Wireless Sensor Network Programming Using TinyOS

A Tutorial

Wenjie Zeng

Oct 2011

Typical WSN Architecture

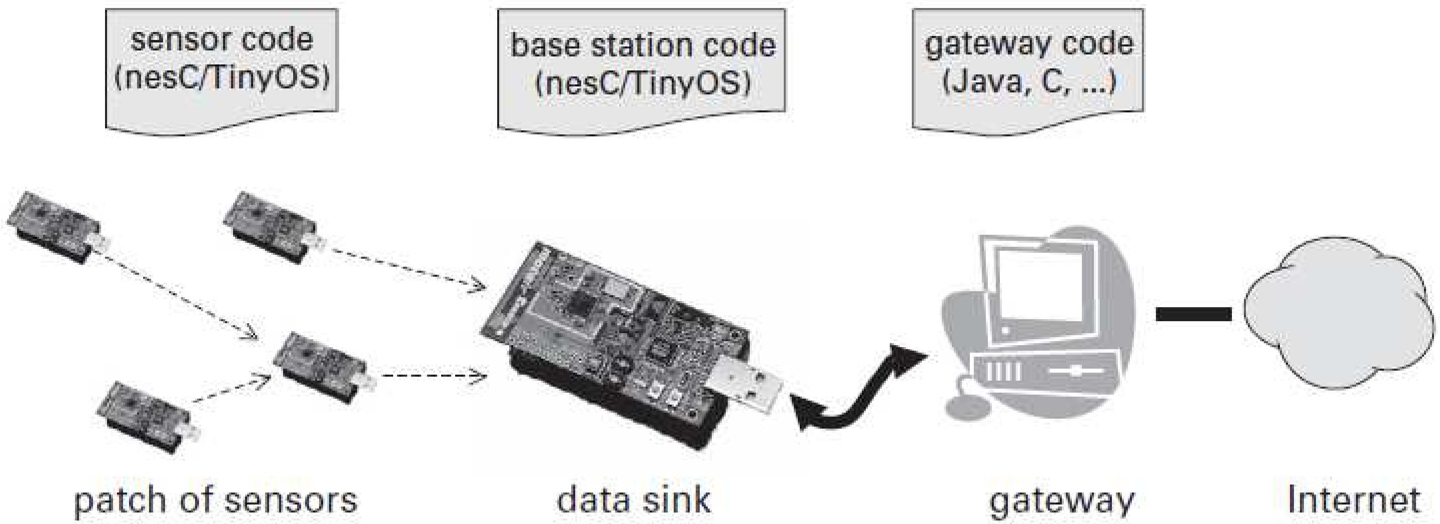
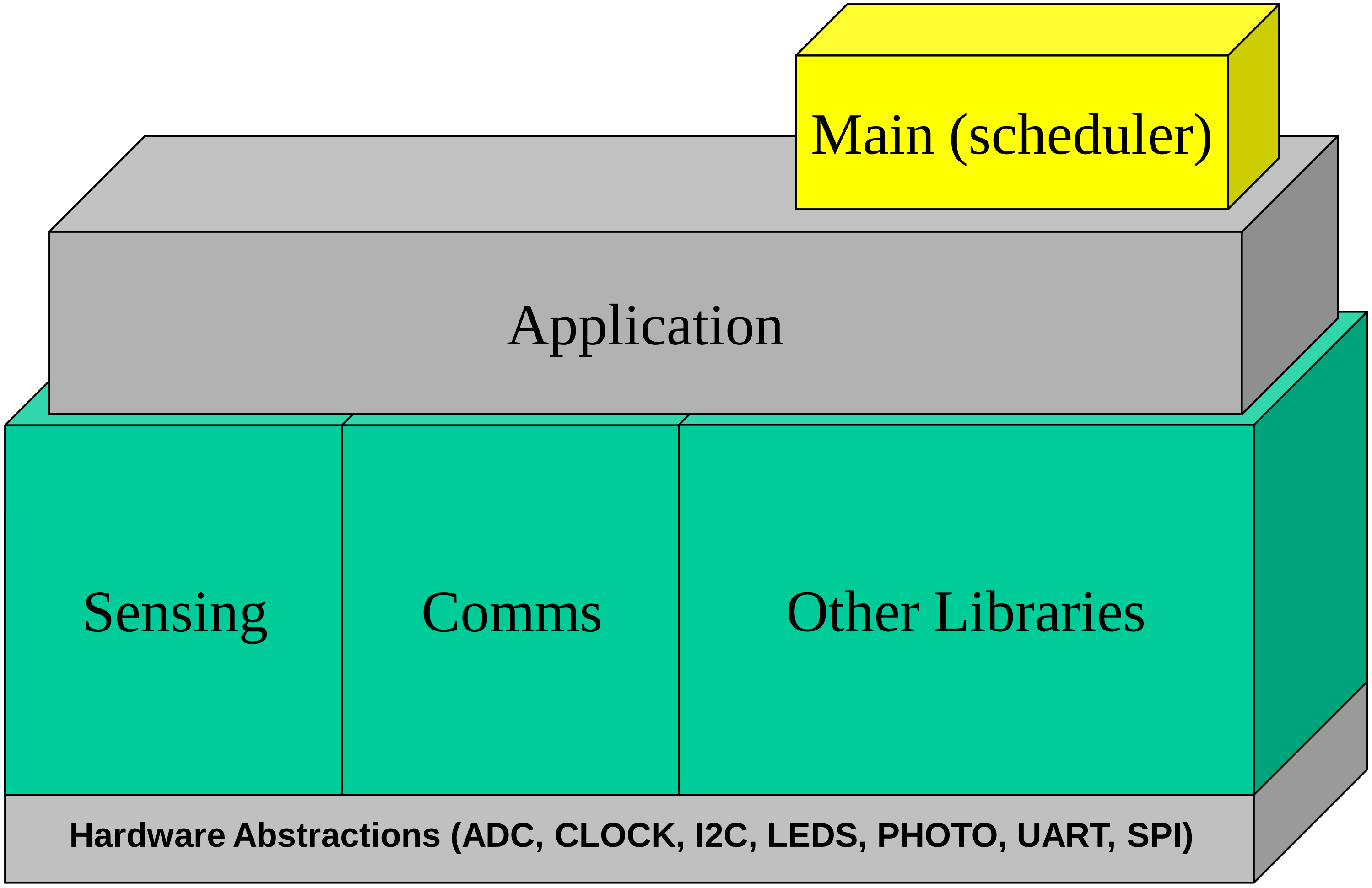


Figure 1.1 A typical sensor network architecture. Patches of ultra-low power sensors, running nesC/TinyOS, communicate to gateway nodes through data sinks. These gateways connect to the larger Internet.

TinyOS Architecture



Compilation

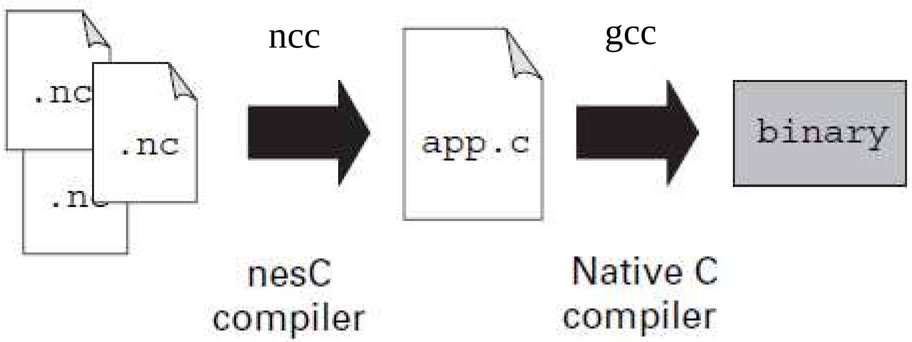


Figure 2.2 The nesC compilation model. The nesC compiler loads and reads in nesC components, which it compiles to a C file. This C file is passed to a native C compiler, which generates a mote binary.

Outline

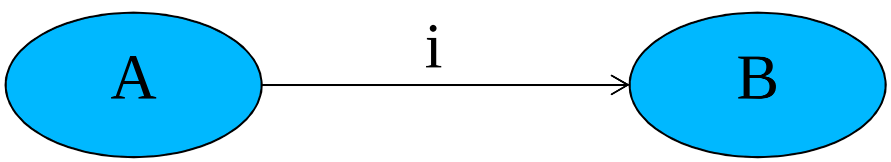
- Components and interfaces
  - Basic example
- Tasks and concurrency
- TinyOS communications
- Compilation and toolchain

Outline

- Components and interfaces
  - Basic example
- Tasks and concurrency
- TinyOS communications
- Compilation and toolchain

Components and Interfaces

- Basic unit of *nesC* code is component
- Components connect via interfaces
  - Connections called “wiring”

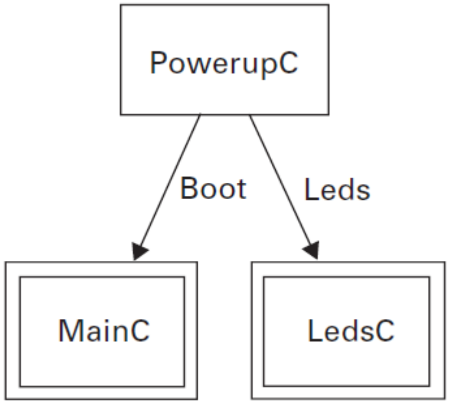


Components

- A component is a file that ends with *.nc*
  - Names must match
- *Modules* are components with variables and executable codes
- *Configurations* are components that wire other components together

Components

```
module PowerupC {
  uses interface Boot;
  uses interface Leds;
}
implementation {
  event void Boot.booted() {
   call Leds.led0On();
  }
}
```



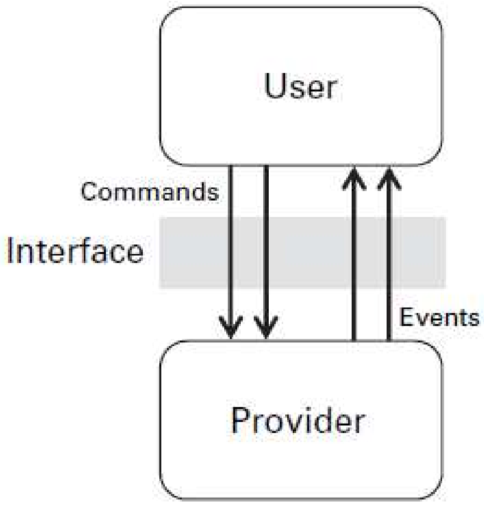

```
configuration PowerupAppC { }
implementation {
  components MainC, LedsC, PowerupC;

  MainC.Boot -> PowerupC.Boot;
  PowerupC.Leds -> LedsC.Leds;
}
```

Wiring: Pick implementations for used interfaces

Listing 2.4 PowerupAppC configuration in nesC

Interfaces

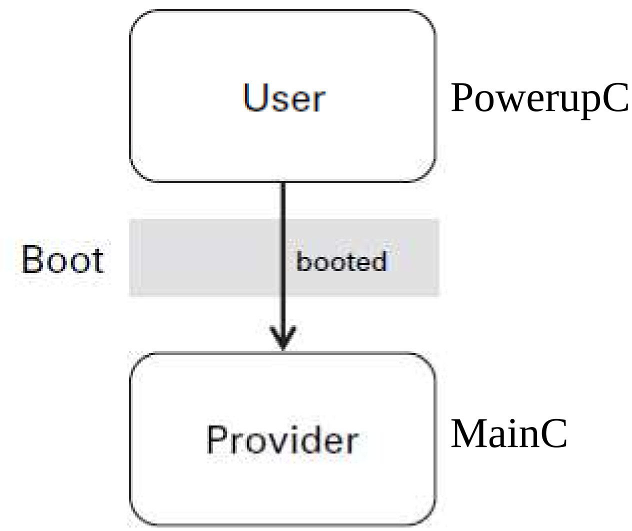
- Collections of related functions
- Define interactions between components
- Interfaces are bidirectional
  - **Commands**
    - Implemented by provider
    - Called by user
  - **Events**
    - Called (signaled) by provider
    - Implemented (captured) by user
- Can have parameters (types)



Who is the provider for the *Boot* interface?

```
module PowerupC {
  uses interface Boot;
  uses interface Leds;
}
implementation {
  event void Boot.booted() {
   call Leds.led0On();
  }
}
```

```
configuration PowerupAppC { }
implementation {
  components MainC, LedsC, PowerupC;

  MainC.Boot -> PowerupC.Boot;
  PowerupC.Leds -> LedsC.Leds;
}
```



Listing 2.4 PowerupAppC configuration in nesC

Interfaces

- Can have parameters (types)

```
interface Queue<t> {
  command bool empty();
  command uint8_t size();
  command uint8_t maxSize();
  command t head();
  command t dequeue();
  command error_t enqueue(t newVal);
  command t element(uint8_t idx);
}
```

```
module QueueUserC {
  uses interface Queue<uint32_t>;
}
```

```
module Queue32C {
provides interface Queue<uint32_t>;
}
```

Outline

- Components and interfaces
  - Basic example
- Tasks and concurrency
- TinyOS communications
- Compilation and toolchain

Basic example

- Goal: an anti-theft program that protects your bike!
- Two parts
  - Detecting theft
    - Assume: thieves will ride the stolen bike
    - A covered (dark) seat -> a stolen bike
    - Mote embedded in seat senses light every *500* ms
  - Reporting theft
    - Beep the pants out of the thief
- What we will use
  - Components, interfaces, and wiring configurations
  - Essential system interfaces for startup, timing, and sensor sampling

The Anti-Theft module

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<Tmilli> as CheckTimer;
  uses interface Read<uint16_t>;
  uses interface Beep;
}
implementation {
  event void Boot.booted() {
    call CheckTimer.startPeriodic(500);
  }
  event void CheckTimer.fired() {
    call Read.read();
  }
  event void Read.readDone(error_t e, uint16_t val) {
    if (e == SUCCESS && val < 200) {
      call Beep.beep();
    }
  }
}
```

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

The Anti-Theft module: split-phase operations

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<Tmilli> as CheckTimer;
  uses interface Read<uint16_t>;
  uses interface Beep;
}
implementation {
  event void Boot.booted() {
    call CheckTimer.startPeriodic(500);
  }
  event void CheckTimer.fired() {
    call Read.read();
  }
  event void Read.readDone(error_t e, uint16_t val) {
    if (e == SUCCESS && val < 200) {
      call Beep.beep();
    }
  }
}
```

In TinyOS, all long-running operations are *split-phase*:
- A command starts the operatin: read
    - Only one outstanding request allowed
- An event signals the completion of the operation: readDone

The Anti-Theft module: split-phase operations

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<Tmilli> as CheckTimer;
  uses interface Read<uint16_t>;
  uses interface Beep;
}
implementation {
  event void Boot.booted() {
    call CheckTimer.startPeriodic(500);
  }
  event void CheckTimer.fired() {
    call Read.read();
  }
  event void Read.readDone(error_t e, uint16_t val) {
    if (e == SUCCESS && val < 200) {
      call Beep.beep();
    }
  }
}
```

In TinyOS, all long-running operations are *split-phase*:

- A command starts the operation: read
    - Only one outstanding request allowed
- An event signals the completion of the operation: readDone
    - Errors are signalled by *error_t* variable

The Anti-Theft configurations

```
configuration AntiTheftAppC {}
implementation {
  components AntiTheftC, MainC, BeepC;

  AntiTheftC.Boot -> MainC;
  AntiTheftC.Beep -> BeepC;

  components new TimerMilliC() as TheTimer;
  AntiTheftC.CheckTimer -> TheTimer;

  components new PhotoC() as PhotoSensor;
  AntiTheftC.Read -> PhotoSensor;
}
```

A configuration is a component built with other components

- It wires the user of interfaces to providers
- It can instantiate generic components
- It can itself provide and use interfaces

The Anti-Theft configurations

```
configuration AntiTheftAppC {}
implementation {
  components AntiTheftC, Ma

  AntiTheftC.Boot -> MainC;
  AntiTheftC.Beep -> BeepC;

  components new TimerMilli
  AntiTheftC.CheckTimer ->

  components new PhotoC() as PhotoSensor;
  AntiTheftC.Read -> PhotoSensor;
}
```

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

```
generic configuration PhotoC() {
  provides interface Read<uint16_t>;
}
implementation {...}
```

A configuration is a component built with other components

- It wires the user of interfaces to providers
- It can instantiate generic components
- It can itself provide and use interfaces

Quick review

- TinyOS application is composed of components
  - Modules contains actual code
  - Configurations wire components together
- Components “wire” with one other through interfaces that can be parameterized
- Interfaces contain commands and events
- Provider of an interface implements the command body
- User of an interface implements the event body
- Long task are split-phase: read -> readDone

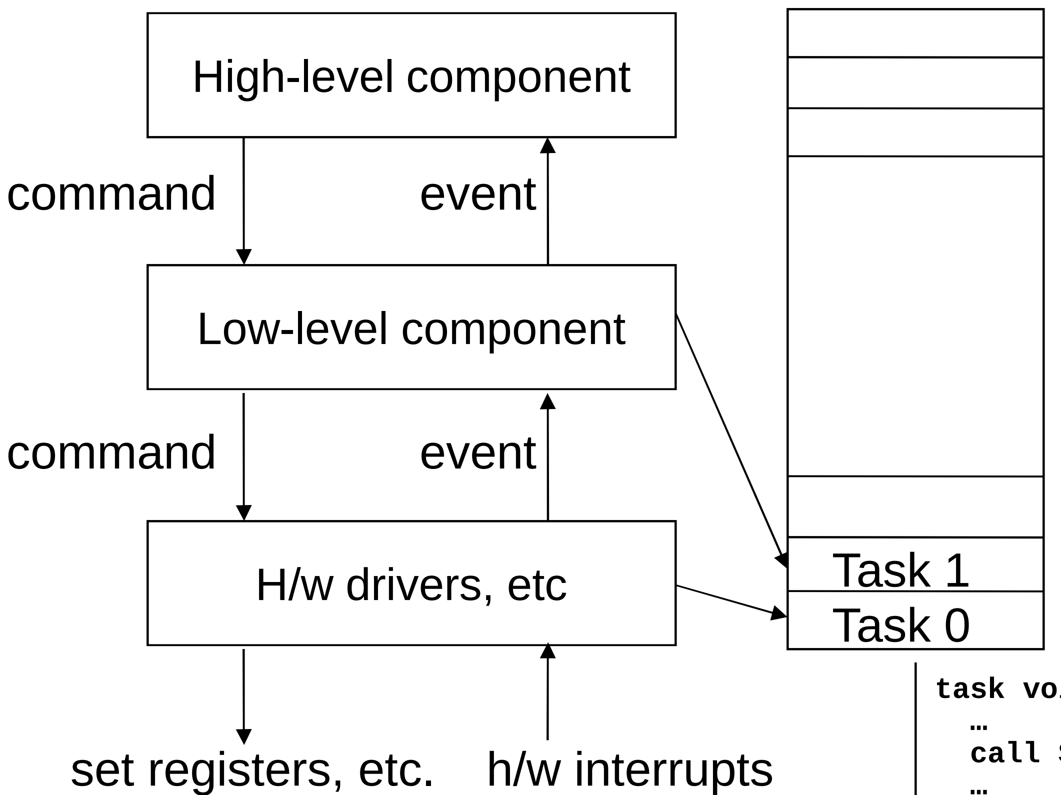
Outline

- Components and interfaces
  - Basic example
- Tasks and concurrency
- TinyOS communications
- Compilation and toolchain

Tasks

- TinyOS has one single thread, shared stack, no heap
  - code executes within commands, events (including interrupt handlers) and tasks
- Tasks: mechanism to defer computation
  - Tells TinyOS to “do this later”
- Tasks run to completion
  - TinyOS scheduler runs tasks in the order they are posted
  - Keep them short
- Interrupts can pre-empt tasks
  - The interrupt handler (function) will be invoked immediately after the interrupt
  - Race conditions
  - Interrupt handlers can post tasks

Commands, Events and Tasks



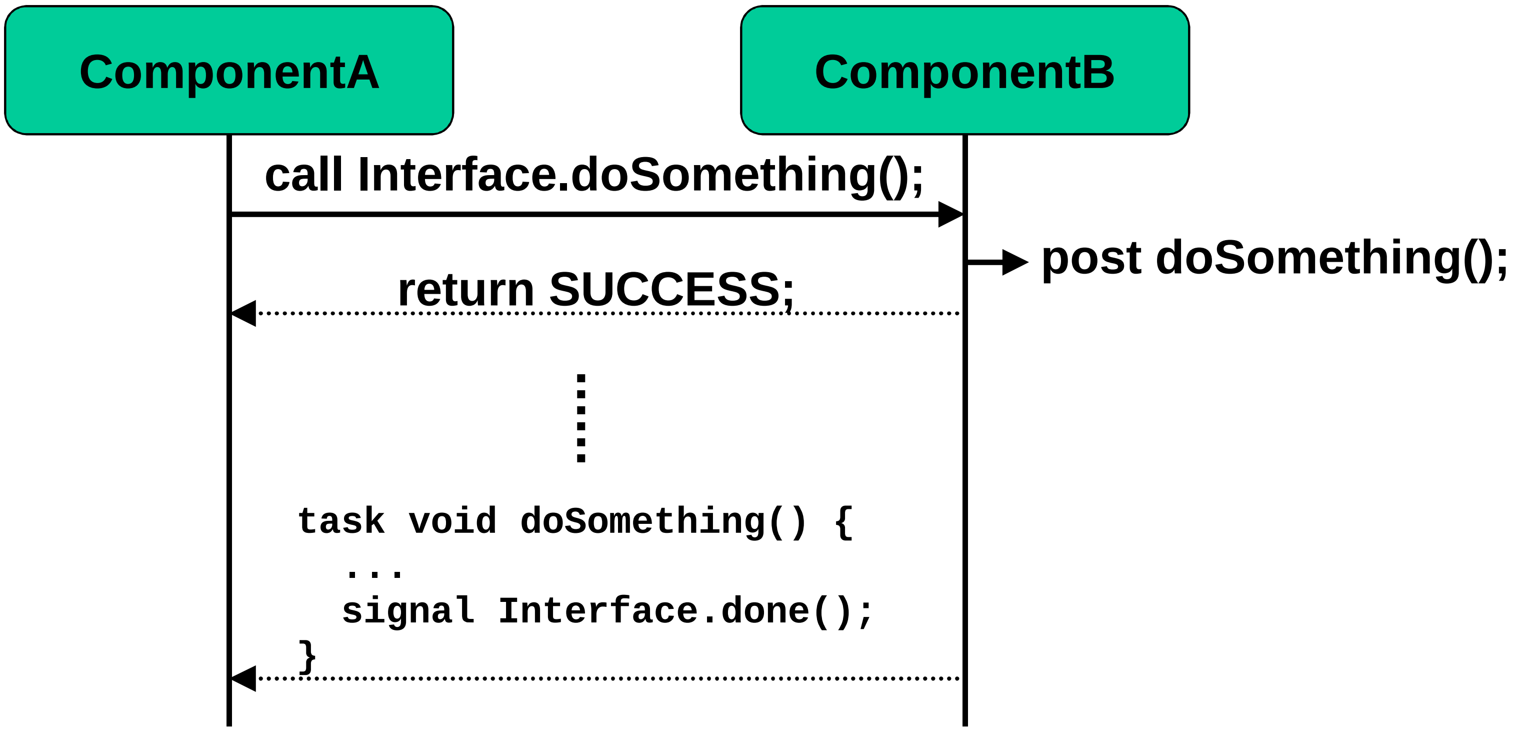
- tasks can call commands and signal events
- commands/events can post tasks or call other commands
- events are synchronous by default (no pre-emption)
- tasks pre-empted by asynchronous events but not other tasks

```
task void Task0 {
  …
  call SomeInterface.someCmd();
  …
}
```

```
configuration SomeComponent {
  provides interface SomeInterface;
}
implementation {…}
```

Task Scheduler

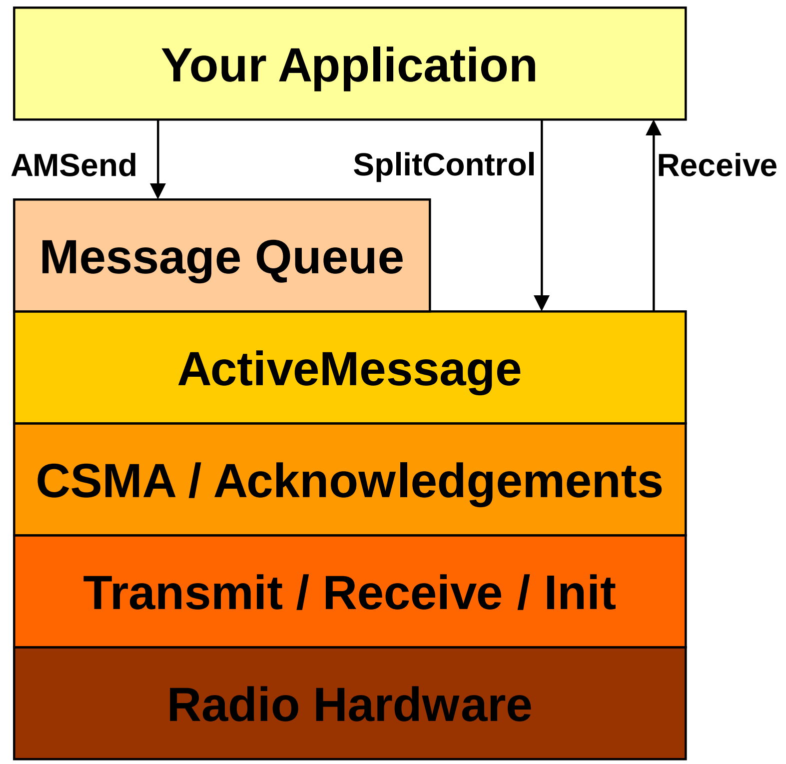
- Tasks result in *Split-Phase* execution



Outline

- Components and interfaces
  - Basic example
- Tasks and concurrency
- TinyOS communications
- Compilation and toolchain

Radio Stacks



Main *Radio* Interfaces

- **SplitControl**
  - Provided by **ActiveMessageC**
- **AMSend**
  - Provided by **AMSenderC**
- **Receive**
  - Provided by **AMReceiverC**

Main *Serial* Interfaces

- **SplitControl**
  - Provided by **SerialActiveMessageC**
- **AMSend**
  - Provided by **SerialAMSenderC**
- **Receive**
  - Provided by **SerialAMReceiverC**

Setting up the Radio: Configuration

```
configuration MyAppC {
}

implementation {
  components MyAppP,
    MainC,
    ActiveMessageC,
    new AMSenderC(0),  // send an AM type 0 message
    new AMReceiverC(0);  // receive an AM type 0 message

  MyAppP.Boot -> MainC;
  MyAppP.SplitControl -> ActiveMessageC;
  MyAppP.AMSend -> AMSenderC;
  MyAppP.Receiver -> AMReceiverC;
}
```

Setting up the Radio: Module

```
module MyAppP {
  uses {
    interface Boot;
    interface SplitControl;
    interface AMSend;
    interface Receive;
  }
}

implementation {
  …
}
```

Turn on the Radio

```
event void Boot.booted() {
  call SplitControl.start();
}

event void SplitControl.startDone(error_t error) {
  post sendMsg();
}

event void SplitControl.stopDone(error_t error) {
}
```

Send Messages

```
message_t myMsg;

task void sendMsg() {
  if(call AMSend.send(AM_BROADCAST_ADDR,
      &myMsg, 0) != SUCCESS) {
    post sendMsg();
  }
}

event void AMSend.sendDone(message_t *msg,
    error_t error) {
  post sendMsg();
}
```

Receive a Message

```
event message_t *Receive.receive(message_t *msg, void
    *payload, uint8_t length) {
  call Leds.led0Toggle();
  return msg;
}
```

Payloads

- A message consists of:
  - Header
  - Payload
  - Optional Footer

message_t

```
typedef nx_struct message_t {
  nx_uint8_t header[sizeof(message_header_t)];
  nx_uint8_t data[TOSH_DATA_LENGTH];
  nx_uint8_t footer[sizeof(message_footer_t)];

  nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

Payloads : Use Network Types

(MyPayload.h)

```
#ifndef MYPAYLOAD_H
#define MYPAYLOAD_H

typedef nx_struct MyPayload {
  nx_uint8_t count;
} MyPayload;

enum {
  AM_MYPAYLOAD = 0x50,
};

#endif
```

Example: Filling out a Payload

```
void createMsg() {
  MyPayload *payload = (MyPayload *) call AMSend.getPayload(&myMsg);
  payload->count = (myCount++);
  post sendMsg();
}
```

Example: Receiving a Payload

```
event void Receive.receive(message_t *msg, void *payload, uint8_t len)
   {
  MyPayload *payload = (MyPayload *) payload;
  call Leds.set(payload->count);
  signal RemoteCount.receivedCount(payload->count);
  return msg;
}
```

Radio layer tips

- How to set the channel using Makefile
  - PFLAGS = -DCC2420_DEF_CHANNEL=12
  - DEFINED_TOS_AM_GROUP: the motes group id (default is 0x22).
  - TOSH_DATA_LENGTH: radio packet payload length (default 28).
  - PFLAGS += "-DCC2420_DEF_RFPOWER=7“: sets the transmit power of the radio (0-31)
- How to change channel using the code
  - CC2420Control
- How do you get the signal strength of a received packet
  - CC2420Packet.getLqi(msg);

Common Gotchas

- TinyOS radio messages are default to 28 bytes
- Always use nx_ prefixed types (network types) in data structures to be sent
- Always check whether a command / event / task post is successful
  - Return value of a command
  - Argument of event carrying status
  - Return value of '*post taskName()*'

Timer Interface

- Timer
  - used to schedule periodic events like sensing
  - one-shot or repeat modes

```
uses interface Timer<TMilli> as Timer0;
call Timer0.startPeriodic( 250 );
call Timer0.startOneShot( 250 );
```

CC2420

- Supports hardware encryption using AES
- Implementation
  - Load 128-bit key to the CC2420 RAM and set a flag
    - The key is built with the binary or transfered using serial port
  - Loading the security RAM buffers on the CC2420 with the information to be encrypted (payload without header)
  - Microcontroller reads out of the security RAM buffer and concatenates the data with the unencrypted packet header.
  - This full packet would be uploaded again to the CC2420 TXFIFO buffer and transmitted.
- Source code and documentation
  - http://cis.sjtu.edu.cn/index.php/The_Standalone_AES_Encryption_of_CC2420_(TinyOS_2.10_and_MICAz)
- Hardware attack on TelosB mote to extract the AES Key
  - Takes advantage of the fact that the Key is loaded into the CC2420 chip, using a well know pin
    - http://travisgoodspeed.blogspot.com/2009/03/breaking-802154-aes128-by-syringe.html

Testing WSN Programs

- IDE: Eclipse + Yeti2 plug-in
  - http://tos-ide.ethz.ch/wiki/pmwiki.php?n=Site.Setup
- TOSSIM
- using actual hardware
  - LEDs – 3 of them so you can debug 8 states ☺
  - Printf library http://docs.tinyos.net/index.php/The_TinyOS_printf_Library
- Testbeds
  - Kansei
  - Peoplenet
  - GENI

Installation

Installing TinyOS 2.x

Read the installation tutorials on

http://docs.tinyos.net/index.php/Getting_started

- **VMPlayer (XubunTOS)**
  - **Download VMPlayer**
    - http://downloads.vmware.com/d/info/desktop_end_user_computing/vmware_player/4_0
  - **Download XubunTos image**
    - http://docs.tinyos.net/tinywiki/index.php/Running_a_XubunTOS_Virtual_Machine_Image_in_VMware_Player

Checking installation

```
$ cd $TOSROOT
$ cd apps/Blink
$ make telosb
$ cd build/telosb
$ ls
main.exe main.ihex tos_image.xml
$ export
```

$MAKERULES, $TOSROOT, $TOSDIR

Installing to a real mote

Connect your mote to the PC/Laptop

$ cd apps/Blink

Find out which port the mote is connected to

$ motelist

The mote id you set → The USB port the mote attached to

Compile and install:

$ make telosb *install,10* bsl,/dev/ttyUSB0

I want to *install* the program specified in the Makefile in the current directory into the *telosb* mote attached to */dev/tty/USB0* and set the id for this mote to *10*

Install an application you've previously compiled:

$ make telosb *reinstall,10* bsl,/dev/ttyUSB0

Getting help for a platform:

$ make telosb *help*

TOSSIM: TinyOS Simulator

- Provided as part of TinyOS package
- dbg statements to observe program state
- Easy to use for simple applications
- More detailed tutorial at http://docs.tinyos.net/index.php/TOSSIM

Debug Statements in TOSSIM

```
event void Boot.booted() {
  call Leds.led0On();
  dbg("Boot,RadioCountToLedsC", "Application
  booted.\n");
  call AMControl.start();
}


dbg("RadioCountToLedsC", "LQI: %d\n", rcvPkt->lqi);
```

Compiling TOSSIM

- Compiling for TOSSIM
  - $ cd $TOSROOT
  - $ cd apps/Blink
  - $ make micaz sim
- Running simulations
  - python blinkSim.py
  - http://www.cse.ohio-state.edu/~sridhara/Siefast/WSN_tutorial/TOSSIM

Sample Exercise-1

LinkQuality Measurement simulation

- Use TOSSIM to inject radio channel model and simulate the following application on 5 nodes
- Application specifications
  - Each node sends a periodic (once every 15 sec) broadcast Msg, with a sequence number.
  - Whenever it receives a message on radio, print the following using debug statements
    - Rcr_node, Src_node, Seq_no, Rssi, Lqi
  - Turn On Blue LED when you send the message and turn it off after you get the sendDone
  - Toggle Green LED whenever you receive a msg
  - You will need components
    - AMSenderC
    - AMReceiverC
    - CC2420Packet
    - Timer
    - Leds

Sample Exercise-2

LinkQuality Measurement on modes

- Program the application on real modes and collect the log files using the SerialForwarder application
- Application Specification
  - Each node sends a periodic (once every 15 sec) broadcast Msg, with a sequence number.
  - Whenever it receives a message on radio, write to UART
    - Rcr_node, Src_node, Seq_no, Rssi, Lqi
  - Turn On Blue LED when you send the message and turn it off after you get the sendDone
  - Toggle Green LED whenever you receive a msg
  - You will need components
    - AMSenderC
    - AMReceiverC
    - *SerialAMSenderC*
    - CC2420Packet
    - Timer
    - Leds

References

- To learn more
  - http://docs.tinyos.net

- Hardware vendors
  - Crossbow.com
  - Moteiv.com
  - Centila.com
  - Sunspots
  - Imote2

Acknowledgment

- TinyOS 2 tutorials at http://docs.tinyos.net/index.php/TinyOS_Tutorials
- David Moss. Rincon Research Corp
  - Some of the slides in this tutorial are taken from http://www.et.byu.edu/groups/ececmpsysweb/cmpsys.2008.winter/tinyos.ppt

Thank You!

- Questions ?

- Contact information

  - Jing Li – DL 281 (jingl@cse)
  - www.cse.ohio-state.edu/~sridhara/Siefast/WSN_tutorial

  - Wenjie Zeng – DL 283
  - zengw@cse.ohio-state.edu