Wireless Sensor Network for Substation Monitoring

by

Siddharth Kamath

March 03, 2010
Need for Substation Monitoring

- Monitoring health of Electrical equipments
- Detecting faults in critical equipments.
  - Example: Oil cooled circuit breaker, transformers, bushings
- Avoiding power outages
Why Wireless Sensor Networks?

- Elimination of “Cabling” or “Wiring”
- Eliminates need for manual data collection
- Can be rapidly deployed
- Modern sensor networks are low cost
- Sensor networks are scalable
Challenges and Objectives

- Low cost and low power sensors
- Accurate signal processing ability
- Reliability - Providing correct data
- Adaptation to increasing network size
- Long operational time (Long battery/sensor life)
- Network should be self organizing and self healing
Selection of Wireless Sensor Platform

Crossbow’s MICAZ wireless sensor nodes
- Atmel’s ATmega128L processor
- 2.4 GHz Chipcon CC2420 radio
- 128KB PM, 512KB Flash, 4KB EEPROM
- Programming - TinyOS

MTS300/MTS310 sensor board
- Built in light, temperature and acoustic sensors
- Dual axis accelerometer, magnetometer

MIB510 Serial Gateway
- Functions as a base station
- Allows PC to collect data from each mote
- RS-232 serial programming interface

Figure 1: Crossbow products used [2]: (a) MICAz mote, (b) MTS300 basic integrated sensor board and (c) MIB510 interface board
A Wireless Node

• The wireless mote along with sensor board and battery packed inside a weather proof case
Deployment of Nodes

- Nodes deployed on transformer circuit breakers (temperature sensor)
- Nodes attached on oil-filled transformers (vibration sensors)
Figure 4: Locations of deployed nodes in TVA's Paradise sub-station.
The Network Base Station

- Crossbow’s “Stargate” node programmed as the BS (Server)
- Simultaneous data acquisition
- 266MHz Intel IXP420 XScale processor
- Ethernet port and two USB ports
- Uses a Linux OS
- Has internet connectivity (To transmit data to a remote PC)
- Uses generic shell scripts
  - Example: data_collect, send_data, archive_data, send_ip etc.

Fig 5. Crossbow [2] Stargate Node
Selection of a Networking Protocol

- Crossbow’s X Mesh routing protocol – Multi-hop communication
- X Mesh
  - Link quality based dynamic routing protocol
  - Periodic route update (RU) messages – 15 minute interval
  - Self organizing mesh network
  - Self healing – Route table updated in case of node failure
  - Protocol supports scalability
  - Bidirectional communication between nodes and base station
Power Modes in XMesh

- **XMesh High Power Mode**
  - Mote radios are always ON
  - High bandwidth, low latency
  - 15 sec RUI

- **XMesh Low Power Mode**
  - Mote radios periodically wake up
  - Low bandwidth, high latency
  - 150 sec RUI

- **XMesh Extended Low Power Mode**
  - Nodes cannot route data
  - Used only for “end nodes” in a network
  - 360 sec RUI
XMesh Features and Benefits

Health Diagnostics
- Nodes transmit health packets to base station
- Radio traffic, battery voltage, parent RSSI
- Data collected using the “Xsniffer” application

Time Synchronization
- Network global time synchronization to ± 1 msec
- Synchronize radio and sensor measurements

Over-the-Air-Programming (OTAP)
- Reprogram node/nodes over the air with new code
- Downstream strategy – Allows images of code to be sent to different motes
XMesh Network Setup

• Step 1: Initial transmissions from motes are broadcast messages (Bcast application)
• Step 2: Base station (BS) transmits a route update message
  • An indication that BS can hear the motes
• Step 3: Motes transmit a route update message
  • An indication to the BS that the mote can hear it
• Step 4: Mote joins the mesh network and directs all its messages to the base station
Programming the Motes

- **Platform:** TinyOS using nesC
- **TinyOS** – An open-source operating system designed for wireless embedded sensors.
- **TinyOS** has its own component library which includes:
  - Network protocols
  - Distributed services
  - Sensor drivers
  - Data acquisition tools
- **Motes programmed through the MIB510 board using the serial or USB port**
Code snippet (Configuration file)

Mote programmed as a temperature sensor

/* File : Temp.nc */ /* Copyright (c) 2000-2003 The Regents of the University of California. */

includes sensorboard;
configuration Temp {
  provides interface ADC as Temp;
  provides interface StdControl;
}
implementation {
  components Main, SenseM, LedsC, TimerC, temp;

  Main.StdControl -> SenseM;
  Main.StdControl -> TimerC;

  SenseM.ADC -> Temp;
  SenseM.Leds -> LedsC;
  SenseM.Timer -> TimerC.Timer[unique("Timer")];
}
Code snippet (Module file)

/* File : TempM.nc */
/* Copyright (c) 2000-2003 The Regents of the University of California. */

implementation {

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 500);
}

event result_t Timer.fired() { //read sensor data in response to the timer fired event
    return call ADC.getData(); // return the result of ADC
}

async event result_t ADC.dataReady(uint16_t data) {
    display(7-((data>>7) &0x7)); // display the bits on to LED’s
    return SUCCESS;
}
}
Observation and Monitoring Results

Fig 7.1 Vibration sensor waveform [5]

Fig 7.2 Node 1 as temperature sensor - waveform [5]

Fig 7.3 Acoustic sensor Vs Date [5]

Fig 7.4 vibration intensity Vs Date [5]
Challenges and Future Work

- Energy usage – Directly proportional to number of nodes
- Communication reliability – Larger the network size, more the hop count
- Design a networking protocol for load balanced routing
- Energy efficient nodes harvesting solar power
Fig 8. Node Locations
References:


