Internet of Things

Tutorials
Ing. Alessandro Redondi
DEIB - 3° Floor

Email: alessandroenrico.redondi@polimi.it
Phone: 02 2399 3403
Classes Objectives

- Giving you an overview on the software used in the IoT community
- Stimulating your curiosity
- Providing you with basic tools to develop simple IoT applications
- Classes time is limited -> Play with these tools on your own!!
Calendar of upcoming events

- 29 Mar 2017 – Introduction
- 12 Apr 2017 – ThingSpeak, Node-Red
- 26 Apr 2017 – TinyOS #1
- 10 May 2017 – TinyOS #2
- 17 May 2017 - Contiki
- 31 May 2017 - Projects
Internet of Things
Applications and enabling tools
Agenda

- A Bit of Context on the IoT and Wireless Sensor Networks (WSNs):
  - Applications, Challenges and Sensor Platforms
- Designing OS for networked sensors
  - Requirements and guidelines
  - The TinyOS example
A Bit of Context on WSNs
Technology, Applications and Sensor Nodes
New Class of Computing

log (people per computer)

year

Mainframe
Minicomputer
Workstation
PC
Laptop
PDA
Technology Push

- CMOS miniaturization
- Micro-sensors (MEMS, Materials, Circuits)
  - acceleration, vibration, gyroscope, tilt, magnetic, heat, motion, pressure, temp, light, moisture, humidity, barometric
  - chemical (CO, CO2, radon), biological, microradar, ...
  - actuators too (mirrors, motors, smart surfaces, micro-robots)
- Communication
  - short range, low bit-rate, CMOS radios (1-10 mW)
- Power
  - batteries remain primary storage (1,000 mW*s/mm³), fuel cells 10x
  - solar (10 mW/cm², 0.1 mW indoors), vibration (~uW/gm), flow
  - 1 cm³ battery => 1 year at 10 msgs/sec
Application Pull

- Monitoring Environments
  - habitat monitoring, conservation biology, ...
  - Precision agriculture, land conservation, ...
  - built environment comfort & efficiency ...
  - alarms, security, surveillance, treaty verification

- Monitoring Structures and Things
  - structural response, condition-based maintenance
  - disaster management
  - urban terrain mapping & monitoring

- Interactive Environments
  - manufacturing, asset tracking, fleet & franchise
  - context aware computing, non-verbal communication
  - Assistance
    - home/elder care
Recent applications

- Smart door locks ([http://remotelock.com](http://remotelock.com))
- BLE trackers ([https://www.thetileapp.com](https://www.thetileapp.com))
- Connected kitchens ([Amazon Dash, GeniCan, Hiku](#))
- Flatware ([Vessyl, Hapifork](#))
- Monitoring ([Hydropoint, Nest](#))
Common features

- Low-cost hardware
- Data sensing
- Wireless communication
- Data logging / storing / analysis
WSNs: Potentials and Challenges

- Potentials
  - Cost viability
  - Flexibility
  - Short time-to-deployment (can be built with off-the-shelf technology)
  - Numbers (# nodes >> # people)

- Challenges at different layers
  - Energy Efficient protocols design
  - Self configurability or planning
  - Robustness
  - Coverage & Connectivity
  - OS Design
Sensor Hardware

Location Finding System
- Sensor
- ADC

Processor
- Memory

Mobilizer
- Transceiver

Power Unit

ANTENNA
MICAz Platform

- Microprocessor: Atmel ATmega128L
  - 8 MHz clock
  - 128 kB of Flash for program memory
  - 4 kB of SRAM for data and variables
  - 2 UARTs (Universal Asynchronous Receive and Transmit)
  - Serial Port Interface (SPI) bus
  - Dedicated hardware I2C bus
- Radio: Chipcon’s CC2420 (IEEE 802.15.4)
  - 250 kbit/s
- External serial flash memory: 512 Kb
  - About 100,000 samples
- 51-pin expansion connector
  - Eight 10-bit analog I/O
  - 21 general purpose digital I/O
- User interface: 3 programmable LEDs
- Powered by two AA batteries
Rich Sensor board

Mica PINS

PHOTO

TEMP

MAGNETOMETER

X Axis

Y Axis

SOUNDER

MICROPHONE

ACCELEROMETER

Gain Adjustment

ADC Signals (ADC1-ADC6)

On/Off Control

I²C Bus

Interrupt

Microphone

Sounder

Magnetometer

Light Sensor

Temperature Sensor

2.25 in

1.25 in
Designing OS for networked sensors

Requirements and guidelines
Traditional Architectures

- Well established layers of abstractions
- Strict boundaries
- Ample resources
- Well attended
Kernel Based Architectures

Problems
- Large memory & storage requirement
- Unnecessary and overkill functionality
- Address space isolation, complex I/O subsystem, UI
- Relative high system overhead, e.g., context switch
- Require complex and power consuming hardware support
“General-Purpose” OS?

MANY functionalities & programming APIs

- Protection between “untrusted” applications and kernel
  - Overhead for crossing kernel/user boundary & interrupt handling

- Multi-threaded architecture
  - Large number of threads → large memory
  - Context switch overhead

- I/O model
  - Blocking I/O: waste memory on blocked threads
  - Polling: waste CPU cycles and power

- Need a new OS architecture!
Sensor OS Requirements

- Small footprint
- Low system overhead
- Low power consumption
- Application-oriented design
- Flexibility in supporting multiple applications
- Solutions:
  - TinyOS
  - Mote Runner
  - Contiki
  - Riot
  - NanoRK
  - LiteOS
  - FreeRTOS
“TinyOS is an open-source operating system designed for wireless embedded sensor networks”

http://www.tinyos.net/
TinyOS Overview

- Event-driven architecture
  - OS operations are triggered by hardware interrupt (asynchronous management)
- Single shared stack
- No kernel/user space differentiation

Diagram:
- Main (includes Scheduler)
- Application (User Components)
- Actuating
- Sensing
- Communication
- Hardware Abstractions
TinyOS Architecture
Overview

- **NO** Kernel - Direct hardware manipulation
- **NO** Process management - Only one process on the fly
- **NO** Virtual memory - Single linear physical address space
- **NO** Dynamic memory allocation - Assigned at compile time
- **NO** Software signal or exception - Function call
TinyOS “Ingredients”

- TinyOS is not an OS in traditional sense
- Provides a programming framework to build application-specific OS instances
- Programming Framework made of:
  - Scheduler (always there)
  - Components
  - Interfaces
TinyOS Concepts

- Scheduler + Graph of Components
  - constrained two-level scheduling model: tasks + events

- A component
  - specifies a set of interfaces by which it is connected to other components
  - provides a set of interfaces to others
  - uses a set of interfaces provided by others

- Interfaces are bi-directional
  - include commands and events

- Constrained Storage Model
  - frame per component, shared stack, no heap

---

**Clock.nc**

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

**StdControl.nc**

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

- Event is independent of FIFO scheduler

- Lowest level events are supported directly by Hardware interrupt

- Software events propagate from lower level to upper level through function call
TASKS

- Provide concurrency internal to a component
  - longer running operations
- Are preempted by events
- Able to perform operations beyond event context
- May call commands
- May signal events
- Not preempted by tasks

```c
{  
  ...  
  post TskName();  
  ...  
}

task void TskName()  
{  
  ...  
}
```
Typical application 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;
}

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

- 128 Hz sampling rate
- Simple FIR filter
- Dynamic software tuning for centering the magnetometer signal (1208 bytes)
- Digital control of analog, not DSP
- ADC (196 bytes)
**Tasks - Examples**

- Transmit packet
  - Send command schedules task to calculate CRC
  - Task initiated byte-level data pump
  - Events keep the pump flowing
- Receive packet
  - Receive event schedules task to check CRC
  - Task signals packet ready if OK
- Byte-level TX/RX
  - Task scheduled to encode/decode each complete byte
  - Must take less time that byte data transfer
TOS Execution Model

- Commands request action
  - ack/nack at every boundary
  - call cmd or post task
- Events notify occurrence
  - HW interrupt at lowest level
  - may signal events
  - call commands
  - post tasks
- Tasks provide logical concurrency
  - preempted by events
- Migration of HW/SW boundary
An Example Application

- **application**
  - **routing**
  - **messaging**
    - **packet**
      - **byte**
  - **Radio byte (MAC)**
  - **Routing Layer**
  - **Messaging Layer**

**sensing application**

- **application**
  - **bits**
    - **RFM**
  - **byte**
    - **Radio byte (MAC)**
  - **packet**
    - **Radio Packet**
  - **messaging**
    - **Routing Layer**
    - **Messaging Layer**

**HW**
- **SW**
  - **photo**
  - **Temp**
  - **ADC**
  - **i2c**
  - **clocks**
Event-Driven Sensor Access Pattern

- 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

```c
command result_t StdControl.start() {
    return call Timer.start(TIMERO_REPEAT, 200);
}
event result_t Timer.fired() {
    return call Sensor.getData();
}
event result_t Sensor.dataReady(uint16_t data) {
    display(data);
    call Led.Toggle();
    return SUCCESS;
}
```
Tasks do computations
- Non-preemptive FIFO scheduling
- Bounded number of pending tasks

Events handle concurrent dataflows
- Interrupts trigger lowest level events
- Events preempt tasks, tasks do not
- Events can signal events, call commands, or post tasks
TinyOS Scheduling

- Event and Task
  - Tasks cannot preempt other tasks
  - Single shared stack
    - Used by both interrupts and function calls
  - Simple FIFO scheduler
  - Events can preempt a task
  - Events can preempt each other
  - When idle, scheduler shuts down the node except for clock
A virtual machine for the IoT
Motivations

- Many software are available for designing WSNs and IoT solutions
- No “standard” configuration is available
- Depending on the application, one solution may be preferable

- Even the installation of a programming environment itself is a time-consuming task
Example: TinyOS

- TinyOS is quite easy to install on Debian-based systems (Linux)
- For installation under Windows, cygwin is required...
- On Mac OS X... nightmare!

- All these issues may discourage researchers to play with it
- Similar issues also when installing other products
Solution: Virtual Machine

- A Virtual Machine is an emulation of a particular computer system
- On the class website, a Virtual Machine containing most of the needed tools is available:

http://home.deib.polimi.it/redondi/IoT/IOT-ubuntu.ova
VM contents

- Contiki env + examples
- TinyOS env + examples
- Moterunner env + examples
- Xively libraries and examples
- Sicsthsense examples
- Node-RED
How to run it

1. Download Oracle VM VirtualBox for your laptop (make sure to have at least 10 GB of free space on your HD)

2. Download the IOT virtual machine from the class website

3. Open VirtualBox and go to File->Import Appliance and import the file IOT-ubuntu.ova

4. Start using the system! (password is “user”)
A small IOT example

- A proximity detector using:
  - TelosB sensor nodes
  - TinyOS as operating system
  - Node-RED for data logging and visualization