25 Years of Development. . .

25 Years of Development. . .

Outline

1. Wireless Sensor Networks
   - Definition and Applications
   - Hardware (Processors, Boards, Radios, ...)
   - Constraints and Challenges
2. NesC and TinyOS
   - NesC Language Overview
   - TinyOS: Operating System for WSNs
   - Demonstration
3. Internet and Wireless Sensor Networks
   - Translating Gateway/Proxy
   - uIP, 6lowpan
   - Demonstration

Outline

1. Wireless Sensor Networks
   - Definition and Applications
   - Hardware (Processors, Boards, Radios, ...)
   - Constraints and Challenges
2. NesC and TinyOS
   - NesC Language Overview
   - TinyOS: Operating System for WSNs
   - Demonstration
3. Internet and Wireless Sensor Networks
   - Translating Gateway/Proxy
   - uIP, 6lowpan
   - Demonstration

Wireless Sensor Networks

Definition

A wireless sensor network (WSN) is a **wireless network** consisting of **spatially distributed autonomous devices** using **sensors** to **cooperatively monitor** physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants.

- Small computers with a wireless interface
- Smart alternatives to dumb RFID tags

Wireless Sensor Networks

Definition

A wireless sensor network (WSN) is a **wireless network** consisting of **spatially distributed autonomous devices** using **sensors** to **cooperatively monitor** physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants.

- Small computers with a wireless interface
- Smart alternatives to dumb RFID tags

Applications

- Environmental monitoring
- Seismic detection
- Disaster situation monitoring and recovery
- Health and medical monitoring
- Inventory tracking and logistics
- Smart spaces (home/office scenarios)
- Military surveillance

Processors — Atmel / TI / Intel

Atmel AVR ATmega 128

- 8 bit RISC at XX MHz, 32 registers
- 4kB RAM, 128kB Flash, 4kB EEPROM

TI MSP430

- 16 bit RISC at 8 MHz, 16 registers
- 10kB RAM, 48kB Flash, 16kB EEPROM

Intel PXA271 XScale

- 32 bit RISC at 13-416MHz, 16 registers
- 256kB SRAM, 32MB SDRAM, 32MB Flash

Processors — Atmel / TI / Intel

Atmel AVR ATmega 128

- 8 bit RISC at XX MHz, 32 registers
- 4kB RAM, 128kB Flash, 4kB EEPROM

TI MSP430

- 16 bit RISC at 8 MHz, 16 registers
- 10kB RAM, 48kB Flash, 16kB EEPROM

Intel PXA271 XScale

- 32 bit RISC at 13-416MHz, 16 registers
- 256kB SRAM, 32MB SDRAM, 32MB Flash

Processors — Atmel / TI / Intel

Atmel AVR ATmega 128

- 8 bit RISC at XX MHz, 32 registers
- 4kB RAM, 128kB Flash, 4kB EEPROM

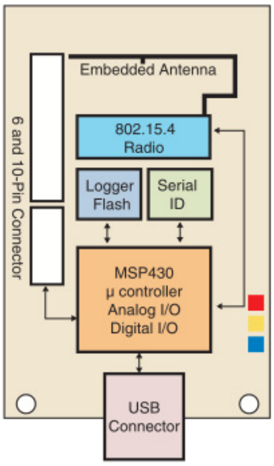
TI MSP430

- 16 bit RISC at 8 MHz, 16 registers
- 10kB RAM, 48kB Flash, 16kB EEPROM

Intel PXA271 XScale

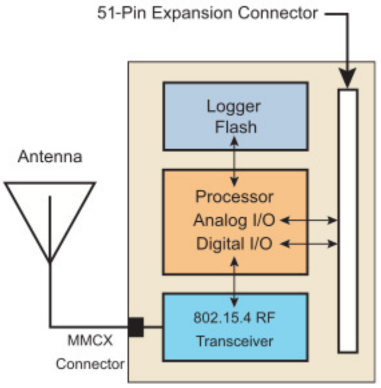
- 32 bit RISC at 13-416MHz, 16 registers
- 256kB SRAM, 32MB SDRAM, 32MB Flash

Boards — Telos-B



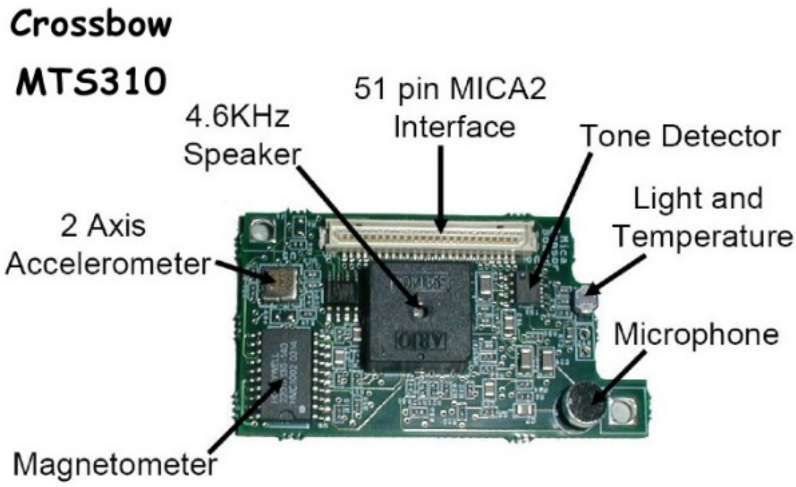
TPR2400CA Block Diagram

Boards — Mica-Z

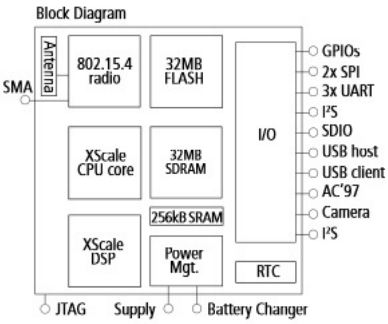


MPR2400CA Block Diagram

Sensors — Mica Sensor Board MTS310



Boards — Imote2



Power Consumption

mote	processor	voltage	active	sleep
Telos-B	IT MSP430	1.8V min	1.8 mA	5.1 uA
Mica-Z	Atmel AVR	2.5V min	8 mA	$<$ 15 uA
Imote2	Intel PXA271	1.3V min	44-66 mA	390 uA

- Imote2 is computationally powerful enough to run an embedded Linux kernel.
- Imote2 requires a relatively decent power supply (or a short usage period).
- Xscale sold to Marvell Technologies in 2006

Radio — IEEE 802.15.4

IEEE 802.15.4 (Zigbee)

- 250 kbps (16 channels, 2.4 GHz ISM band)
- personal area networks (few meters range)
- PHY and MAC layer covered
- Link encryption (AES) (no key management)
- Full / Reduced function devices

ChipCon CC2420

- popular 802.15.4 air interface
- 128byte TX/RX buffer

Radio — IEEE 802.15.4

IEEE 802.15.4 (Zigbee)

- 250 kbps (16 channels, 2.4 GHz ISM band)
- personal area networks (few meters range)
- PHY and MAC layer covered
- Link encryption (AES) (no key management)
- Full / Reduced function devices

ChipCon CC2420

- popular 802.15.4 air interface
- 128byte TX/RX buffer

Design Goals

- cheap
  - ideally less than 1 Euro
- many
  - lots of devices, economies of scale
- robust
  - unattended operation (no repair)
- small
  - importance depends on the circumstances
- low-power
  - difficult/impossible to replace batteries

Design Goals

- cheap
  - ideally less than 1 Euro
- many
  - lots of devices, economies of scale
- robust
  - unattended operation (no repair)
- small
  - importance depends on the circumstances
- low-power
  - difficult/impossible to replace batteries

Design Goals

- cheap
  - ideally less than 1 Euro
- many
  - lots of devices, economies of scale
- robust
  - unattended operation (no repair)
- small
  - importance depends on the circumstances
- low-power
  - difficult/impossible to replace batteries

Design Goals

- cheap
  - ideally less than 1 Euro
- many
  - lots of devices, economies of scale
- robust
  - unattended operation (no repair)
- small
  - importance depends on the circumstances
- low-power
  - difficult/impossible to replace batteries

Design Goals

- cheap
  - ideally less than 1 Euro
- many
  - lots of devices, economies of scale
- robust
  - unattended operation (no repair)
- small
  - importance depends on the circumstances
- low-power
  - difficult/impossible to replace batteries

Research Topics

- **Embedded systems and languages**
- Energy-aware resource management
- Cross-layer design and optimization
- (Ad-hoc) mesh routing protocols
- **Internetworking**
- Middleware for wireless sensor networks
- Localization, time synchronization, ...
- Data fusion, control, actuation, ...
- Security and Applications

Outline

1. Wireless Sensor Networks
   - Definition and Applications
   - Hardware (Processors, Boards, Radios, ...)
   - Constraints and Challenges
2. NesC and TinyOS
   - NesC Language Overview
   - TinyOS: Operating System for WSNs
   - Demonstration
3. Internet and Wireless Sensor Networks
   - Translating Gateway/Proxy
   - uIP, 6lowpan
   - Demonstration

NesC and TinyOS History

- developed by a consortium led by UC Berkeley
- two versions
  - TinyOS 1.1
  - TinyOS 2.0
- 2.0 not backwards compatible with 1.1

NesC: Programming Language for Embedded Systems

- Programming language:
  - a dialect/extension of C
  - static memory allocation only (no malloc/free)
  - whole-program analysis, efficient optimization
  - race condition detection
- Implementation:
  - pre-processor – output is a C-program, that is compiled using gcc for the specific platform
  - statically linking functions
- For more details, see [3]

NesC — Interfaces

- *commands*
  - can be called by other modules
  - think functions
- *events*
  - signalled by other modules
  - have to be handled by this module

Interface Example

```
interface Send {
  command error_t send(message_t* msg, uint8_t len);
  event void sendDone(message_t* msg, error_t error);

}
```

NesC — Components

- a NesC application consists of *components*
- components provide and use *interfaces*
- components can be accessed only via interfaces (cannot call an arbitrary C-function from another module)

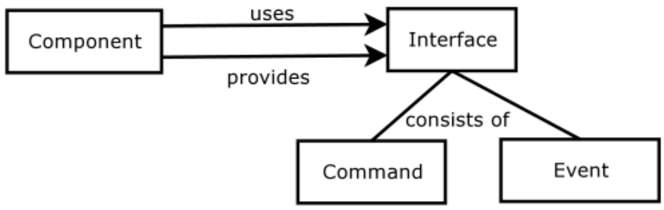


Figure: NesC Interface

NesC — Components

- *modules* – implement interfaces
- *configurations* – connect modules together via their interfaces (*wiring*)

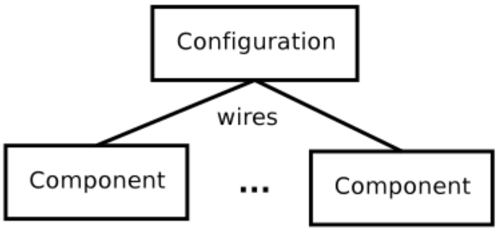


Figure: NesC Configuration

NesC — Concurrency — Tasks

Define a Task

```
task void task_name() {      }
```

Post a Task

```
post task_name()
```

- posting a task – the task is placed on an internal task queue which is processed in FIFO order
- a task runs to completion before the next task is run, i.e. tasks do not preempt each other
- tasks can be preempted by hardware events

TinyOS

- written in nesC
- event-driven architecture
- no kernel/user space differentiation
- single shared stack
- no process or memory management
- no virtual memory
- multi-layer abstractions
- components statically linked together

TinyOS — Functionality

- hardware abstraction
- access to sensors
- access to actuators
- scheduler (tasks, hardware interrupts)
- timer
- radio interface
- Active Messages (networking)
- storage (using flash memory on the motes)
- ...

TinyOS — Demo — Blink

- no screen on which we could print

  '*Hello World*'

- let's blink an led instead
- using a timer to blink an led
- 2 source files
  - BlinkC.nc
  - BlinkAppC.nc

BlinkC.nc

```
module BlinkC
{
    uses interface Timer<TMilli> as Timer0;
    uses interface Leds;
    uses interface Boot;
}
implementation
{
    event void Boot.booted()
    {
        call Timer0.startPeriodic(250);
    }
    event void Timer0.fired()
    {
        call Leds.led0Toggle();
    }
}
```

BlinkAppC.nc

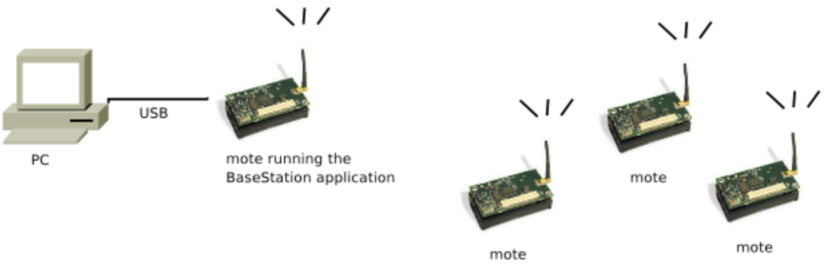
```
configuration BlinkAppC
{
}
implementation
{
    components MainC, BlinkC, LedsC;
    components new TimerMilliC() as Timer0;

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

TinyOS — Demo — Blink

- in reality using 3 timers
  - 250 ms
  - 500 ms
  - 1000 ms
- each timer toggling one led
- the result is a 3-bit counter

TinyOS — Demo — Oscilloscope

- motes – periodically get a sensor reading and broadcast over the radio
- BaseStation mote – forward packets between the radio and the serial interface
- PC - java application reading packets from the serial interface and plotting sensor readings



TinyOS — Demo — Oscilloscope

node ID	sensor
7	light
8	light
18	temperature

- temperature $= -38.4 + 0.0098 * \text{data}$

Outline

1. Wireless Sensor Networks
   - Definition and Applications
   - Hardware (Processors, Boards, Radios, ...)
   - Constraints and Challenges
2. NesC and TinyOS
   - NesC Language Overview
   - TinyOS: Operating System for WSNs
   - Demonstration
3. Internet and Wireless Sensor Networks
   - Translating Gateway/Proxy
   - uIP, 6lowpan
   - Demonstration

Why connect WSNs via the Internet?

- Internet: IP
  - ubiquitous
  - de-facto standard
  - already deployed
  - plethora of applications available
- TinyOS' notion of networking
  - Active Messages

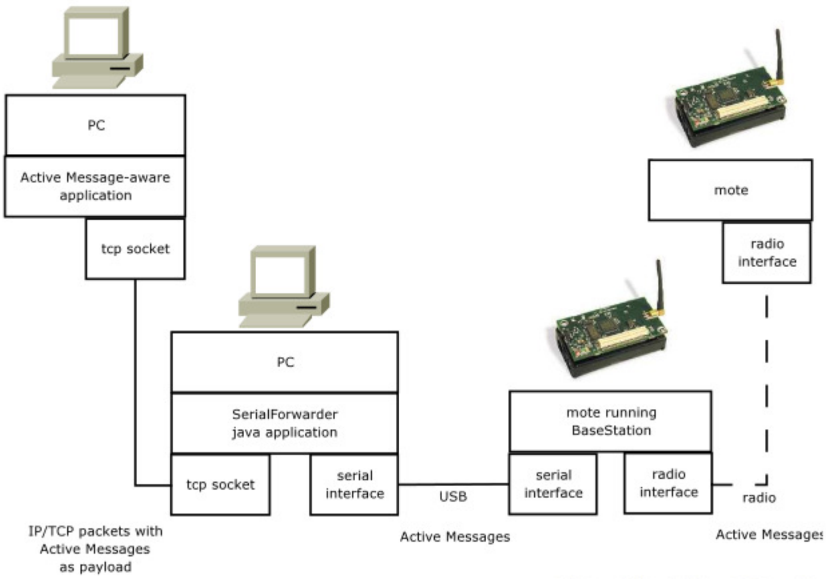
Connecting WSNs to the Internet

- translating using gateway/proxy (motes use Active Messages)
  - Serial Forwarder
  - Sensor Internet Protocol
- make motes IP-aware
  - uIP
  - 6lowpan

Serial Forwarder

- Active Messages tunneled inside TCP to a gateway
- gateway: PC attached to a BaseStation mote
- BaseStation mote – forwarding messages between the radio and the serial interface (BaseStation application)
- drawback: application on the PC has to be Active Message-aware

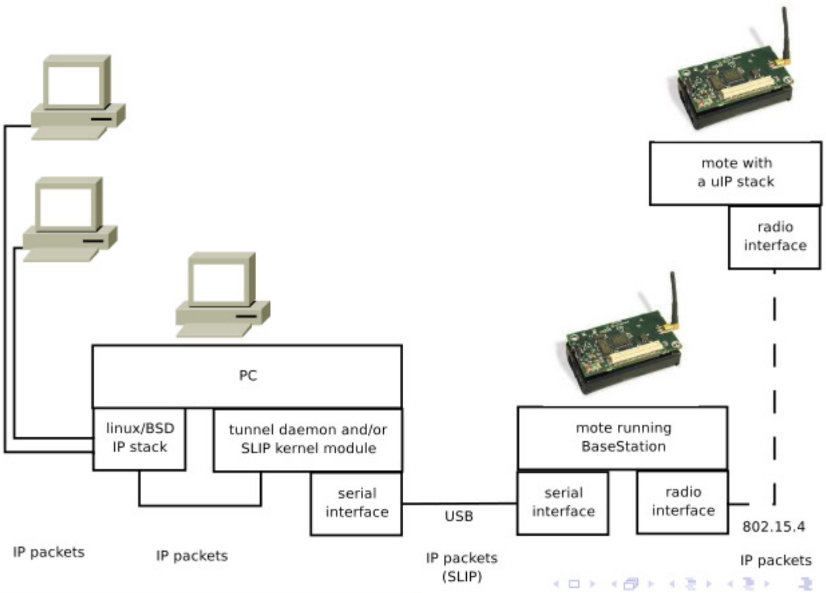
Serial Forwarder



uIP

- TCP/IPv4 stack implementation by Adam Dunkels (KTH)
- very small code size and memory footprint
- written in C
- using gotos, global variables and few functions for efficiency
- ported to TinyOS 1.x by Andrew Christian from Hewlett-Packard

uIP


mote with
a uIP stack
radio
interface
PC
linux/BSD
IP stack
tunnel daemon and/or
SLIP kernel module
serial
interface
USB
mote running
BaseStation
serial
interface
radio
interface
802.15.4
IP packets
IP packets
IP packets
(SLIP)
IP packets

6lowpan — IPv6 over 802.15.4

- IETF working group (IPv6 over low-power wireless personal area networks)
- 6lowpan header/dispatch value before the IP header

layer 2 header (802.15.4)
optional mesh addressing header (6lowpan)
optional broadcast header (6lowpan)
optional fragmentation header (6lowpan)
IPv6 header (6lowpan-compressed)
layer 4 header (i.e. 6lowpan compressed UDP header)
layer 4/application payload

Table: 802.15.4 frame with 6lowpan payload

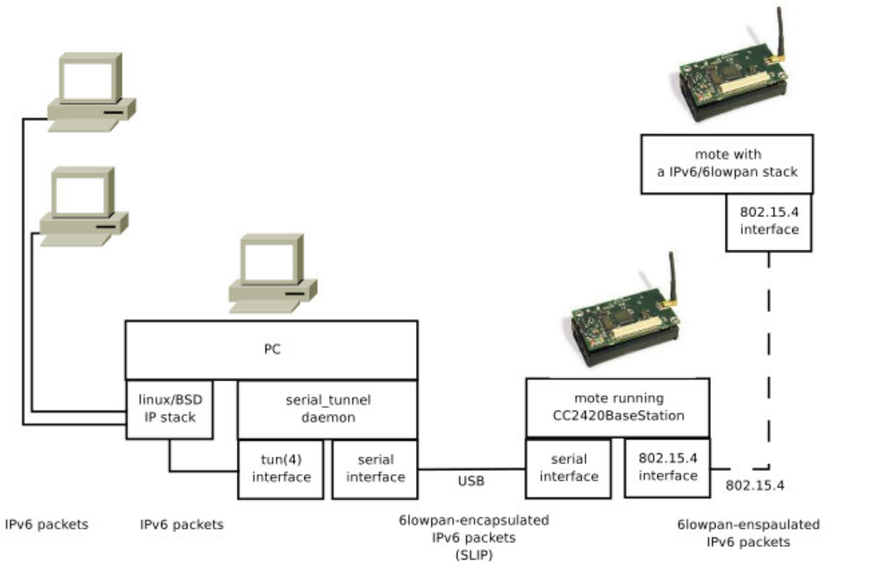
6lowpan — Details

- header compression
  - IPv6 and UDP headers can ideally be compressed from $40 + 8$ to $2 + 4$ bytes
  - no prior communication for context estabilishment necessary
- fragmentation below the IP layer
  - IPv6 requires a minimum MTU of 1280 bytes, but 802.15.4 can at best provide 102 bytes
- mesh networking support
  - routing algorithms and further details out of scope of the 6lowpan working group

6lowpan — Jacobs University

- motes – IP-aware and communicate via radio with the BaseStation mote
- BaseStation mote – forwards packets between the radio and the serial interface
- PC – IP-aware, running `serial_tunnel` application for exchanging packets between the serial interface and the networking stack
- `serial_tunnel` is doing 6lowpan en-/decapsulation

6lowpan — Jacobs University



6lowpan — Challenges

- compression
- fragmentation
- efficiency
  - end-to-end retransmissions (i.e. TCP, caching on intermediate nodes)
- energy-consumption an issue
- ways to save energy
  - sleep (duty-cycling)
  - do not use the radio
  - minimize the amount of data sent over the radio

6lowpan — Demonstration

- Ping (IPv6)
- cli (telnet over IPv6/UDP)

References

K. Römer and F. Mattern.
The Design Space of Wireless Sensor Networks
IEEE Wireless Communications 11(6), December 2004.

J. Polastre, R. Szewczyk and David Culler.
Telos: Enabling Ultra-Low Power Wireless Research
IEEE IPSN, April 2005.

D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler.
The nesC Language: A Holistic Approach to Networked Embedded Systems
ACM PLDI, June 2003.

A. Dunkels.
Full TCP/IP for 8-Bit Architectures
ACM MOBISYS 2003, May 2003.

G. Montenegro, N. Kushalnagar, J. Hui and D. Culler.
Transmission of IPv6 Packets over IEEE 802.14.4 Networks
Internet-Draft draft-ietf-6lowpan-format-13 (work in progress), April 2007.

Questions?