Principles of Wireless Sensor Networks

https://www.kth.se/social/course/EL2745/

Lecture 2

Introduction to Programming WSNs

Piergiuseppe Di Marco
Ericsson Research
e-mail:pidm@kth.se
http://pidm.droppages.com/

Royal Institute of Technology
Stockholm, Sweden

September 1, 2015

Course content

- Part 1
  - Lec 1: Introduction to WSNs
  - Lec 2: Introduction to Programming WSNs
- Part 2
  - Lec 3: Wireless Channel
  - Lec 4: Physical Layer
  - Lec 5: Medium Access Control Layer
  - Lec 6: Routing
- Part 3
  - Lec 7: Distributed Detection
  - Lec 8: Static Distributed Estimation
  - Lec 9: Dynamic Distributed Estimation
  - Lec 10: Positioning and Localization
  - Lec 11: Time Synchronization
- Part 4
  - Lec 12: Wireless Sensor Network Control Systems 1
  - Lec 13: Wireless Sensor Network Control Systems 2

WSN Programming

Introduction

WSN Programming Challenges

- Scarce resources
  - Low cost, small size platforms
  - Limited memory (kB)
- Event-driven programming model
  - Reactive approach, depending on interaction with environment
  - Data collection / control specific, opposite to general purpose computation
- Reliability / durable stand-alone operation
  - Applications run for months/years without human intervention
  - Reduce run time errors and complexity
- Soft real-time requirements
  - Few time-critical tasks (sensor acquisition and radio timing)
  - Timing constraints through complete control over application

Programming WSNs

- Software
  - Programming language (e.g., nesC, C, C++. Java,...)
    - Compiler, etc.
  - OS/runtime libraries (e.g., TinyOS, ContikiOS, mbedOS, MantisOS, LiteOS,...)
    - Access to system resources
    - Application programming interfaces (APIs) for communication, sensors, etc.

- Hardware
  - WSN module (e.g., TelosB, Tmote Sky, MicaZ, ARM mbed,...)

CPU	16bit, 8 MHz
RAM	10kB
ROM	48kB
Flash	1MB
Batteries	2xAA

Today's Topics

- NesC
  - designed specifically for WSN programming
- TinyOS
  - widespread, especially in academia, large community, easy support,...
- Examples
- Hands-on
  - Your turn!

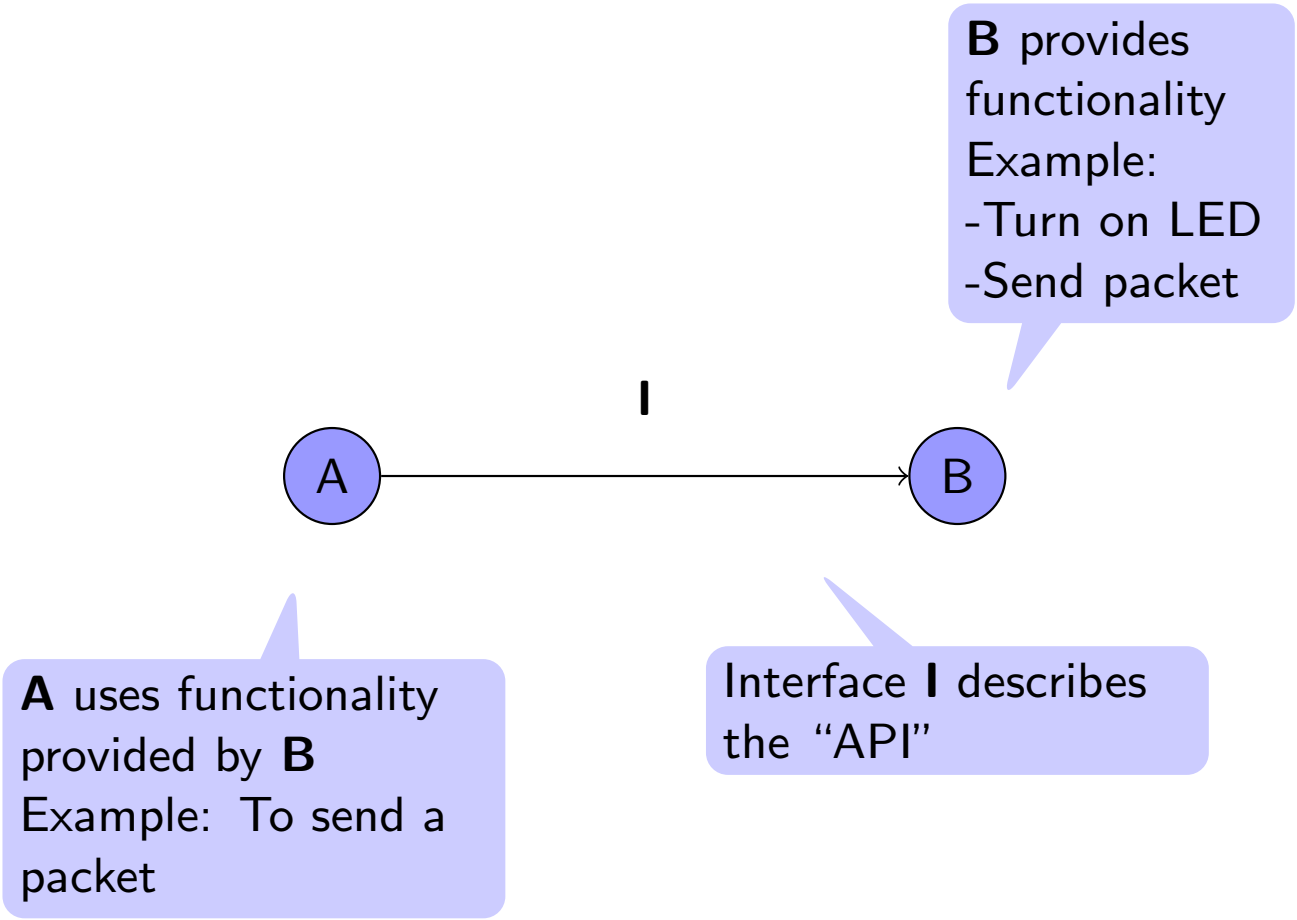
TinyOS and NesC

Component Model

Network Embedded Systems C (NesC)

- NesC is the programming language used to build applications for the TinyOS platform
- It is based on C language (a dialect of C) with extra features
- The basic unit of NesC code is a **component**
- Components connect via **interfaces**
- Connections are called "wiring"
- Interfaces are used to:
  - Access to the functions of component
  - Provide functionality to the outside

Example

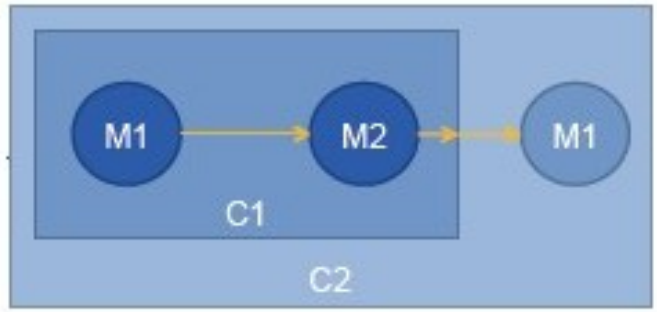

B provides functionality
Example:
-Turn on LED
-Send packet
I
A
B
A uses functionality provided by **B**
Example: To send a packet
Interface **I** describes the "API"

TinyOS Components

There are two types of components in TinyOS:

- **Modules** (M):
  - base executable code
  - contains variables / implementations
- **Configurations** (C):
  - contain multiple sub-components (modules or configurations): nesting
  - describe connections (e.g., which module uses the function)
  - parameterize sub-components: (e.g., options)

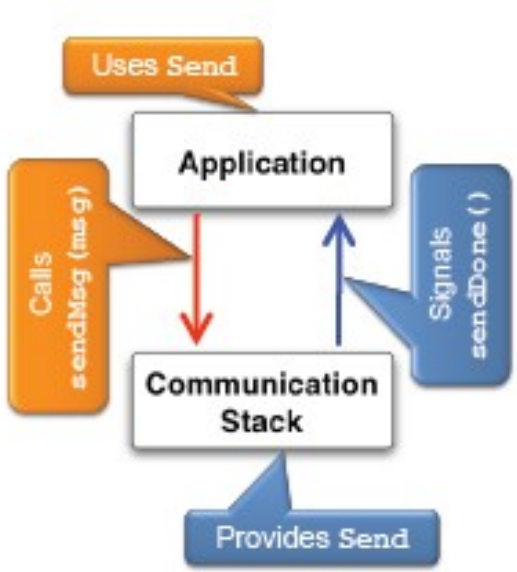
Components are very similar to classes (Java, C,...), statically linked via interfaces



TinyOS Components

- **Components** *use* or *provide* interfaces
  - component uses functionalities of others
  - provide functionalities to others
- **Interfaces** contain the API, expressed through *commands* and *events*

Function calls	**Commands**	**Events**
Use	Call Command	Implement Event Handler
Provide	Implement Command Body	Signal Event



TinyOS Modules

TinyOS modules are composed of

- **Specification**
  - List of interfaces that the component provides to others and uses from others
  - Alias interfaces as new name
- **Implementation**
  - Commands of provided interfaces
  - Events of used interfaces

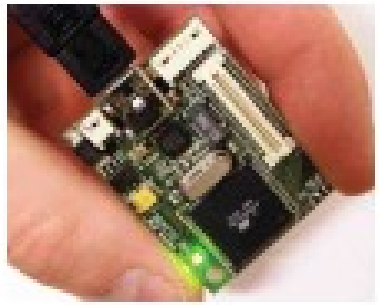
```
module FooM {
  //Specification
  provides {
    interface Foo;
  }
  uses {
    interface Poo as PooFoo;
    interface Boo;
  }
}
//Implementation
implementation {
  //Command handlers
  command result_t Foo.comm{
    ...
  }
  //Event handlers
  event void Boo.event{
    ...
   }
}
```

TinyOS and NesC

Example: Hello World

Example: "Hello World"

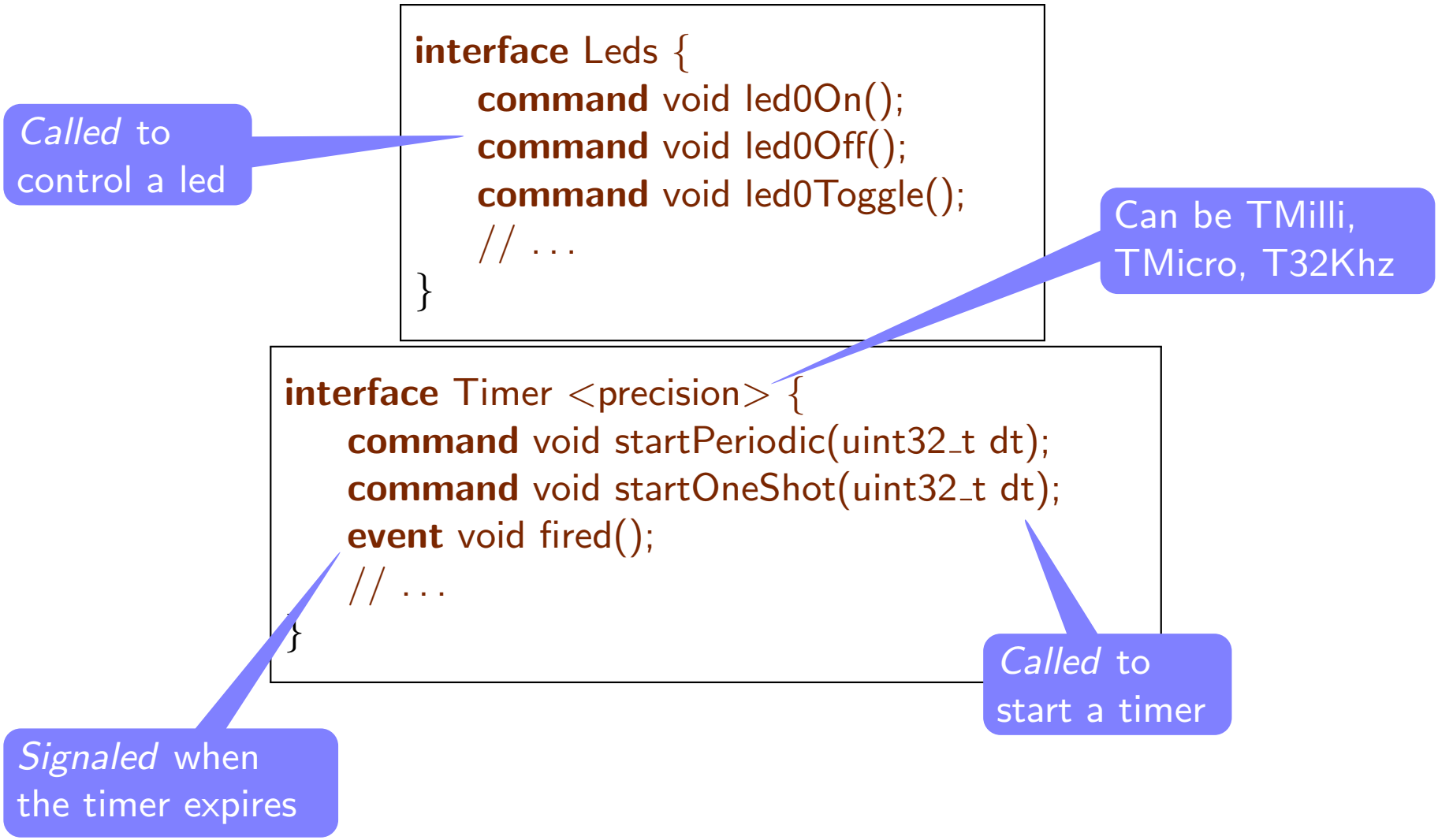
- No display, motes cannot print "Hello World"
- We let them periodically blink the LEDs!



Ingredients:

- A timer to trigger the periodic blinking
  - TinyOS provides a *Timer* library
- A way to turn the LEDs on and off
  - TinyOS provides a *LedsC* component to control the LEDs

"Hello World"


interface Leds {
command void led0On();
command void led0Off();
command void led0Toggle();
// ...
}
interface Timer <precision> {
command void startPeriodic(uint32_t dt);
command void startOneShot(uint32_t dt);
event void fired();
// ...
}
Called to control a led
Can be TMilli, TMicro, T32Khz
Called to start a timer
Signaled when the timer expires

"Hello World"

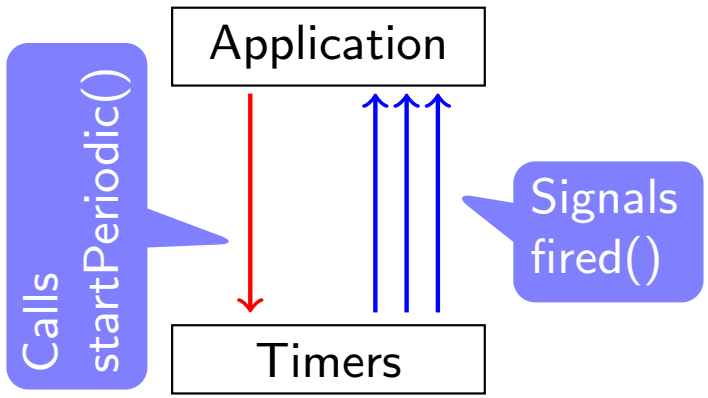
Timer with millisecond precision

```
module BlinkC {
    uses interface Timer<TMilli> as BlinkTimer;
    uses interface Leds;
    uses interface Boot;
}

implementation {
    event void Boot.booted() {
        call BlinkTimer.startPeriodic(1000);
    }

    event void BlinkTimer.fired() {
        call Leds.led0Toggle();
    }
}
```

Signaled when hw ready



"Hello World"

No interfaces used or provided

```
configuration BlinkAppC {}

implementation {
    components MainC,
               BlinkC,
               LedsC;
    components new TimerMilliC as Timer0;

    BlinkC -> MainC.Boot;
    BlinkC.BlinkTimer -> Timer0;
    BlinkC.Leds -> LedsC;
}
```

Provides Boot

TimerMilliC is a generic component: can be instantiated multiple times!

Wirings

- Instantiation is done at compile-time
  - very different from typical object-oriented programming

"Hello World"

Cross-compile for a specific WSN platform

```
$ make telosb
    // ...
    compiled BlinkAppC to build/telosb/main.exe
                  2648 bytes in ROM
                    54 bytes in RAM
    msp430-objcopy –output-target=ihex build/telosb/
    main.exe build/telosb/main.ihex
        writing TOS image
$ make telosb install, 0 bsl,/dev/ttyUSB0
    // ...
```

Node id

USB port the node is attached to

- Use "motelist" command to determine USB port

Programming (cont.)

TinyOS and NesC

Split-phase Operation

- TinyOS has a single stack and non blocking operation
- When a program calls a long-running operation, the call returns immediately, and a callback is issued when the operation is complete
- Invocation and completion are split into two separate phases of execution
- The approach guarantees memory efficient concurrency

```
event void Boot.booted() {
    call BlinkTimer.startPeriodic(1000);
}
event void BlinkTimer.fired() {
    call Leds.led0Toggle();
}
```



```
void main() {
    while() {
        sleep(1);
        printf('hello');
    }
}
```

Tasks

- **Tasks** are functions that can be posted into the task queue (FIFO)
  - they run until completion and do not interrupt each other
  - long tasks can reduce responsiveness, keep them short!
- **Interrupts** low level hardware events that run on stack
  - interrupt can post and interrupt tasks
  - interrupts can interrupt each other

```
// ...
task void compute () {
    // do something;
}

event void Timer.fired
()
{
    post compute()
}
// ...
```

TOSSIM

- TOSSIM is a simulator for TinyOS applications
- Allows NesC code to run on a standard machine (Linux etc.)
- Includes models for
  - Communication
  - Sensing
  - ...
- It works by replacing components with simulation implementations
  - Run the same code in simulator and sensor node
- Good for testing before deployment

Code examples

- TinyOS has an "apps" folder
  - Many good examples
- Other sources of information
  - TinyOS Book
    - TinyOS Programming by Philip Levis, David Gay
    - In part: csl.stanford.edu/∼ pal/pubs/tinyos-programming.pdf
  - TinyOS web book
    - csl.stanford.edu/∼ pal/pubs/tos-programming-web.pdf
  - TinyOS tutorial
    - IPSN 2009 (enl.usc.edu/talks/cache/tinyos-ipsn2009.ppt)
  - TinyOS website: http://www.tinyos.net/
  - TinyOS mailing list (and its archives)
    - On the TinyOS website

Programming (cont.)

Hands-On

Today's Task

- Goal: program an anti-theft device
  - Detect when someone steels your sensor node
- Two parts:
  - Detecting theft
    - Assume: thieves put the motes in their pockets
    - So, a "dark" mote is a stolen mote
    - Every N ms check if light sensor is below some threshold
  - Reporting theft
    - Assume: bright flashing lights deter thieves
    - Theft reporting algorithm: light the red LED for a little while!
- What you will use
  - Basic components, interfaces, wiring
  - Essential system interfaces for startup, timing, sensor sampling
- Start from Blink
  - In apps/Blink
  - See apps/RadioSensoToLeds on sensor use
    - Replace "DemoSensorC" with "HamamatsuS10871ParC" or "HamamatsuS10871TsrC"

Anti-Theft Module

```
module AntiTheftC {
    uses interface Boot;
    uses interface Timer<TMilli> as Check;
    uses interface Read<uint16_t>;
}
implementation {
    event void Boot.booted() {
        call Check.startPeriodic(1000);
    }
    event void Check.fired() {
        call Read.read();
    }
    event void Read.readDone(error_t ok, uint16_t val) {
        if (ok == SUCCESS && val < 200)
            theftLed();
    }
}
```

```
interface Boot {
    /* Signaled when OS booted */
    event void booted();
}
```

```
interface Timer< tag> {
    command void startOneShot(uint32_t period)
    command void startPeriodic(uint32_t period)
    event void fired();
}
```

Anti-Theft Module

```
module AntiTheftC {
    uses interface Boot;
    uses interface Timer<TMilli> as Check;
    uses interface Read<uint16_t>;
}
implementation {
    event void Boot.booted() {
        call Check.startPeriodic(1000);
    }
    event void Check.fired() {
        call Read.read();
    }
    event void Read.readDone(error_t ok, uint16_t val) {
        if (ok == SUCCESS && val < 200)
            theftLed();
    }
}
```

Split-phase example:
- *Read.read* starts the operation
- *Read.readDone* signals completion

```
interface Read< val_t > {
    command error_t read();
    event void readDone(error_t ok, val_t val);
}
```

Anti-Theft Configuration

```
configuration AntiTheftAppC
implementation
{
    components AntiTheftC, MainC, LedsC;

    AntiTheftC.Boot -> MainC.Boot;
    AntiTheftC.Leds -> LedsC;

    components new TimerMilliC() as MyTimer;
    AntiTheftC.Check -> MyTimer;

    components new HamamatsuS10871ParC();
    AntiTheftC.Read -> HamamatsuS10871ParC;
}
```

```
generic configuration TimerMilliC() {
    provides interface Timer<TMilli>;
}
implementation { ... }
```

```
generic configuration PhotoC() {
    provides interface Read;
}
implementation { ... }
```

Your Tasks

1. System test
   - Install Blink on your sensor node
     - See apps/Blink
     - Compile, install (see prev. slides, see tutorial from TA)
2. Main task:
   - Program Anti-Theft App for your sensor node

Thanks!

Questions?

- These slides are based on material from
  - Olaf Landsiedel, Phil Levis, David Gay, David Culler, Luca Mottola, Hamad Alizai, and many others