Wireless Sensor Networks: Background and Introduction

Anandha Gopalan

01/12/08

Outline

- Lecture 1
  - Basic background and Introduction
- Lecture 2
  - Where are we now ? – A look at some current research
- Lecture 3
  - Further discussion about current research and a look at future direction and challenges

Lecture 1: Outline

- Basic background
- A deeper look
  - Sensor node hardware
    - Example devices
  - Sensor node software
    - Operating system
    - Programming language
- Research challenges
- Conclusion

Overview

- What are wireless sensor networks (WSNs) ?
  - A WSN is a network consisting of a large number of low-cost, low-power multi-functional sensor nodes
    - Densely deployed to monitor a specific state of the environment
    - Sensor node has a wireless link, an on-board processor for basic computation and a sensor

Goals

- Provide a link between the physical world and data networks
- An ability to
  - Access sensors deeply embedded in the environment
  - Observe previously unobservable environmental states

Characteristics

- Distributed
- Quickly deployable and disposable
- Scalable, rapidly configurable
- Coordinated and synchronized
- Affordable and cost-effective

WSNs vs. Ad hoc networks

WSNs	Ad hoc Networks
Hundreds of thousands of nodes	Tens to hundreds of nodes
Densely deployed	Not densely deployed
Limited with respect to power, cpu and memory	Limited with respect to the capability of the machine being used
Generally stationary after deployment	Nodes are usually fully mobile
Deployed for a specific reason	Networks are created on the fly

Can optimise by choosing the correct set of sensors to use

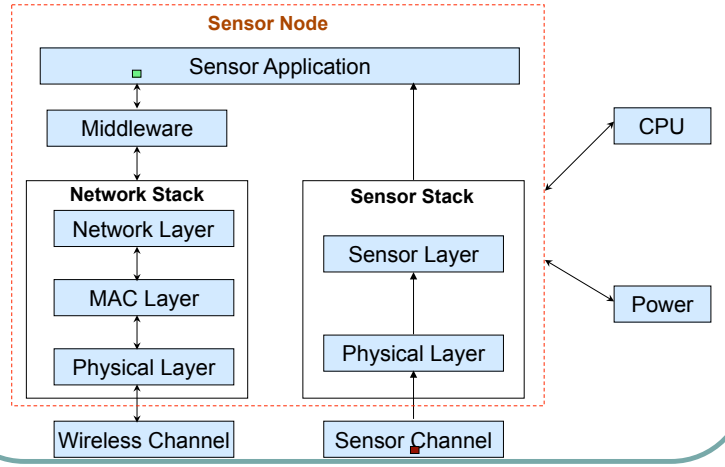
Applications

- Home automation (smart homes)
- Environmental monitoring (air, water, surveillance)
- Habitat monitoring
- Seismic monitoring
- Military applications
- Traffic control
- Emergency response
- Building climate control

Lecture 1: Outline

- Basic background
- A deeper look
  - Sensor node hardware
    - Example devices
  - Sensor node software
    - Operating system
    - Programming language
- Research challenges
- Conclusion

Sensor Node Components



WSNs - Available Platforms

- Wireless Network
  - Zigbee
  - WirelessHART
    - Extension of the HART protocol suite
  - 6lowpan
  - ISA100
    - New standard under development, to be finished in 2009
  - All of the above are based on the IEEE 802.15.4 standard

IEEE 802.15.4 Standard

- 16 channels in the 2450 MHz band, 10 channels in the 915 MHz band, and 1 channel in the 868 MHz band
- Over-the-air data rates of 250 kbps, 40kbps and 20 kbps
- Allocated 16 bit short or 64 bit extended addresses
- CSMA/CA channel access
- Energy detection and link quality indication

Courtesy: Bhaskar Krishnamachari, University of Southern California

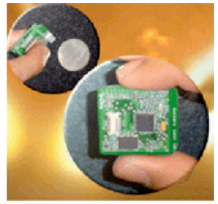
WSNs - Available Platforms

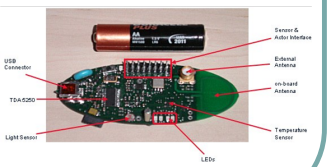
- Sensors
  - Accelerometer, temperature, light, infra red, movement, proximity, …
- Software
  - Operating Systems
    - TinyOS, Contiki, MANTIS, SOS, Nano-RK, …
  - Programming Languages
    - nesC, SNACK, DCL, Java “Sentilla”, …
  - Simulators
    - TOSSSIM, J-Sim, …

Focus

WSNs - Available Devices

- Body Sensor Node (Imperial)
  - TI MSP430 ultra low power processor
  - Zigbee
  - Range 50m
  - TinyOS
- eyesIFXv2 (TU Berlin)
  - TI MSP430
  - TinyOS
  - Integrated temperature and light sensors

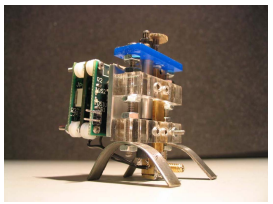


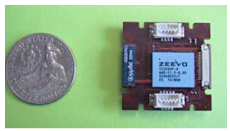


Courtesy: http://ubimon.doc.ic.ac.uk/bsn/index.php?m=206

WSNs - Available Devices

- Parasitic node (MIT)
  - Silicon Labs C8051F311 microcontroller
  - 2 axis accelerometer, microphone, active IR proximity sensor, temperature, light sensor …
  - Motorola FS OnCOre single chip GPS module
- iMote (Intel)
  - ARM core
  - Bluetooth
  - Zeevo module





Courtesy: http://ubimon.doc.ic.ac.uk/bsn/index.php?m=206

TinyOS

- Event based operating environment designed for use with embedded networked sensors
- Open source
- Programming language is *nesC*
- http://www.tinyos.net

TinyOS

- Characteristics
  - Small footprint
  - Low system overhead
  - Low power consumption
- Only one process at a time
- Single linear address space
- No dynamic memory allocation

Lecture 1: Outline

- Basic background
- A deeper look
  - Sensor node hardware
    - Example devices
  - Sensor node software
    - Operating system
    - Programming language
- Research challenges
- Conclusion

TinyOS Scheduler

- Scheduler is simple FIFO
- Two-level scheduling
  - Tasks
    - Time flexible
    - Typically commands
    - Atomic with respect to other tasks (single threaded)
    - Pre-empted by events
  - Events
    - Time critical
    - Shorter duration (hand off to task if need be)
    - Interrupts task (higher priority)
    - Last-in first-out semantics (no priority among events)

```
main {
    …
    while (1) {
            while (more tasks)
              schedule_task;
            sleep;
    }
}
```


TinyOS Memory Model

- Static memory allocation
  - No heap
  - No function pointers ☺
  - Size required is determined at compile time
- Global variables
  - Conserve memory
  - Use pointers
- Local variables
  - On the stack

TinyOS Communication Model

- Uses Active Messaging (AM)
  - Light weight architecture
  - Each Active Message contains
    - User-level handler to be invoked on arrival
    - Data payload passed as argument
  - Event-centric nature
    - Enables network communication to overlap with sensor-interaction
  - Handler functions
    - Extract message quickly from network
    - Provide data for computation/forward data
    - Prevent network congestion

TinyOS Communication Model

- AM Component
  - Accepts commands from application
  - Fires events to message handlers
  - Event to signal completion of transmission
  - Send command includes
    - Destination Address, Handler ID, Message body
  - Address checking and dispatching
  - Relies on components for packet transmission
- Radio Packet
  - 30 Byte fixed length packet
  - 16-bit CRC check

nesC

- Programming language for networked embedded systems
- Extends a subset of C
- “Static” language
  - No dynamic memory allocation
  - Call-graph is fully known at compile-time
- Supports and reflects TinyOS’s design
- nesC applications are built out of *components* with well-defined, bidirectional *interfaces*

nesC Components

- nesC applications are built by writing and assembling components
- Two types: *modules* and *configurations*
  - Modules
    - Provide application code, implementing one or more interfaces
  - Configurations
    - Used to wire other components together, connecting interfaces used by components to interfaces provided by others.
    - Every nesC application is described by a *top-level configuration* that wires together the components used
- Contains frame (internal state), functions (implementation of commands, events and tasks)
- Provides and uses *interfaces*

nesC

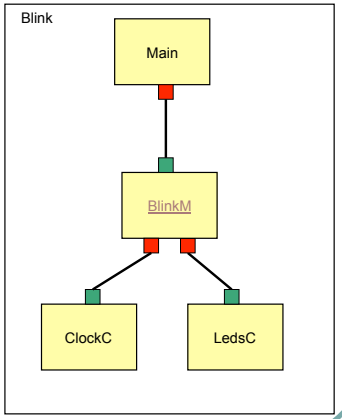
- Interfaces
  - Are bi-directional
  - Define interaction boundary between components
  - Declare a set of functions
    - Commands (call down): implemented by provider
    - Events (call up): implemented by user
  - A single component may use of provide multiple interfaces
    - Can also have multiple instances of the same interface
  - For a component to call a command in an interface, it must implement the events of that interface

nesC Example

- Blink application

```
configuration Blink {
}

implementation {
  components Main, BlinkM, ClockC, LedsC;

  Main.StdControl->BlinkM.StdControl;
  BlinkM.Clock->ClockC;
  BlinkM.Leds->LedsC;
}
```

Blink.nc



Courtesy: Radu Stoleru, University of Virginia

Example

- BlinkM module:

```
module BlinkM {
  provides interface StdControl;
  uses interface Clock;
  uses interface Leds;
}

implementation {
  bool state;

  command result_t StdControl.init() {
    state = FALSE;
    call Leds.init();
    return SUCCESS;
  }
```

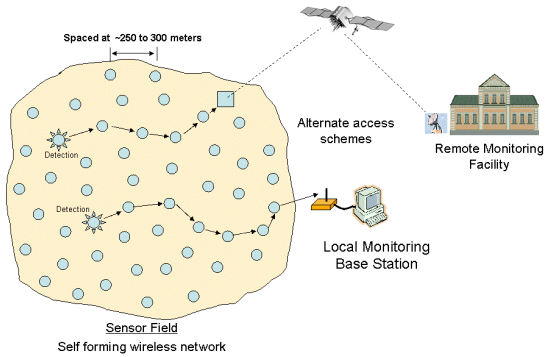
Blink.nc

```
  command result_t StdControl.start() {
    return call Clock.setRate(128, 6);
  }

  command result_t StdControl.stop() {
    return call Clock.setRate(0, 0);

  event result_t Clock.fire() {
    state = !state;
    if (state) call Leds.redOn();
    else call Leds.redOff();
  }
}
```

Blink.nc

Courtesy: Radu Stoleru, University of Virginia

A Typical Wireless Sensor Network



Courtesy: http://www.alicosystems.com/

Lecture 1: Outline

- Basic background
- A deeper look
  - Sensor node hardware
    - Example devices
  - Sensor node software
    - Operating system
    - Programming language
- Research challenges
- Conclusion

Research Challenges

- Data Storage
- Data Dissemination
- Power Management
- Fault tolerance
- Scalability
- Network Topology
- Security

Conclusion

- Wireless sensor networks are an emerging technology with enormous potential
- As well as being implemented and used in various areas, they have found a niche in research as well
- Some sample of current research will be covered in the next lecture

☺ Questions ☺