This lecture will answer:
• What are building blocks of a WSN?
• What is a WSN used for?

Structure:
• Hardware platforms (“motes”)
• Sensing applications
• Canonical problems
• Examples
• Operating systems
WSN Platforms

What are the differences between WSN platforms and typical computers?

- Battery power
  - Goal: maximum system lifetime with no recharge/replacement
- Low-power radios for communication
  - 10-200kbit/sec
- Small CPUs
  - E.g. 8bit, 4k RAM.
- Flash storage
- Sensors

Battery Power

Example: Mica2 “mote”

- Total battery capacity: 2500mAH (2 AA cells)
- System consumption: 25 mA (CPU and radio on)
- Lifetime: 100 hours (4 days)

Alternatives:
- Bigger batteries
- Solar/wind/… (“energy harvesting”)
- **Duty cycling**
Low Power Radios

• ISM band – 430, 900, or 2400 MHz
• Varying modulation and protocol:
  – Custom (FSK?) – Mica2, 20 kbit/s
  – Bluetooth
  – Zigbee (802.15.4) - ~200kbit/sec
• Short range
  – Typically <100 meters
• Low power. E.g. Chipcon CC2420:
  – 9-17 mA transmit (depending on output level)
  – 19 mA receive
• Listening can take more energy than transmitting

Small CPUs

• Example: Atmel AVR
  – 8 bit
  – 4 KB RAM
  – 128 KB code flash
  – ~2 MIPS @ 8MHz
  – ~8 mA
• Example: TI MSP430
  – 16 bit (sort of)
  – 10 KB RAM
  – 48 KB code flash
  – 2 mA

Higher-powered processors:

ARM7 (Yale XYZ platform)
  32 bit, 50 MHz, >>1MB RAM

ARM9 (StarGate, others)
  32 bit, 400 MHz, >>16MB RAM
Flash Storage

Raw flash
- Small (serial NOR), very low power (NAND)
- Page-at-a-time write
- No overwrite without erasing
- Divided into pages and erase blocks
- Typical values: 512B pages, 32 pages in erase block
- Garbage collection needed to gather free pages for erasing

“Cooked” flash
- Disk-like interface
- 512B re-writable blocks
- Very convenient
- Higher power consumption

Sensors
- Temperature
- Humidity
- Magnetometer
- Vibration
- Acoustic
- Light
- Motion (e.g. passive IR)
- Imaging (cameras)
- Ultrasonic ranging
- GPS
- Lots of others…
Sensor Applications

- Data driven
  - Distributed computation, not communication network
- Homogeneous
  - All sensors typically participate in the same application(s)
- Typical architecture: data collection, fusion, and transport

Canonical WSN Problems

- Localization
- Time Synchronization
- Routing
- Duty cycled networking
- Data aggregation
Localization

Determining relative or absolute location of a sensor

Solutions:
- GPS
- Ranging and triangulation
  - Radio strength (RSSI)
  - RF time-of-flight (interferometry)
  - Acoustic time-of-flight
- Directional triangulation
  - Acoustic – phase measurement

Problems in Localization

- GPS is expensive, sometimes difficult to use, and power-hungry
  - Requires line-of-sight to 3 or 4 satellites overhead
  - 80mA for 1-5 minutes to acquire fix
- Radio ranging is not accurate
- Acoustic ranging is limited
  - Range
  - Applications
- Sensitivity to errors
  - Robust triangulation is hard
Time Synchronization

- Applications:
  - Event detection by arrival time
    - E.g. gunshot triangulation
  - Duty cycling synchronization
- External reference
  - GPS, WWV
- Autonomous synchronization
  - Receiver-receiver
  - Sender-receiver
  - Drift estimation

Autonomous Synchronization

Idea:
  - Sample time at A
  - Transmit to B

Issues:
  - B receives $T_A$ at $T_A + \Delta$
  - Software delays ($T_{tx}$, $T_{rx}$)
  - Channel acquisition ($T_{mac}$)
  - Propagation delay ($T_{prop}$)

Clock drift
  - Quartz crystal:
    - 50 ppm = 50µS/s = 180ms/hr
  - Varies with e.g. temperature
Synchronization methods

• Receiver-receiver  
  – Eliminate transmit uncertainty

  \[ T_A = T_B \pm T_{rx} \]

• Sender-receiver  
  – Reduce transmit uncertainty

• Drift estimation  
  – Estimate and correct

\[ T_B = T_A + T_{prop} \]

Routing

• What addresses make sense in a sensor network?  
  – Location  
  – Data

• Geographic routing  
  – GPSR  
  – Beacon routing

• Flooding, tree construction  
  – Data collection architectures

GPSR – forward to node physically closest to destination
More Routing

• How to handle duty cycling?
  – Sleep or go around?

• Wireless vs. wired

More Routing

• Network lifetime
  – More packets = more battery drain

1 packet/s

4 packet/s

Data sink
Duty Cycled Networking

Problem: continuous listening is too expensive
Solution: listen periodically

Low-power listening

Synchronized low-power listening

Example - Directed Diffusion

- **Name data** (not nodes), use physicality
- **Sensors publish** event notifications and **users subscribe** to specific types.
- Optimize path with **gradient-based feedback**
- Opportunistic **in-network aggregation** and **nested queries**.
Directed Diffusion

• Expressing an Interest
  – Using attribute-value pairs
  – E.g.,
    
    | Type         | Wheeled vehicle // detect vehicle location |
    |--------------|--------------------------------------------|
    | Interval     | 20 ms // send events every 20ms            |
    | Duration     | 10 s // Send for next 10 s                |
    | Field        | [x1, y1, x2, y2] // from sensors in this area |

• Uses publish/subscribe
  – Inquirer expresses an interest, \( i \), using attribute values
  – Sensor sources that can service \( i \), reply with data

Gradient-based Routing

• Inquirer (sink) broadcasts exploratory interest, \( i_1 \)
  – Intended to discover routes between source and sink

• Neighbors update interest-cache and forwards \( i_1 \)

• Gradient for \( i_1 \) set up to upstream neighbor
  – No source routes
  – Gradient – a weighted reverse link
  – Low gradient \( \rightarrow \) Few packets per unit time needed

Bidirectional gradients established on all links through flooding
Examples - TinyDB

TinySQL:

SELECT <aggregates>, <attributes>
[FROM {sensors | <buffer>}]  
[WHERE <predicates>]  
[GROUP BY <exprs>]  
[SAMPLE PERIOD <const> | ONCE]  
[INTO <buffer>]  
[TRIGGER ACTION <command>]

Data Model

• Entire sensor network as one single, infinitely-long logical table: sensors
• Columns consist of all the attributes defined in the network
• Typical attributes:
  – Sensor readings
  – Meta-data: node id, location, etc.
  – Internal states: routing tree parent, timestamp, queue length, etc.
• Nodes return NULL for unknown attributes
• On server, all attributes are defined in catalog.xml
Acquisitional Query Processing

- What’s really new & different about databases on (mote-based) sensor networks?
- This paper’s answer:
  - Long running queries on physically embedded devices that control when and where and with what frequency data is collected
  - Versus traditional DBMS where data is provided a priori
- For a distributed, embedded sensing environment, ACQP provides a framework for addressing issues of
  - When, where, and how often data is sensed/sampled
  - Which data is delivered

PRESTO: Model-driven Push

Insight:
- Models are expensive to create, but simple to check
- Data which can be predicted does not need to be reported.
Data Management

• Skip this one…

Operating Systems

What features does an operating system need?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Unix</th>
<th>TinyOS</th>
<th>SOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware drivers, system init</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Loadable programs</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>File system</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Resource allocation (e.g. memory)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Processes / threads</td>
<td>Yes</td>
<td>No</td>
<td>Sort of</td>
</tr>
<tr>
<td>Networking support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IPC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Event scheduling / timers</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
TinyOS & nesC Concepts

- New Language: nesC. Basic unit of code = Component

- Component
  - Process Commands
  - Throws Events
  - Has a Frame for storing local state
  - Uses Tasks for concurrency

- Components provide interfaces
  - Used by other components to communicate with this component

- Components are wired to each other in a configuration to connect them

Application = Graph of Components
TinyOS Code Structure

Y.multiRead()

Return OK post A()

(A runs sometime)

Flash. read()

Flash. readDone()

If (bytes remain)

post A()

Else

signal Y.multiReadDone()

Y.multiReadDone()

SOS

• Micro-kernel architecture
  – User-space, kernel-space separation
  – Supports dynamic, run-time addition of modules
  – Memory protection possible between module & kernel space

• Each application has one or more modules
  – Within a module, interaction uses regular function calls
  – Modules interact by passing messages
  – Modules can retain state, allocate / deallocate memory

Module 1

Module 2

Module-space

Micro-kernel

Kernel-space
Modules: SOS vs TinyOS

- Threading implemented as macros

```c
#include "pt.h"
struct pt pt;

PT_THREAD(example(struct pt *pt))
{
    PT_BEGIN(pt);
    while(1)
    {
        if(initiate_io())
        {
            timer_start(&timer);
            PT_WAIT_UNTIL(pt, io_completed()) || timer_expired(&timer));
            read_data();
        }
    }
    PT_END(pt);
}
```

TinyOS – compile-time    SOS – run-time
Wrap-up

• What did we talk about?

• Energy management
  – Esp. duty-cycled radios

• Routing
  – By naming and finding information or locations

• In-network processing
  – Aggregation (tinyDB)
  – Model checking (PRESTO)

• Light weight operating systems