Last lecture: Why use WSNs

Ease of deployment: Wireless communication means no need for a communication infrastructure setup

Low-cost of deployment: Nodes are built using off-the-shelf cheap components

Fine grain monitoring: Feasible to deploy nodes densely for fine grain monitoring
HARDWARE PLATFORMS
3 broad types of nodes

Grain sized: RFID, smart dust

Matchbox sized: Berkeley motes and derivatives

Brick sized: Stargates (potentially using wall power)
<table>
<thead>
<tr>
<th>Node</th>
<th>CPU</th>
<th>Power</th>
<th>Memory</th>
<th>I/O and Sensors</th>
<th>Radio</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spec 2003</td>
<td>4–8Mhz Custom 8-bit</td>
<td>3mW peak 3uW idle</td>
<td>3K RAM</td>
<td>I/O Pads on chip, ADC</td>
<td>50–100Kbps</td>
<td>Full custom silicon, traded RF range and accuracy for low-power operation.</td>
</tr>
<tr>
<td>Rene 1999</td>
<td>ATMEL 8535</td>
<td>.036mW sleep 60mW active</td>
<td>512B RAM 8K Flash</td>
<td>Large expansion connector</td>
<td>10Kbps</td>
<td>Primary TinyOS development platform.</td>
</tr>
<tr>
<td>Mica-2 2001</td>
<td>ATMega 128</td>
<td>.036mW sleep 60mW active</td>
<td>4K RAM 128K Flash</td>
<td>Large expansion connector</td>
<td>76Kbps</td>
<td>Primary TinyOS development platform.</td>
</tr>
<tr>
<td>Telos 2004</td>
<td>Motorola HCS08</td>
<td>.001mW sleep 32mW active</td>
<td>4K RAM</td>
<td>USB and Ethernet</td>
<td>250Kbps</td>
<td>Supports IEEE 802.15.4 standard. Allows higher-layer Zigbee standard.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>CPU</th>
<th>Power</th>
<th>Memory</th>
<th>I/O and Sensors</th>
<th>Radio</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-bandwidth Sensor Nodes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imote 1.0 2003</td>
<td>ARM 7TDMI 12-48MHz</td>
<td>1mW idle 120mW active</td>
<td>64KB SRAM 512KB Flash</td>
<td>UART, USB, GPIO, I²C, SPI</td>
<td>Bluetooth 1.1</td>
<td>Multihop using scatternets, easy connections to PDAs, phones,TinyOS 1.0, 1.1.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gateway Nodes</th>
<th>CPU</th>
<th>Power</th>
<th>Memory</th>
<th>I/O and Sensors</th>
<th>Radio</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stargate 2003</td>
<td>Intel PXA255</td>
<td>64KNSRM</td>
<td>2 PCMICA/CF, com ports, Ethernet, USB</td>
<td>Serial connection to sensor network</td>
<td>Flexible I/O and small form factor power management.</td>
<td></td>
</tr>
<tr>
<td>Inrysnc Cerfcube 2003</td>
<td>Intel PXA255</td>
<td>32KB Flash 64KB SRAM</td>
<td>Single CF card, general-purpose I/O</td>
<td></td>
<td>Small form factor, robust industrial support, Linux and Windows CE support.</td>
<td></td>
</tr>
<tr>
<td>PC104 nodes</td>
<td>X86 processor</td>
<td>32KB Flash 64KB SRAM</td>
<td>PCI Bus</td>
<td></td>
<td>Embedded Linux or Windows support.</td>
<td></td>
</tr>
</tbody>
</table>
GRAIN-SIZED
RFIDs are powered by inductive coupling to a transmission from a reader device to transmit a message back, and are available commercially at very low prices.

Computation power is severely limited, usually they only transmit stored unique id and variable.
Spec mote (2003)

size 2x2.5mm, AVR RISC core, 3KB memory, FSK radio (CC1000), encrypted communication hardware support, memory-mapped active messages
MATCHBOX-SIZED
Matchbox-sized nodes

Examples are, Mica, Mica2, Telos motes, XSM node

8-bit microprocessor, 4MHz CPU ATMEGA 128, ATMEL 8535, or Motorola HCS08

~8Kb RAM, holds run-time state (values of the variables) of the program
Flash memory in motes

~128Kb programmable Flash memory, holds the application program which is downloaded via a programmer-board or wirelessly

additional Flash memory storage space up to 512Kb for logging sensor data
Mica2 and MicaDot

ATmega128 CPU
Self-programming
Chipcon CC1000
FSK, Tunable frequency
2 AA battery = 3V
Basic sensor board

Light (Photo)

Temperature

Prototyping space for new hardware designs
Mica sensor board

- Light
- Temperature
- Acceleration 2 axis
  Resolution: ±2mg
- Magnetometer
  Resolution: 134 μG
- Microphone
- Sounder 4.5kHz
Magnetometer/compass

Resolution: 400 μ Gauss

Three axis, under $15 in large quantities
Ultrasonic transceiver

Used for ranging
Up to 2.5m range
6cm accuracy
Dedicated microprocessor
Mica weather board

Photosynthetically Active Radiation
Humidity
Temperature
Barometric Pressure
Acceleration 2 axis
UCB, Crossbow, UCLA
MicaDot sensorboards

“Dot” sensorboards (1” diameter)

HoneyDot: Magnetometer

Ultrasonic Transceiver

Weather Station
XSM node

Derived from Mica2

Better sensor range

- 4 Passive Infrared: ~25m for SUV
- Sounder: ~10m
- Microphone: ~50m for ATV
- Magnetometer: ~7m for SUV

Better radio range ~30m

Grenade timer

Wakeup circuits (Mic, PIR)
Telos mote

Low Power

Integrated antenna (50m - 125m)

USB

IEEE 802.15.4

(CC2420 radio)

10kB RAM, 16-bit core

DMA transfers while CPU off
Telos is low power

<table>
<thead>
<tr>
<th>Operation</th>
<th>Telos</th>
<th>Mica2</th>
<th>MicaZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Voltage</td>
<td>1.8V</td>
<td>2.7V</td>
<td>2.7V</td>
</tr>
<tr>
<td>Mote Standby (RTC on)</td>
<td>5.1 μA</td>
<td>19.0 μA</td>
<td>27.0 μA</td>
</tr>
<tr>
<td>MCU Idle (DCO on)</td>
<td>54.5 μA</td>
<td>3.2 mA</td>
<td>3.2 mA</td>
</tr>
<tr>
<td>MCU Active</td>
<td>1.8 mA</td>
<td>8.0 mA</td>
<td>8.0 mA</td>
</tr>
<tr>
<td>MCU + Radio RX</td>
<td>21.8 mA</td>
<td>15.1 mA</td>
<td>23.3 mA</td>
</tr>
<tr>
<td>MCU + Radio TX (0dBm)</td>
<td>19.5 mA</td>
<td>25.4 mA</td>
<td>21.0 mA</td>
</tr>
<tr>
<td>MCU + Flash Read</td>
<td>4.1 mA</td>
<td>9.4 mA</td>
<td>9.4 mA</td>
</tr>
<tr>
<td>MCU + Flash Write</td>
<td>15.1 mA</td>
<td>21.6 mA</td>
<td>21.6 mA</td>
</tr>
<tr>
<td>MCU Wakeup</td>
<td>6 μs</td>
<td>180 μs</td>
<td>180 μs</td>
</tr>
<tr>
<td>Radio Wakeup</td>
<td>580 μs</td>
<td>1800 μs</td>
<td>860 μs</td>
</tr>
</tbody>
</table>
BRICK-SIZED
Stargate

Mini Linux computers communicating via 802.11 radios
Computationally powerful
High bandwidth
Requires more energy (AA infeasible)
Used as a gateway between the Internet and WSN
Manufacturers

Crossbow (www.xbow.com): Mica2, Dot, Micaz, Dot

Intel Research: Stargate, iMote, iMote2

Moteiv: Telos Mote

Dust Inc: Smart Dust

Sensoria Corporation (www.sensoria.com): WINS NG

Millenial Net (www.millenial.com): iBean sensor nodes

 Ember (www.ember.com): IEEE 802.15.4 (zigbee) nodes
# Challenges in WSNs

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy constraint</td>
<td>battery powered</td>
</tr>
<tr>
<td>Unreliable commn.</td>
<td>limited bursty bandwidth</td>
</tr>
<tr>
<td>Unreliable sensors</td>
<td>false positives</td>
</tr>
<tr>
<td>Ad hoc deployment</td>
<td>no pre-configuration</td>
</tr>
<tr>
<td>Large scale networks</td>
<td>inscalable algorithms</td>
</tr>
<tr>
<td>Limited computation</td>
<td>no centralized algorithms</td>
</tr>
<tr>
<td>Distributed execution</td>
<td>difficult to debug &amp; get it right</td>
</tr>
</tbody>
</table>
# Opportunities in WSNs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundancy</td>
<td>Many nodes in same area</td>
</tr>
<tr>
<td>Precise clock at nodes</td>
<td>Synchronized clocks</td>
</tr>
<tr>
<td>Atomic broadcast primitive</td>
<td>All recipients hear same message at same time</td>
</tr>
<tr>
<td>Geometry</td>
<td>Dense nodes over 2D</td>
</tr>
<tr>
<td>New applications</td>
<td>Tracking, querying, localization, network reprogramming, etc.</td>
</tr>
</tbody>
</table>
SOFTWARE PLATFORMS
TinyOS

Most popular OS for WSN developed by UC Berkeley

Features a component-based architecture

software is written in modular components

each component denotes the interfaces that it provides

an interface declares a set of functions called commands that the interface provider implements and another set of functions called events that the interface user should be ready to handle

Easy to link components together by “wiring” their interfaces to form larger components similar to using Lego blocks
TinyOS ...

Provides a component library that includes network protocols, services, and sensor drivers

An application consists of

1) a component written by the application developer and

2) the library components that are used by the components in (1)

An application developer writes only the application component that describes the sensors used, and configures the middleware services with parameters
Benefits of using TinyOS

1) Separation of concerns

TinyOS provides a proper networking stack for wireless communication that abstracts away the underlying problems and complexity of message transfer from the application developer.

E.g., MAC layer
Benefits of using TinyOS...

2) Concurrency control

TinyOS provides a scheduler that achieves efficient concurrency control (at the node level)

An interrupt-driven execution model is needed to achieve a quick response time for the events and capture the data

For example, a message transmission may take up to 100msec, and without an interrupt-driven approach the node would miss sensing and processing of interesting data in this period

TinyOS scheduler takes care of the intricacies of interrupt-driven execution and provides concurrency in a safe manner by scheduling the execution in small threads
Benefits of using TinyOS...

3) Modularity

TinyOS’s component model facilitates reuse and reconfigurability since software is written in small functional modules. Several middleware services are available as well-documented components.

Over 500 research groups and companies are using TinyOS and numerous groups are actively contributing code to the public domain.
TinyOS concepts

Microthreaded OS (lightweight thread support) and efficient network interfaces

Two level scheduling structure

Long running tasks that can be interrupted by hardware events

Small, tightly integrated design allows crossover of software components into hardware
TinyOS concepts...

Scheduler + Graph of Components

Component includes:

Commands

Event Handlers

Tasks (concurrency)

Frame (storage) per component, shared stack, no heap
Application is a graph of components
TinyOS execution model

Commands request action
ack/nack at every boundary
call command or post task

Events notify occurrence
hardware interrupt at lowest level
signal event, call command, or post task

Split-phase operations
command-acked quickly, work done by task, event signals completion
Event-driven sensing app.

```c
command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 200);
}

event result_t Timer.fired() {
    return call sensor.getData();
}

event result_t sensor.dataReady(uint16_t data) {
    display(data)
    return SUCCESS;
}
```

clock event handler initiates data collection
sensor signals data ready event
data event handler calls output command
device sleeps or handles other activity while waiting
conservative send/ack at component boundary
TinyOS commands & events

```c
{  
  ...
  status = call CmdName(args)
  ...
}
```

```c
event EvtName(args) {
  ...
  return status;
}
```

```c
command CmdName(args) {
  ...
  return status;
}
```

```c
{  
  ...
  status = signal EvtName(args)
  ...
}
```
TinyOS execution contexts

Events generated by interrupts preempt tasks

Tasks do not preempt tasks
Tasks

Provide concurrency internal to a component, and longer running operations

Tasks are preempted by events, able to perform operations beyond event context, may call commands, may signal events, not preempted by tasks
Typical use of tasks

event driven data acquisition

schedule task to do computational portion

```c

event result_t sensor.dataReady(uint16_t data) {
    putdata(data);
    post processData();
    return SUCCESS;
}

_task void processData() {
    int16_t i, sum=0;
    for (i=0; i < maxdata; i++)
        sum += (rdata[i] >> 7);
    display(sum >> shiftdata);
}
```

Task scheduling

Currently simple fifo scheduler

Bounded number of pending tasks

When idle, shuts down node except clock

Uses non-blocking task queue data structure

Simple event-driven structure + control over complete application/system graph instead of complex task priorities
Maintaining schedule agility

Need logical concurrency at many levels of the graph

While meeting hard timing constraints, sample the radio in every bit window

Retain event-driven structure throughout application

Tasks extend processing outside event window

All operations are non-blocking
The complete application

- RadioTiming
- SecDedEncode
- RadioCRCPacket
- UART
- UARTnoCRCPacket
- ADC
- phototemp
- AMStandard
- ClockC
- bit
- byte
- packet
- SenseToRfm
- IntToRfm
- MicaHighSpeedRadioM
- RandomLFSR
- SPIByteFIFO
- SlavePin
- CRCfilter
- noCRCPacket
- Timer
- photo
- phototemp
- HW
- SW
TINYOS SYNTAX
TinyOS

TinyOS 2.0 is written in an extension of C, called nesC, applications are also in nesC

NesC provides syntax for TinyOS concurrency and storage model: commands, events, tasks, local frame variable

Compositional support: separation of definition and linkage, robustness through narrow interfaces and reuse

Whole system analysis and optimization
A component specifies a set of interfaces by which it is connected to other components:

- provides a set of interfaces to others, and
- uses a set of interfaces provided by others

**Interfaces are bidirectional:** includes commands and

```plaintext
provides
  interface StdControl;
  interface Timer:
uses
  interface Clock
```
Component Interface

logically related set of commands and events

StdControl.nc

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

Clock.nc

interface Clock {
    command result_t setRate(char interval, char scale);
    event result_t fire();
}
Component types

Configurations:
link together components to compose new component
configurations can be nested
complete “main” application is always a configuration

Modules:
provides code that implements one or more interfaces and internal behavior
configuration Blink {
}

implementation {
    components Main, BlinkM, TimerC, LedsC;

    Main.StdControl -> TimerC.StdControl;
    Main.StdControl -> BlinkM.StdControl;

    BlinkM.Timer -> TimerC.Timer[unique("Timer")];
    BlinkM.Leds -> LedsC;
}
module BlinkM {
    provides interface StdControl;
    uses interface Timer;
    uses interface Leds;
}

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 Clock.fire() {
        call Leds.redToggle();
        return SUCCESS;
    }
}
configuration SenseToRfm {
}
implementation {
  components Main, SenseToInt, IntToRfm, TimerC, Photo as Sensor;

  Main.StdControl -> SenseToInt;
  Main.StdControl -> IntToRfm;

  SenseToInt.Timer -> TimerC.Timer[unique"Timer"];  
  SenseToInt.ADC -> Sensor;
  SenseToInt.ADCControl -> Sensor;
  SenseToInt.IntOutput -> IntToRfm;
}

SenseToRFM example
Nested configuration

includes IntMsg;
configuration IntToRfm
{
    provides {
        interface IntOutput;
        interface StdControl;
    }
}
implementation
{
    components IntToRfmM, GenericComm as Comm;
    IntOutput = IntToRfmM;
    StdControl = IntToRfmM;
    IntToRfmM.Send -> Comm.SendMsg[AM_INTMSG];
    IntToRfmM.SubControl -> Comm;
}
IntToRFM module

includes IntMsg;

module IntToRfmM
{
  uses {
    interface StdControl as SubControl;
    interface SendMsg as Send;
  }
  provides {
    interface IntOutput;
    interface StdControl;
  }
}

implementation
{
  bool pending;
  struct TOS_Msg data;

  command result_t StdControl.init() {
    pending = FALSE;
    return call SubControl.init();
  }

  command result_t StdControl.start() {
    return call SubControl.start();
  }

  command result_t StdControl.stop() {
    return call SubControl.stop();
  }

  command result_t IntOutput.output(uint16_t value) {
    ...
    if (call Send.send(TOS_BCAST_ADDR, sizeof(IntMsg), &data)
        return SUCCESS;
    ...
  }

  event result_t Send.sendDone(TOS_MsgPtr msg, result_t success) {
    ...
  }
}
Atomicity support in nesC

Split phase operations require care to deal with pending operations

Race conditions may occur when shared state is accessed by preemptible executions, e.g. when an event accesses a shared state, or when a task updates state (preemptible by an event which then uses that state)

nesC supports atomic block

implemented by turning of interrupts

for efficiency, no calls are allowed in block

access to shared variable outside atomic block is not allowed
Supporting hw evolution

Component design so HW and SW look the same

e.g. temp component

may abstract particular channel of ADC on the microcontroller

may be a SW I2C protocol to a sensor board with digital sensor or ADC

HW/SW boundary can move up and down with minimal changes
bool pending;
struct TOS_Msg data;
command result_t IntOutput.output(uint16_t value) {
    IntMsg *message = (IntMsg *)data.data;
    if (!pending) {
        pending = TRUE;
        message->val = value;
        message->src = TOS_LOCAL_ADDRESS;
        if (call Send.send(TOS_BCAST_ADDR, sizeof(IntMsg), &data))
            return SUCCESS;
        pending = FALSE;
    }
    return FAIL;
}
event result_t IntOutput.sendDone(TOS_MsgPtr msg, result_t success)
{
    if (pending && msg == &data) {
        pending = FALSE;
        signal IntOutput.outputComplete(success);
    }
    return SUCCESS;
}
TinyOS limitations

Static allocation allows for compile-time analysis, but can make programming harder

No support for heterogeneity

Limited visibility, Debugging, Intra-node ft-tolerance
TinyOS tools...

TOSSIM: a simulator for tinyos programs

ListenRaw, SerialForwarder: java tools to receive raw packets on PC from base node

Oscilloscope: java tool to visualize sense data real time

Memory usage: breaks down memory usage per component (in contrib)
TinyOS tools

Peacekeeper: detect RAM corruption due to stack overflows (in lib)

Stopwatch: tool to measure execution time of code block by timestamping at entry and exit (in osu CVS server)

Makedoc and graphviz: generate and visualize component hierarchy

Surge, Deluge, SNMS, TinyDB