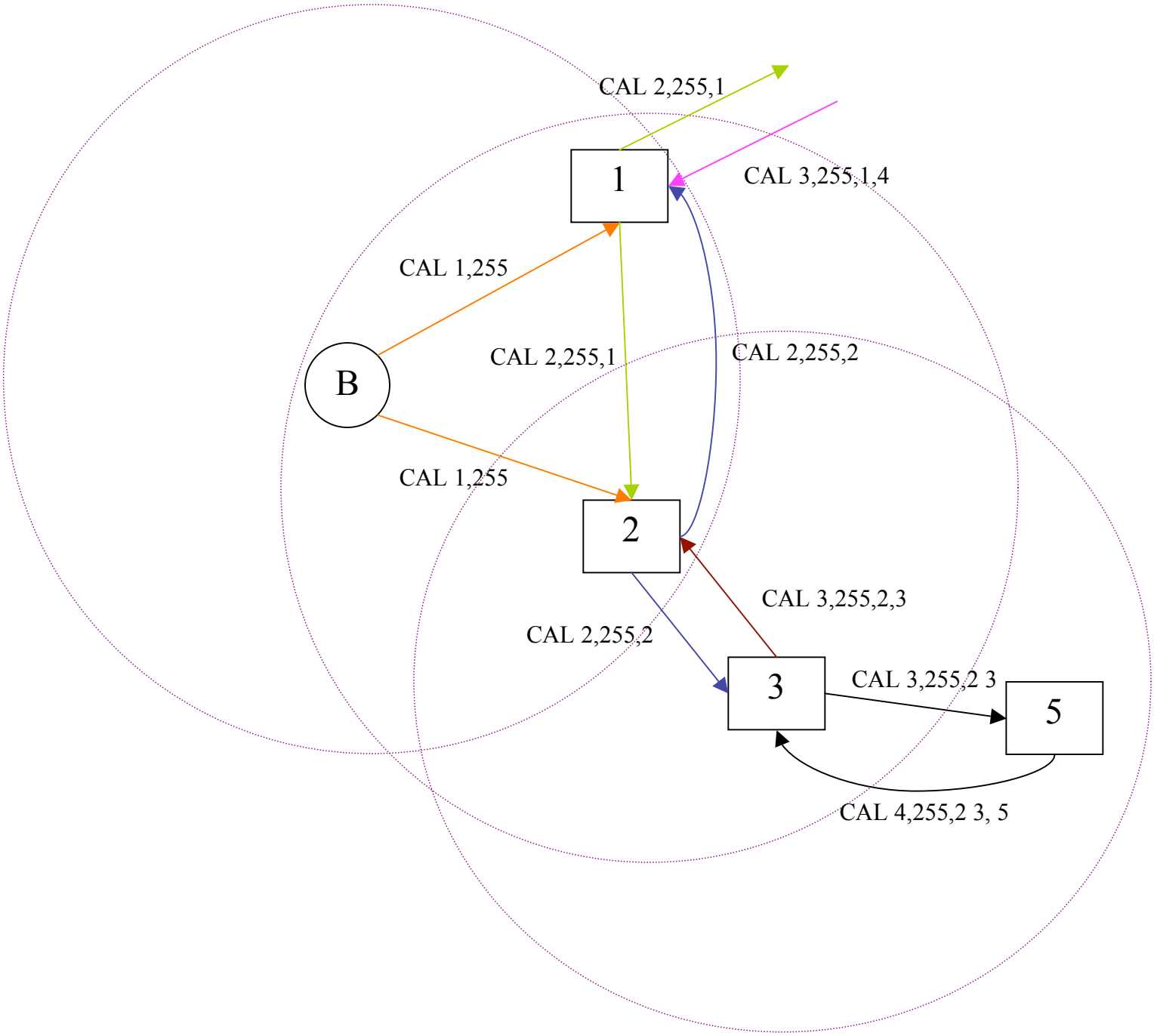

Figure 3. Example 2 of networking scheme.

Programming

There are two kinds of motes in our project. One is remote nodes, the other is base node.

Remote mote

The functions of program for remote motes include Reading temperature sensor and transmitting data, Relaying data packet from other remote nodes and path message from base node, maintaining mote ID of inbound contact that provides the shortest path to the base node. The remote nodes can also change their pathlength and contact ID when some of them come online or offline. A node resets its network connection when it loses the connection with base note for a specific time. When this reset occurs, the remote node can connect through another possible path.

The configuration of the application for remote nodes is below:

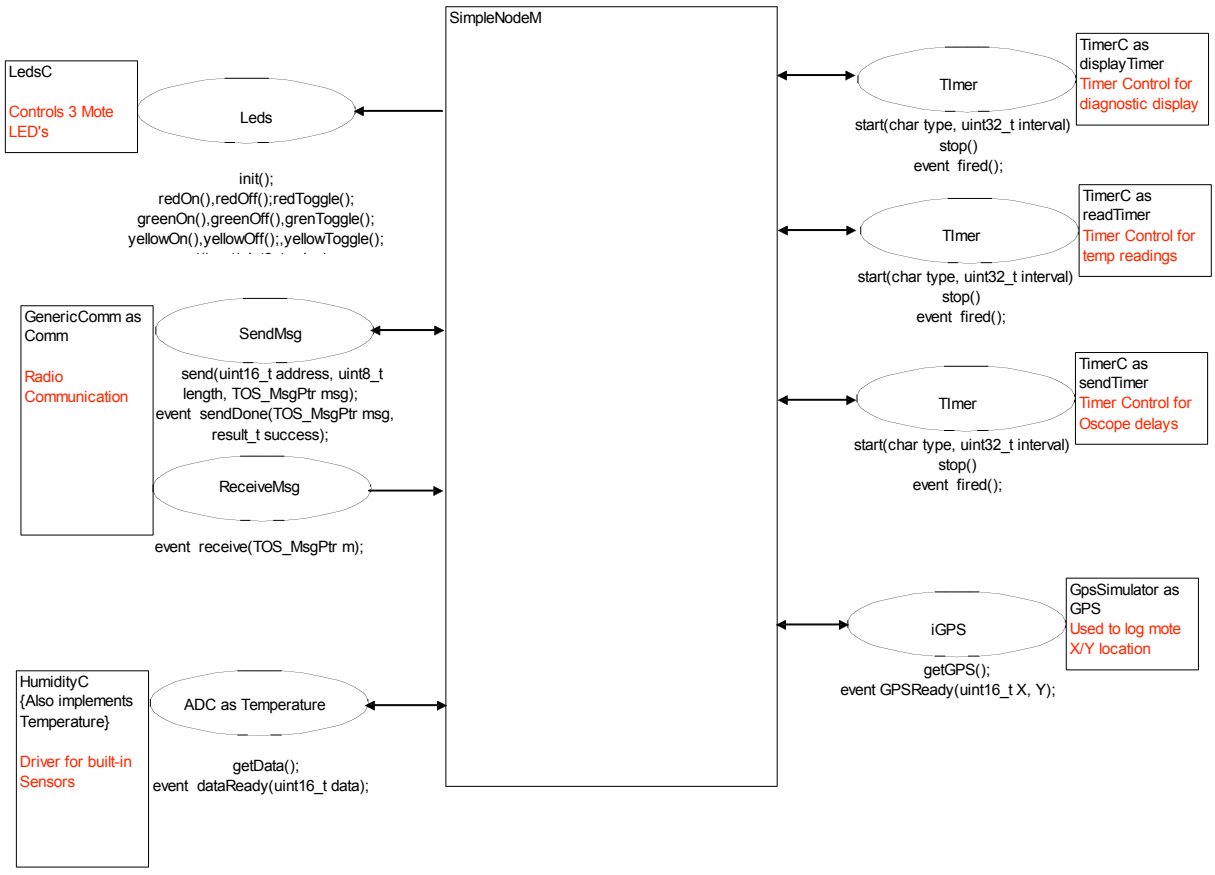

Figure 4. Module and interfaces for remote nodes.

Base node

The base node is responsible for transferring between USB message and radio message and initiating PATHCAL message to allow the remote nodes to map the shortest path. It provides interface between PC and display software.

The modules and interfaces used by the base node are shown in figure below:

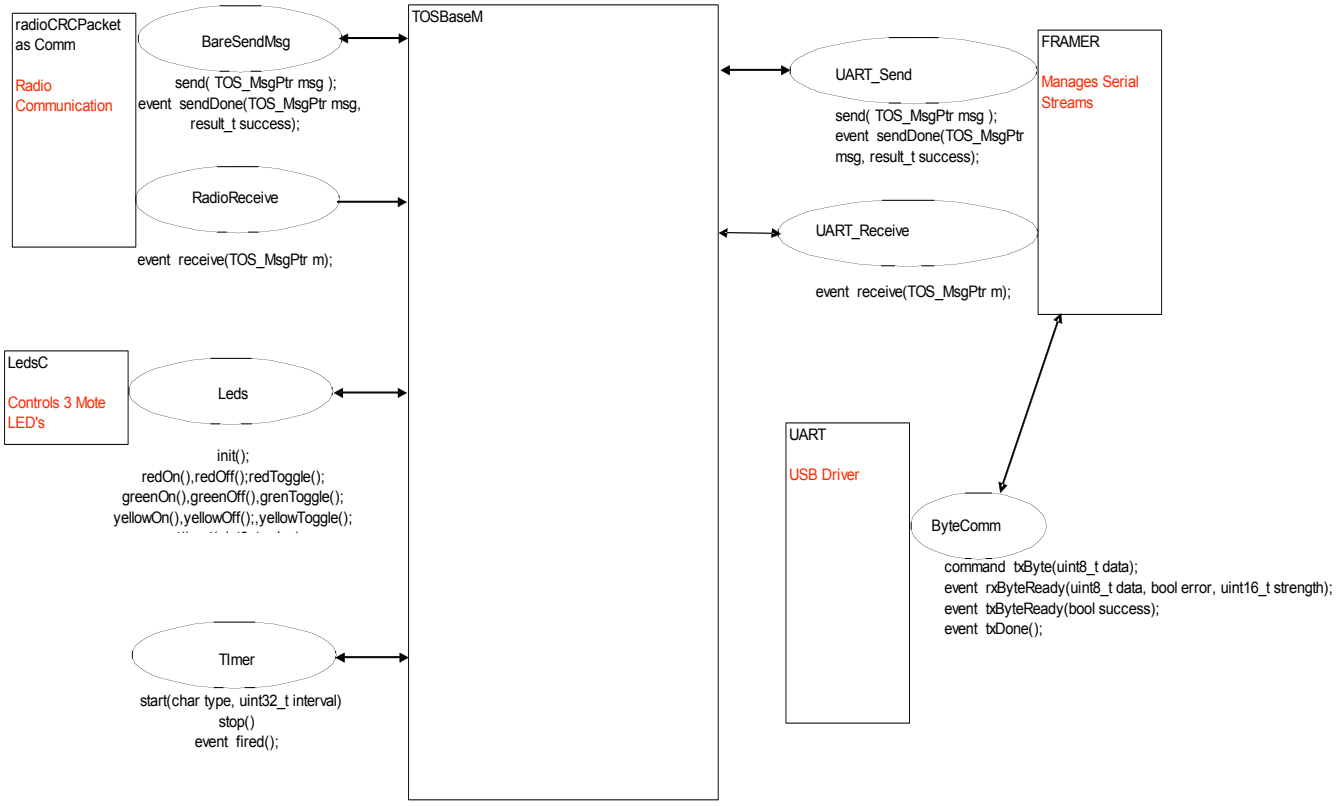

Figure 5. Modules and interfaces for base node

Oscilloscope

We also modified the java tool oscope provided with TinyOS sample application section. It sends mote commands via USB link with base node and graphs incoming data. Additional controls and display features are added such as start/stop command and last read values.

Result

We tested our design with six motes. One was programmed with TOSBase, acting as a base node and the other five were programmed with remote node application. Then we started serial forward and opened the oscilloscope. After the start command was sent, we saw that the motes started forwarding the temperature readings to the base node. We also checked the path by looking at the raw data packets. It all worked as we expected.

Conclusion

In this project, we designed a wireless sensor network with Tmote sky sensing system. We developed our own shortest path network routing algorithm and modified the java tool for custom display. We first built a simple network between two motes. Based on the two-mote wireless network, we studied the applications came with the TinyOS system. Then we selected proper tools to forward the packets over internet connection and display and simulation. Later we designed the network scheme for multiple motes communicate in a meshed network. We tested our design using real motes and simulation software. The result shows that our design works as we expected.