

TinyOS Tutorial

Mo Sha

Fall 2012

Outline

- **Installing TinyOS and Building Your First App**
- Hardware Primer
- Basic nesC Syntax
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques

TinyOS Installation

➤ TinyOS Community

❑ <http://www.tinyos.net/>

❑ Various installation options listed under “Getting started” section



TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. A worldwide community from academia and industry use, develop, and support the operating system as well as its associated tools, averaging 35,000 downloads a year.

Latest News

July 21, 2010: The transition from hosting TinyOS at Sourceforge to **Google code** is now complete. Part of this transition included placing all of TinyOS under a **New BSD license** (in Sourceforge several compatible licenses were used). Thanks to all of the developers for agreeing to move to a uniform license!

April 6, 2010: TinyOS 2.1.1 is now officially released; you can download it from the debian packages on tinyos.stanford.edu, or manual installation with RPMs with **the instructions on docs.tinyos.net**. TinyOS 2.1.1 includes:

- Support for the epic, mulle, and shimmer2 platforms,
- Support for 6lowpan, an IPv6 networking layer within the TinyOS network,
- Support for simple, uniform low-power networking across many protocols,
- Support for security on the CC2420 radio
- Improvements to many existing services and protocols, including the inclusion of a new dissemination protocol (DHV), improvements to CTP, improved TOSThreads documentation, and numerous bug fixes.

FAQ

Frequently asked questions

Learn

Download TinyOS and learn

Community

TinyOS Working Groups, mailing

TinyOS Installation (cont.)

TinyOS Documentation Wiki - T...


Log in / create a...

article

discussion

view source

history



navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

search

Go Search

toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

TinyOS Documentation Wiki

(Redirected from [Main Page](#))

This is the TinyOS Documentation Wiki. There are actually [541](#) articles in this Wiki. Everyone is welcome to edit these pages and contribute TinyOS documentation. In order to edit, you must first create an account using the "create account" link on the upper right. Consult the [Wiki User's Guide](#) for information on using the wiki software.

This site actively encourages contributions. If you are not familiar with the Wiki system, consult the [Wiki User's Guide](#). We prefer all documentation to live on this site, rather than having links to external sites; you really need to point at an external site, external links are OK too.

All content on this website is covered by the [Creative Commons Attribution-Share Alike 3.0 license](#). By contributing content to this site, you are hereby assigning this license to such content.

Starting with TinyOS

[Getting started](#): downloading, installing the most recent version of TinyOS (2.1.1), and where to go next ([instructions for 2.1](#) and [2.0.2](#))

[The simplest TinyOS program](#): simple example code that compiles

[Tutorials](#): an introduction to TinyOS programming.

[Using TinyOS](#): complete list of tutorials, programming guides, and other resources for getting started with TinyOS.

[Quick Start Guide](#): TinyOS 2.1 on Ubuntu 9.10 with TelosB motes

[Search engine](#): search TinyOS documentation and mailing lists for answers to common problems

[TinyOS Development Tree](#): the active TinyOS development source tree on Google code

Detailed Documentation

[TinyOS Programming Manual](#): This detailed (200 page) book on programming TinyOS 2.0 is an early version of [TinyOS Programming](#) (Excerpt with chapters 1-7).

[TEPs](#): TinyOS Enhancement Proposals, which describe the TinyOS APIs and their implementations.

[Source Code Documentation](#): HTML documentation of TinyOS source code

[Platform Hardware](#) layouts, chips, and other details


[nesC 1.1 reference manual](#). The nesC 1.3 release includes a newer version in doc/ref.pdf ([nesC 1.3 tarball from Sourceforge](#)).

TinyOS Installation (cont.)

Installing TinyOS 2.1.1

TinyOS has numerous improvements to TinyOS 2.1. Its features include:

- Support for the epic, mulle, and shimmer2 platforms,
- Support for 6lowpan, an IPv6 networking layer within the mote network,
- Support for simple, uniform low-power networking across many protocols,
- Improvements to many existing services and protocols, including the inclusion of a new dissemination protocol (DHV), improved

More information can be found in the [release notes](#) .

Officially Supported Methods

- Full System:
 - [One step installation with a Live CD](#), (doesn't currently work)
- Windows:
 - [Manual installation using cygwin and RPM packages](#)
 - [Running a XubuntuTOS Virtual Machine Image in VMware Player](#)
- Linux:
 - [Manual installation using RPM packages](#)
 - [Automatic installation for debian systems using the TinyOS debian repository](#)
 - [Running a XubuntuTOS Virtual Machine Image in VMware Player](#)

User Contributed Methods

- [One-step Install, TinyOS 2.1.1 OSIAN IPv6 : Ubuntu 10.04 Bootable DVD \(32-bit i386\)](#) 
- [Installing TinyOS on Mac OS \(Tiger & Leopard\)](#)
- [Installing TinyOS-2.x on Mac OS X \(Snow Leopard\)](#)
- [Installing TinyOS on Gentoo](#) 
- [Installing TinyOS on Ubuntu](#)  [Chinese Version](#)  [Chinese](#) 
- [Unofficial Macport tool chain for TinyOS from Johns Hopkins](#) 
- [Installing TinyOS from Source on Fedora 13 64bit](#)
- [Installing Xubuntos in VirtualBox](#)
- [Automated TinyOS installer for Linux](#) 

Other Methods

TinyOS Installation (cont.)

- Pre-compiled .rpm and .deb packages for Fedora and Ubuntu Linux users
 - ❑ **Ubuntu users: be sure to remove brlty package**
 - ❑ **Automatic installation for debian systems using the TinyOS debian repository (Manual install TI MSP430 Tools)**
 - ❑ All necessary drivers already included with Linux kernel
- OS X unofficially supported but works well
- Windows installation uses Cygwin to emulate Linux software layer
 - ❑ Works well under XP, refuses to work on some Vista/7 machines (updating Cygwin after the installation may help)
 - ❑ **Step 5b (optional): Post-install fixes in Windows/Cygwin environment**
- “Running a XubunTOS Virtual Machine Image in VMware Player”



Post-install fixes in Windows/Cygwin

Installing T...

```
export CLASSPATH= cygpath -w $TOSROOT/support/sdk/java/tinyos.jar
export CLASSPATH="$CLASSPATH;."
```

TinyOS 2.x

Environment Variable	Windows	Linux
TOSROOT	/opt/tinyos-2.x	same as in Cygwin
TOSDIR	\$TOSROOT/tos	same as in Cygwin
CLASSPATH	C:\cygwin\opt\tinyos-2.x\support\jdk\java\tinyos.jar;	\$TOSROOT/support/sdk/java/tinyos.jar;
MAKERULES	\$TOSROOT/support/make/Makerules	same as in Cygwin
PATH†	/opt/msp430/bin:/opt/jflashmm:\$PATH	same as in Cygwin

†Only necessary if you're using MSP430 or iMote2 platform/tools.

In addition to the above environment variables, do the following on Linux machines:

1. Change the ownership on your /opt/tinyos-2.x files: `chown -R <your uid> /opt/tinyos-2.x`
2. Change the permissions on any serial (/dev/ttyS<N>), usb (/dev/ttyUSB<N>), or parallel (/dev/parport) devices you are going to use: `chmod 666 /dev/<devicename>`

Step 5b (optional): Post-install fixes in Windows/Cygwin environment

If you later experience problems when building some tutorials, running Java tools, ... see [Geoffrey Lo's excellent blog post](#), especially step 2.

Step 6: Installing Graphviz

Go to [download page](#) of the Graphviz project and download the appropriate RPM. You only need the basic graphviz RPM (graphviz-); you don't need all of the add-ons, such as -devel, -doc, -perl, etc. If you are not sure what version of Linux you're running,

```
uname -a
```



Washington University in St. Louis

TinyOS Installation (cont.)

- OS X unofficially supported but works well
- Precompiled packages available at <http://students.cec.wustl.edu/~ms31/tinyos-2.1.1.dmg>
 - ❑ Need to install pySerial (<http://pyserial.sourceforge.net>) and FTDI FT232R serial drivers (<http://www.ftdichip.com/Drivers/VCP.htm>) separately
- Can also compile using MacPorts: see TinyOS Wiki



TinyOS Installation Problems

➤ If you run `tos-check-env` and it says:

-> WARNING: CLASSPATH environment variable doesn't exist.

➤ then add:

```
export CLASSPATH=/opt/tinyos-2.1.1/support/sdk/java/  
tinyos.jar:.
```

to your `.bash_profile` or `.bash_aliases` file



TinyOS Installation Problems

- If you try to run `make micaz sim` and it gives you a long error message starting with:

 `...: error: inttypes.h: No such file or directory`
- then you need to install the C library/header files for your own architecture
- On Ubuntu/Debian, run `apt-get install build-essential`



TinyOS Directory Structure

➤ /opt/tinyos-2.1.1 (\$TOSROOT)

- ❑ apps
- ❑ support
 - make
 - sdk
- ❑ tools
- ❑ tos



make System

- \$TOSROOT/support/make includes lots of Makefiles to support the build process
- Create a simple stub Makefile in your app directory that points to main component

```
COMPONENT=[MainComponentC]  
SENSORBOARD=[boardtype] # if needed  
include $(MAKERULES)
```

- make [platform] in app directory
 - ❑ Builds but does not install program
 - ❑ platform: one of the platforms defined in \$TOSROOT/tos/platforms (mica2, micaz, telosb)



make System

- `make [re]install.[node ID] [platform]`
`[programming options]`
 - ❑ node ID: 0 - 255
 - ❑ programming options:
 - mica2/micaz: mib510, /dev/ttyXYZ
 - telosb: bs1, /dev/ttyXYZ
- `make clean`
- `make docs [platform]`
 - ❑ Generates HTML documentation in `$TOSROOT/doc/nedoc/[platform]`

Useful commands

- `motelist`
 - ❑ See the list of motes connecting with pc
- `make telosb`
 - ❑ Compile your code
- `make telosb reinstall,1`
 - ❑ Program the mote
- `make telosb install,1`
 - ❑ Compile your code
 - ❑ Program the mote
- `make docs telosb`
 - ❑ Generate docs



Build Stages

```
Terminal — bash — 80x35 — 361
gwh2@rooster148:/opt/tinyos-2.1.0/apps/Blink :(> make install.0 telosb bsl,/dev
/tty.usbserial-M4A5L524
mkdir -p build/telosb
compiling BlinkAppC to a telosb binary
ncc -o build/telosb/main.exe -Os -O -mdisable-hwmu1 -Wall -Wshadow -Wnesc-all -
target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x2
2 -DIDENT_APPNAME=\"BlinkAppC\" -DIDENT_USERNAME=\"gwh2\" -DIDENT_HOSTNAME=\"roo
ster148.cse.\" -DIDENT_USERHASH=0xb9e110b0L -DIDENT_TIMESTAMP=0x48c59807L -DIDEN
T_UIDHASH=0x46b3cb61L BlinkAppC.nc -lm
compiled BlinkAppC to build/telosb/main.exe
2650 bytes in ROM
55 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
writing TOS image
tos-set-symbols --objcopy msp430-objcopy --objdump msp430-objdump --target ihex
build/telosb/main.ihex build/telosb/main.ihex.out-0 TOS_NODE_ID=0 ActiveMessageA
ddressC$addr=0
Could not find symbol ActiveMessageAddressC$addr in build/telosb/main.exe, ignor
ing symbol.
Could not find symbol TOS_NODE_ID in build/telosb/main.exe, ignoring symbol.
installing telosb binary using bsl
tos-bsl --telosb -c /dev/tty.usbserial-M4A5L524 -r -e -I -p build/telosb/main.ih
ex.out-0
MSP430 Bootstrap Loader Version: 1.39-telos-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
2682 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out-0 build/telosb/main.ihex.out-0
gwh2@rooster148:/opt/tinyos-2.1.0/apps/Blink :>
```

Preprocess .nc to .c, then compile .c to binary

Set AM address and node ID in binary

Program mote



Example under Windows

```
Mo ShaEmo ~  
$ cd /opt/tinyos-2.x/apps/Blink/
```

C:\cygwin\bin

<http://students.cec.wustl.edu/~ms31/520S/>

```
Mo ShaEmo /opt/tinyos-2.x/apps/Blink  
$ motelist  
Reference      CommPort      Description  
-----  
XBTFJTE2      COM34          Crossbow Telos Rev.B  
  
Mo ShaEmo /opt/tinyos-2.x/apps/Blink  
$ make telosb install,1 bsl,33  
mkdir -p build/telosb  
    compiling BlinkAppC to a telosb binary  
ncc -o build/telosb/main.exe -Os -O -mdisable-hwmutl -fnesc-separator=__ -Wall -  
Wshadow -Wnesc-all -target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFI  
NED_TOS_AM_GROUP=0x22 -DIDENT_APPNAME=\"BlinkAppC\" -DIDENT_USERNAME=\"MoSha\" -  
DIDENT_HOSTNAME=\"mo\" -DIDENT_USERHASH=0xbf4e9ec7L -DIDENT_TIMESTAMP=0x4e5bd537  
L -DIDENT_UIDHASH=0xe48ce2f1L BlinkAppC.nc -lm  
    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  
tos-set-symbols --objcopy msp430-objcopy --objdump msp430-objdump --target ihex  
build/telosb/main.ihex build/telosb/main.ihex.out-1 TOS_NODE_ID=1 ActiveMessageA  
ddressC__addr=1  
Could not find symbol ActiveMessageAddressC__addr in build/telosb/main.exe, igno  
ring symbol.  
Could not find symbol TOS_NODE_ID in build/telosb/main.exe, ignoring symbol.  
    installing telosb binary using bsl  
tos-bsl --telosb -c 33 -r -e -I -p build/telosb/main.ihex.out-1  
MSP430 Bootstrap Loader Version: 1.39-telos-8  
Mass Erase...  
Transmit default password ...  
Invoking BSL...  
Transmit default password ...  
Current bootstrap loader version: 1.61 <Device ID: f16c>  
Changing baudrate to 38400 ...  
Program ...  
2680 bytes programmed.  
Reset device ...  
rm -f build/telosb/main.exe.out-1 build/telosb/main.ihex.out-1  
  
Mo ShaEmo /opt/tinyos-2.x/apps/Blink
```


“Homework”

- Install TinyOS 2.1.1 and build Blink

(Not graded, but a good idea to make sure you have everything up and running)



How to Get Help

- TinyOS Documentation Wiki: <http://docs.tinyos.net>
- TinyOS Programming Manual: 139-page PDF intro to nesC and TinyOS 2.x:
<http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf>
- TinyOS Tutorials: short HTML lessons on using parts of TinyOS (sensors, radio, TOSSIM, etc.):
http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials





gation

[Main Page](#)[Community portal](#)[Current events](#)[Recent changes](#)[Random page](#)[Help](#)[Donations](#)

rch

Go

Search

box

[What links here](#)[Related changes](#)[Upload file](#)[Special pages](#)[Printable version](#)[Permanent link](#)[article](#)[discussion](#)[view source](#)[history](#)

TinyOS Tutorials

These brief tutorials are intended to get you started with TinyOS. They show you the basics of writing, compiling, and installing TinyOS applications, communication, sensing, and storage. The later tutorials go a little deeper into some of the more advanced areas of TinyOS, such as handling

Contents [\[hide\]](#)

1 Working Group Tutorials

[1.1 Getting Started with TinyOS](#)[1.2 Modules and the TinyOS Execution Model](#)[1.3 Mote-mote radio communication](#)[1.4 Mote-PC serial communication and SerialForwarder](#)[1.5 Sensing](#)[1.5.1 ADC](#)[1.6 Boot Sequence](#)[1.7 Storage](#)[1.8 Resource Arbitration and Power Management](#)[1.9 Concurrency](#)[1.10 Platforms](#)[1.11 TOSSIM](#)[1.12 Network Protocols](#)[1.13 TinyOS Toolchain](#)[1.14 Building a simple but full-featured application](#)[1.15 The TinyOS printf Library](#)[1.16 Writing Low-Power Applications](#)[1.17 TOSThreads Tutorial](#)[1.18 CC2420 Security Tutorial](#)

2 Other Tutorials

[2.1 Ipsn2009-tutorial](#)

3 User Contributed Tutorials

[3.1 Platform Creation and Testing](#)[3.2 Rssi Demo](#)

Working Group Tutorials

How to Get Help

- nesdoc: annotated API for all interfaces and components in TinyOS:
http://docs.tinyos.net/tinywiki/index.php/Source_Code_Documentation
- TinyOS Enhancement Protocols (TEP): formal documentation for TinyOS features:
<http://docs.tinyos.net/tinywiki/index.php/TEPs>



Outline

- Installing TinyOS and Building Your First App
- **Hardware Primer**
- Basic nesC Syntax
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques



Available Hardware

- Motes, sensor boards, and gateways
- Most up to date list here:
[http://wsn.cse.wustl.edu/index.php/
Equipment for course projects](http://wsn.cse.wustl.edu/index.php/Equipment_for_course_projects)
- Hardware is in Mobile Computing Laboratory (Jolley 519)

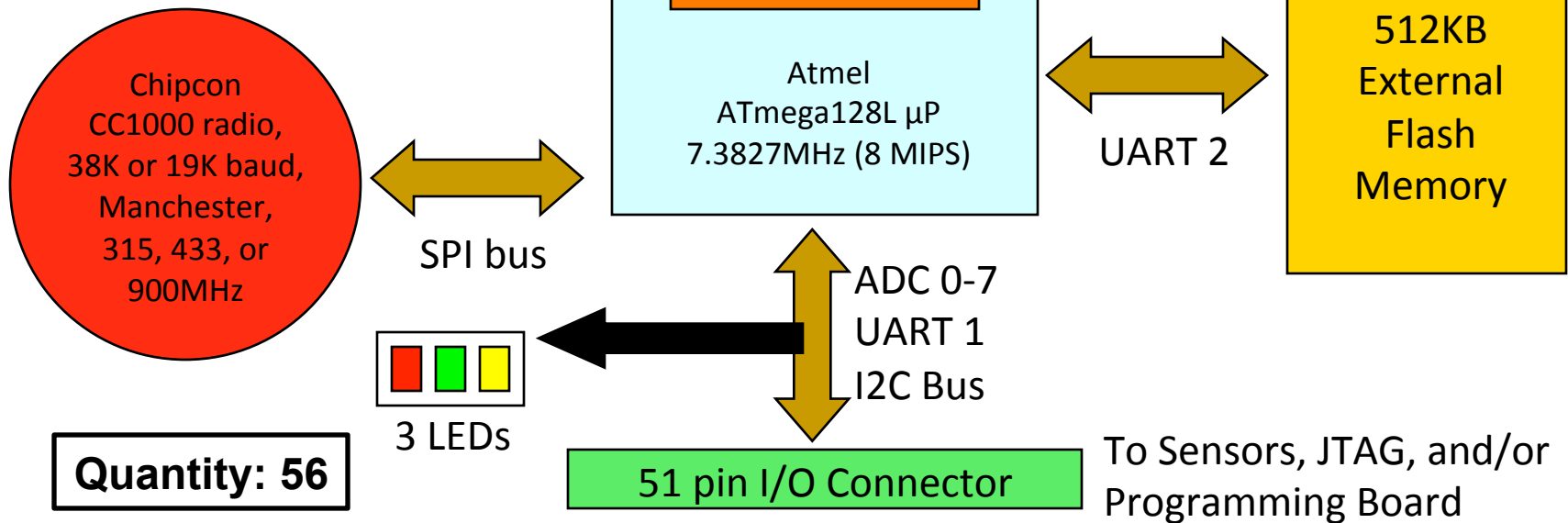
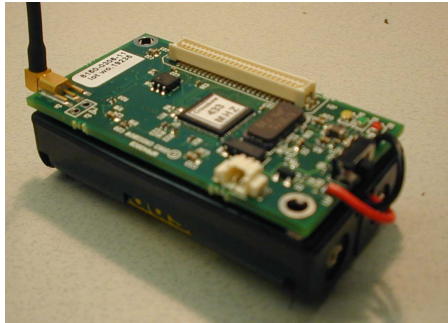


Tmote Sky (aka TelosB)

- IEEE 802.15.4 Radio
 - ❑ 250kbps
- TI MSP430 microcontroller
 - ❑ 16MHz, 16 MIPS, 10kB RAM
- Integrated antenna & USB interface
- Low power utilization
 - ❑ 1.8mA/5.1μA vs. Mica 2's 8mA/15μA
- Quantity w/o on-board sensors: 6
- Quantity w/temperature, light, and humidity sensors: 20
- Quantity w/SBT80 sensor board: 4



MICA2 Mote (MPR400CB)

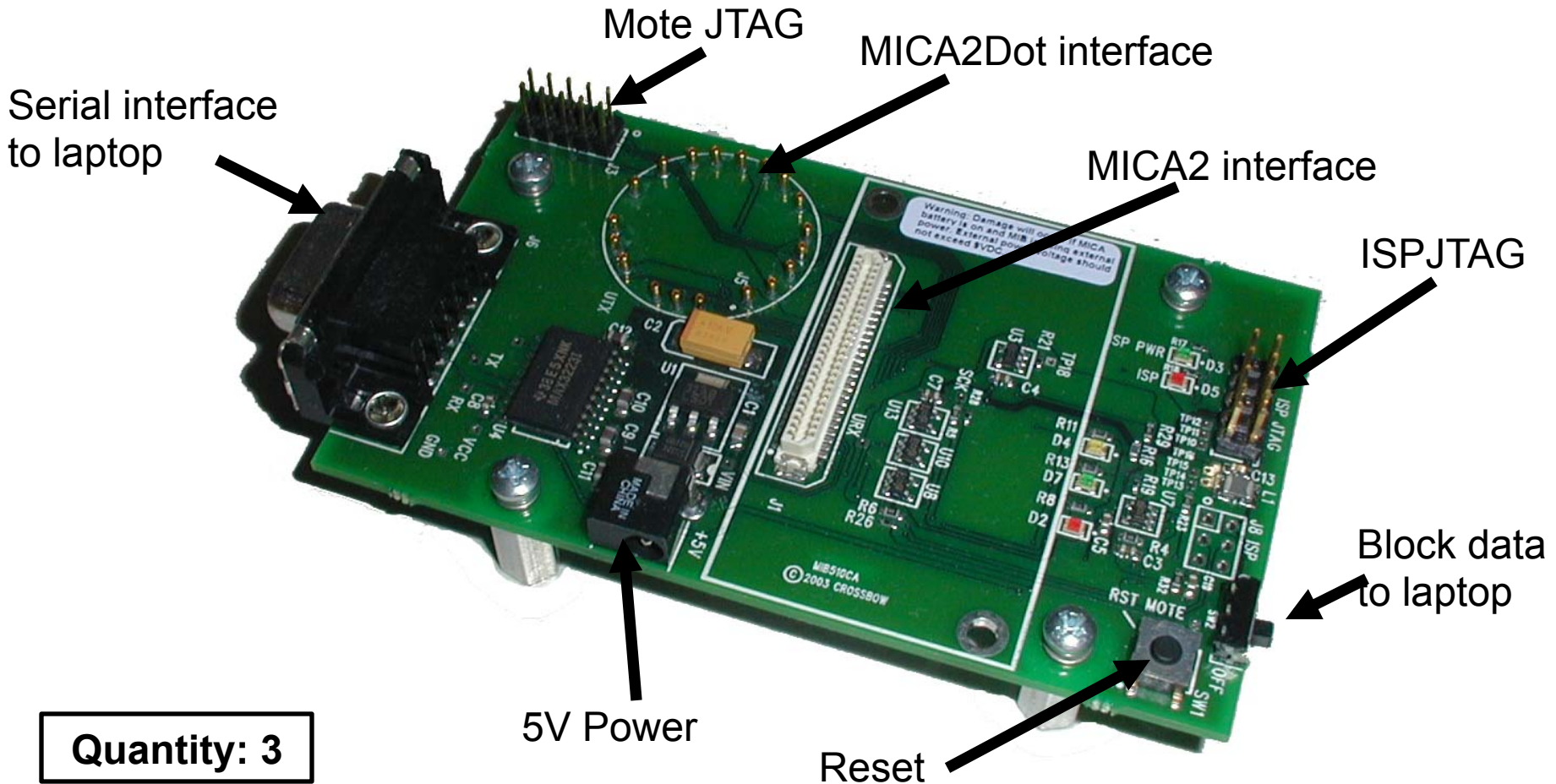


MPR2400 MICAz

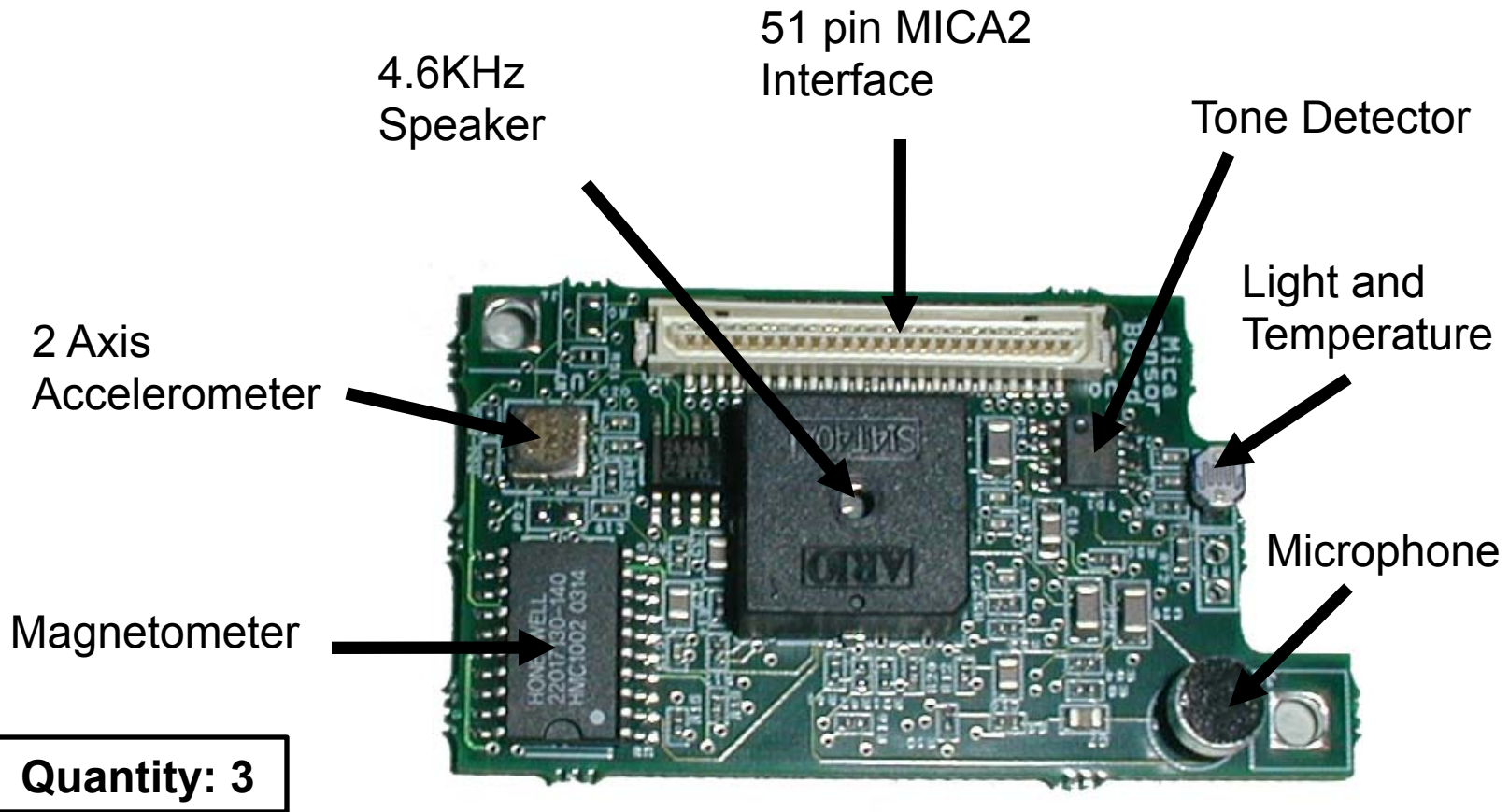
- Same as Mica2 except with IEEE 802.15.4 radio
 - ❑ 2.4GHz
 - ❑ 250kbps
- Quantity: 7



Programming Board (MIB510)



MTS310CA Sensor Board



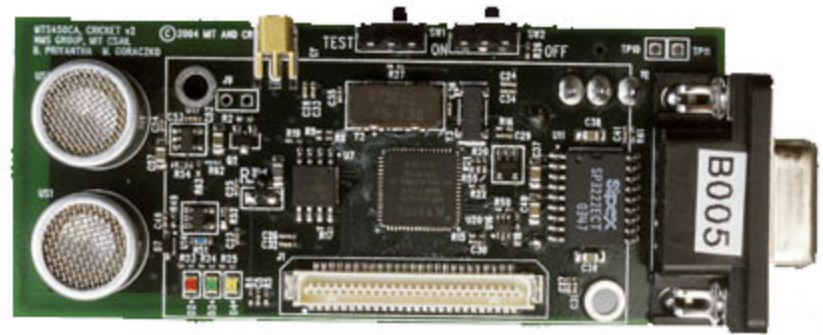
NSLU2 Network Storage Link (“Slug”)

- 266MHz Xscale CPU, 32MB SDRAM, 8MB flash, 1x Ethernet port
- Wired power
- No built-in radio, but 2x USB 2.0 ports for add-on 802.11/Bluetooth/mote interface
- Can be easily converted to an embedded Linux box with third-party firmware
 - ❑ Our testbed uses the OpenWrt distribution (<http://openwrt.org>)
- Quantity: 15



MCS410 Cricket Mote

- A Mica2 mote with ultrasound Rx/Tx
- Indoor localization
 - ❑ 10.5m range
- Distance accuracy:
 - ❑ 1 cm (< 3.5m dist)
 - ❑ 2cm (>3.5m dist)
- Mote finds distance, PC finds location
- <http://cricket.csail.mit.edu/>
- Quantity: 9



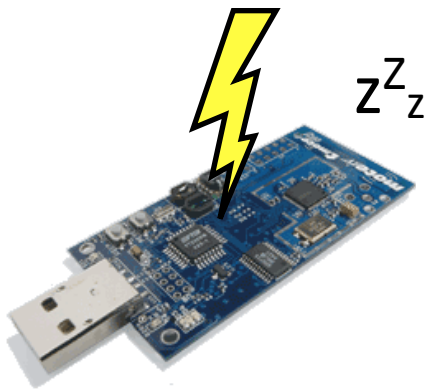
Outline

- Installing TinyOS and Building Your First App
- Hardware Primer
- **Basic nesC Syntax**
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques



TinyOS Execution Model

- To save energy, node stays asleep most of the time
- Computation is kicked off by hardware interrupts
- Interrupts may schedule tasks to be executed at some time in the future
- TinyOS scheduler continues running until all tasks are cleared, then sends mote back to sleep



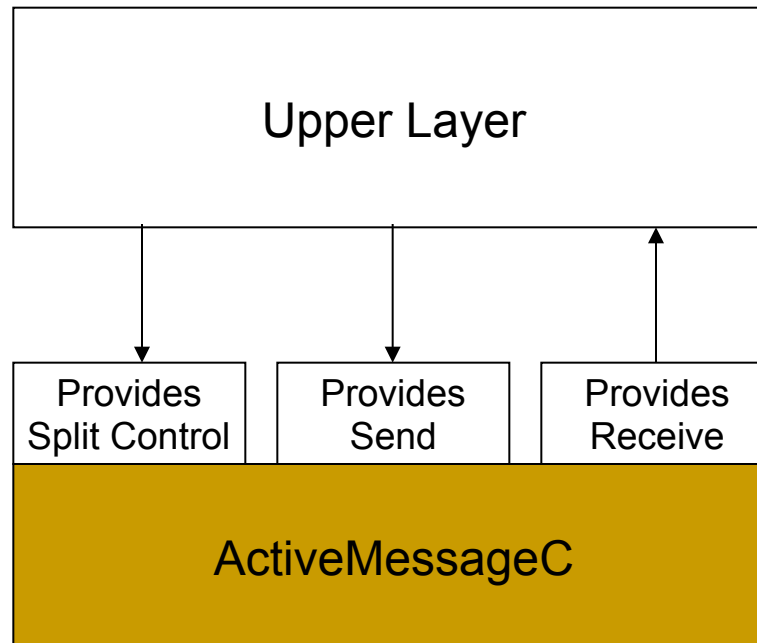
HandlePacket

ReadSensor

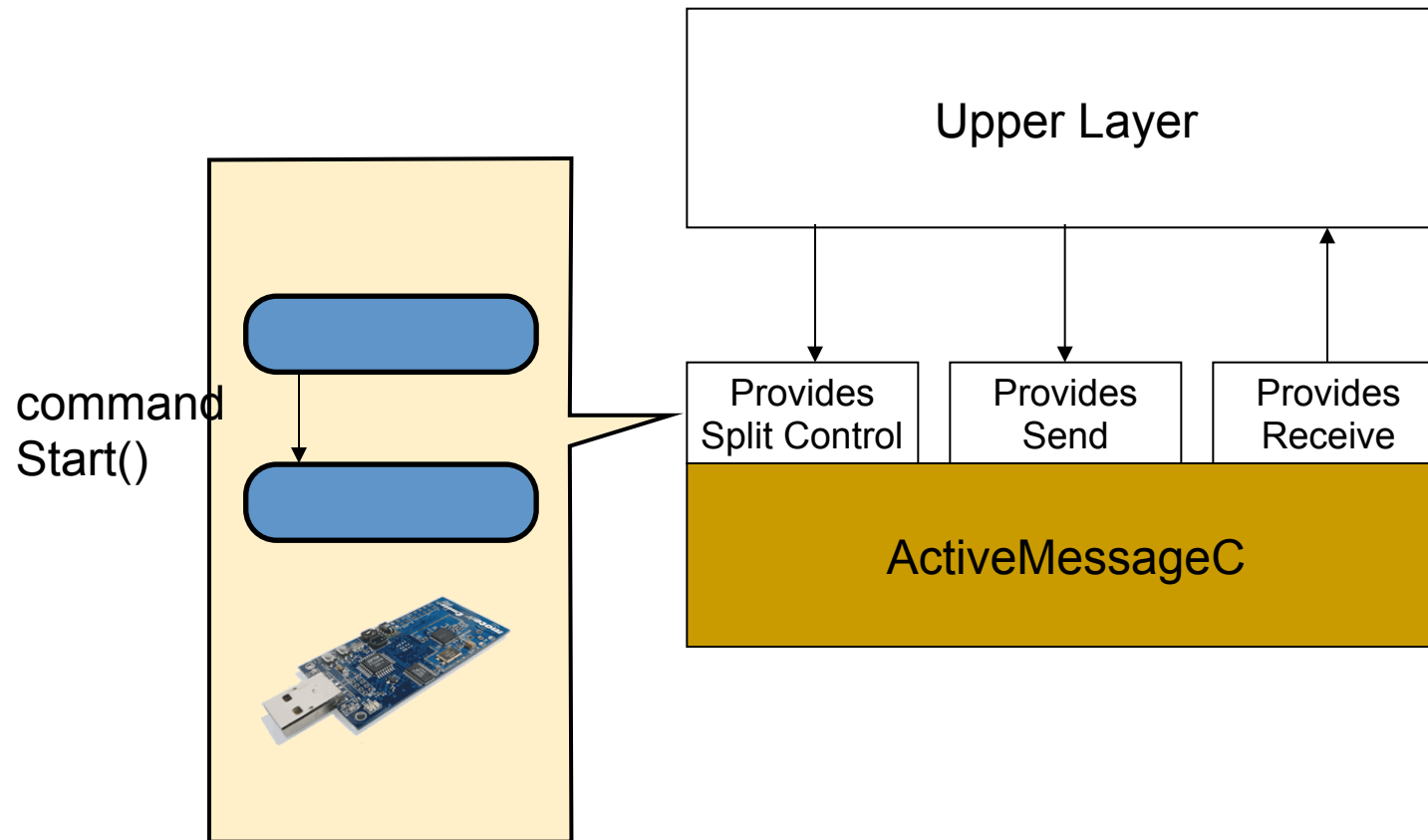
TimerFired



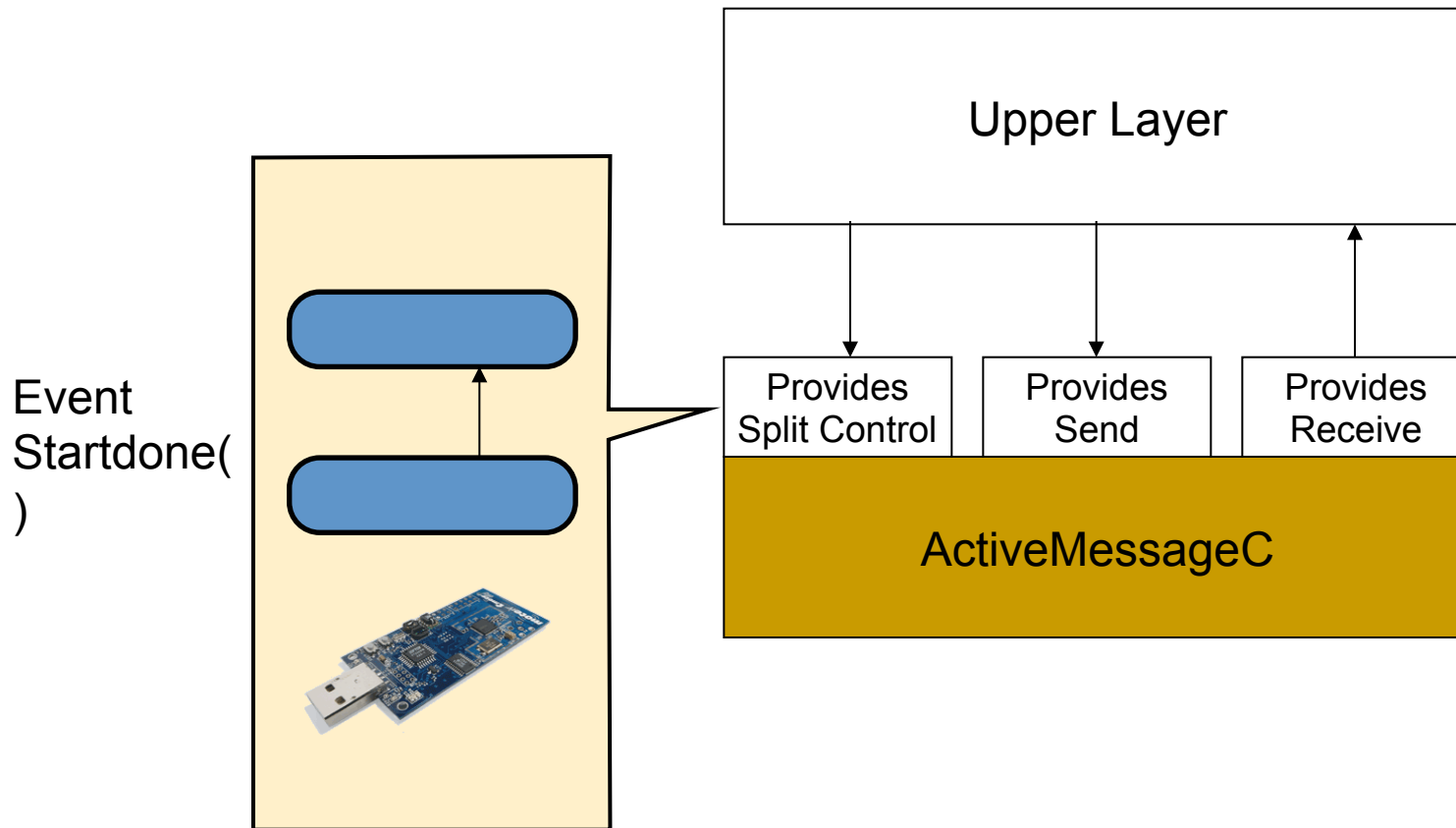
TinyOS Component Model



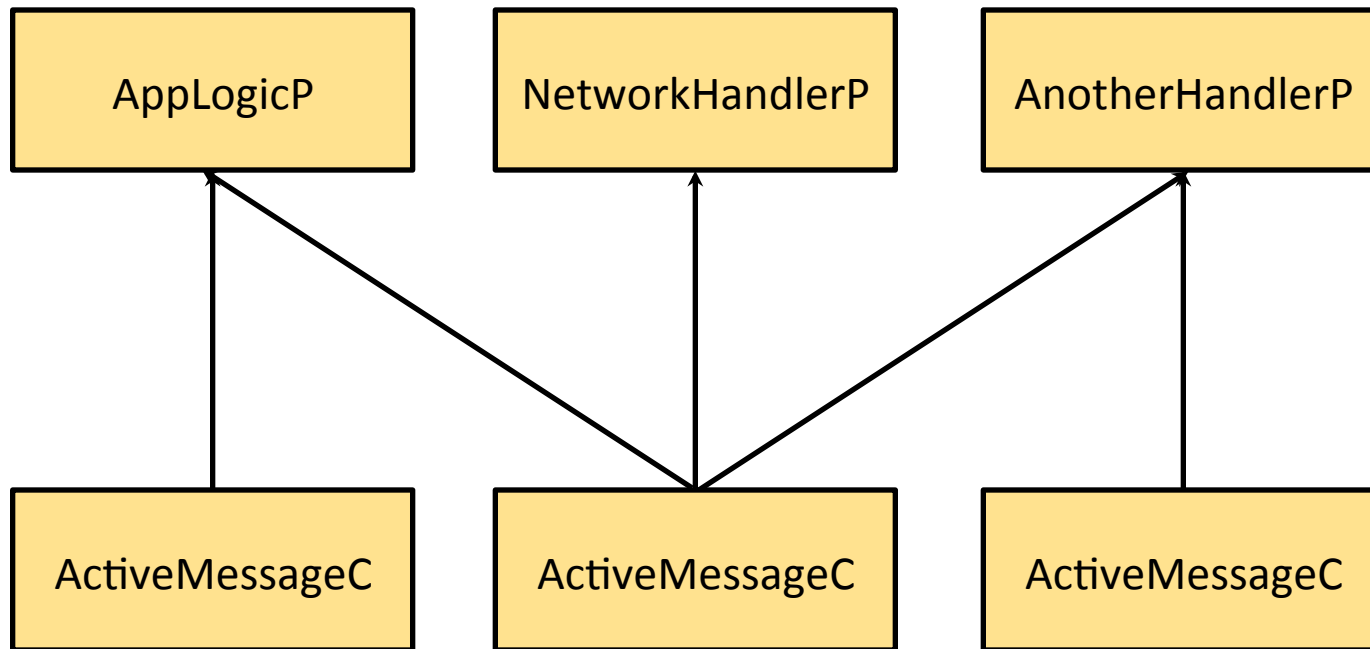
TinyOS Component Model



TinyOS Component Model



Components != Objects



Interfaces

- List of exposed events and commands
- Like ordinary C function declarations, except with event or command in front

```
interface Receive {  
    event message_t * Receive(message_t * msg, void * payload,  
        uint8_t len);  
    command void * getPayload(message_t * msg, uint8_t * len);  
    command uint8_t payloadLength(message_t * msg);  
}
```




Modules

- Modules provide the implementation of one or more interfaces

- They may consume (use) other interfaces to do so

```
module ExampleModuleP {  
    provides interface SplitControl;  
    uses interface Receive;  
    uses interface Receive as OtherReceive;  
}  
implementation {  
    ...  
}
```



- “Rename” interfaces with the as keyword -- required if you are using/providing more than one of the same interface!

Modules

- `implementation` block may contain:
 - ❑ Variable declarations
 - ❑ Helper functions
 - ❑ Tasks
 - ❑ Event handlers
 - ❑ Command implementations

Modules: Variables and Functions

- Placed inside `implementation` block exactly like standard C declarations:

```
...  
implementation {  
    uint8_t localVariable;  
    void increment(uint8_t amount);  
  
    ...  
  
    void increment(uint8_t amount) {  
        localVariable += amount;  
    }  
}
```



Modules: Tasks

- Look a lot like functions, except:
 - ❑ Prefixed with task
 - ❑ Can't return anything or accept any parameters

```
implementation {  
    ...  
    task void legalTask() {  
        // OK  
    }  
    task bool illegalTask() {  
        // Error: can't have a return value!  
    }  
    task void anotherIllegalTask(bool param1) {  
        // Error: can't have parameters!  
    }  
}
```

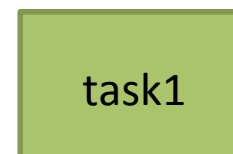


Modules: Task Scheduling

- Tasks are scheduled using the post keyword

```
post handlePacket();
```

- TinyOS guarantees that task will *eventually* run
 - ❑ Default scheduling policy: FIFO



...



Modules: Commands and Events

- Commands and events also look like C functions, except:
 - ❑ they start with the keyword command or event
 - ❑ the “function” name is in the form
InterfaceName.CommandOrEventName

➤ e.g.

```
implementation {  
    command error_t SplitControl.start() {  
        // Implements SplitControl's start() command  
    }  
  
    event message_t * Receive.receive(message_t * msg, void * payload,  
        uint8_t len) {  
        // Handles Receive's receive() event  
    }  
}
```



Modules: Commands and Events

- Commands are invoked using the `call` keyword:

```
call Leds.ledToggle();  
// Invoke the ledToggle command on the Leds interface
```

- Event handlers are invoked using the `signal` keyword:

```
signal SplitControl.startDone();  
// Invoke the startDone event handler on the SplitControl interface
```



Modules: Commands and Events

- A command, event handler, or function can call or signal *any* other command or event from *any* interface wired into the module:

```
module ExampleModuleP {
    uses interface Receive;
    uses interface Leds;
}
implementation {
    event message_t Receive.receive(message_t * msg, void * payload,
        uint8_t len) {
        // Just toggle the first LED
        call Leds.led0Toggle();
        return msg;
    }
    ...
}
```



Configurations

```
configuration NetworkHandlerC {  
    provides interface SplitControl;  
}  
implementation {  
    components NetworkHandlerP as NH,  
        ActiveMessageP as AM;  
    //NH.Receive -> AM.Receive;  
    //SplitControl = NH.SplitControl;  
    NH.Receive -> AM;  
    SplitControl = NH;  
}
```



Reminder: Race Conditions

- Use `atomic` blocks to avoid race conditions

```
implementation {  
    uint8_t sharedCounter;  
  
    async event void Alarm.fired() {  
        ...  
        sharedCounter++;  
    }  
  
    event void Receive.receive(...) {  
        ...  
        sharedCounter++;  
    }  
}
```



Reminder: Race Conditions

- Use `atomic` blocks to avoid race conditions

```
implementation {  
    uint8_t sharedCounter;  
  
    async event void Alarm.fired() {  
        atomic {  
            sharedCounter++;  
        } } Interrupts are disabled here -- use sparingly  
        and make as short as practical  
    }  
  
    event void Receive.receive(...) {  
        ...  
        sharedCounter++;  
    }  
}
```



Reminder: Race Conditions

- Tasks are always synchronous
- If timing isn't crucial, defer code to tasks to avoid race conditions

```
implementation {  
    uint8_t sharedCounter;  
  
    task void incrementCounter() { sharedCounter++; }  
  
    async event void Alarm.fired() {  
        post incrementCounter();  
    }  
  
    event void Receive.receive(...) {  
        ...  
        sharedCounter++;  
    }  
}
```

Task is scheduled
immediately, but
executes later



nesC and Race Conditions

- nesC can catch some, but not all, potential race conditions
- If you're absolutely sure that there's no race condition (or don't care if there is), use the `norace` keyword:

```
implementation {  
    uint8_t sharedCounter;  
  
    async event void Alarm1.fired() {  
        sharedCounter++;  
        call Alarm2.start(200);  
    }  
  
    async event void Alarm2.fired() {  
        sharedCounter--;  
        call Alarm1.start(200);  
    }  
}
```

Race condition is
impossible; events
are mutually
exclusive



TOSThreads

- New in TinyOS 2.1: the TOSThreads threading library
- Threads add a third execution context to TinyOS's concurrency layer
 - ❑ Lowest priority: only run when TinyOS kernel is idle
 - ❑ Threads are preemptable by anything: sync, async, or other threads
- Also adds a library of synchronization primitives (mutex, semaphore, etc.) and blocking wrappers around non-blocking I/O
- Described in TOSThreads Tutorial (http://docs.tinyos.net/index.php/TOSThreads_Tutorial) or TEP 134



Example-Blink (opt\tinyos-2.x\apps\Blink)

Three files:

1.Makefile

2.BlinkC.nc (module)

3.BlinkAppC.nc (configuration)



Makefile

```
COMPONENT=BlinkAppC  
include $(MAKERULES)
```



BlinkC.nc

```
#include "Timer.h"
```

```
module BlinkC {  
  uses interface Timer<TMilli> as Timer0;  
  uses interface Timer<TMilli> as Timer1;  
  uses interface Timer<TMilli> as Timer2;  
  uses interface Leds;  
  uses interface Boot;  
}
```

implementation

```
{  
  event void Boot.booted()  
  {  
    call Timer0.startPeriodic( 250 );  
    call Timer1.startPeriodic( 500 );  
    call Timer2.startPeriodic( 1000 );  
  }  
  event void Timer0.fired()  
  {  
    call Leds.led0Toggle();  
  }  
  event void Timer1.fired() {..}  
  event void Timer2.fired() {...}  
}
```



BlinkAppC.nc

```
configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;

  BlinkC -> MainC.Boot;
  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Leds -> LedsC;
}
```



Outline

- Installing TinyOS and Building Your First App
- Hardware Primer
- Basic nesC Syntax
- **Advanced nesC Syntax**
- Network Communication
- Sensor Data Acquisition
- Debugging Tricks and Techniques



High-Level Summary

- nesC includes a lot of complex features that try to alleviate design problems with TinyOS 1.x
- The good news: you will probably never have to **write** code that incorporates these features
- The bad news: you're almost certain to **use** code that incorporates these features



Interfaces with Arguments

- Creating new interfaces to support different data types can get redundant fast

```
interface ReadUint16 {  
    command error_t read();  
    event void readDone(error_t error, uint16_t value);  
}
```

```
interface ReadBool {  
    command error_t read();  
    event void readDone(error_t error, bool value);  
}
```



Interfaces with Arguments

- If you want to make an interface adapt to different underlying types, then put a placeholder in angle brackets:

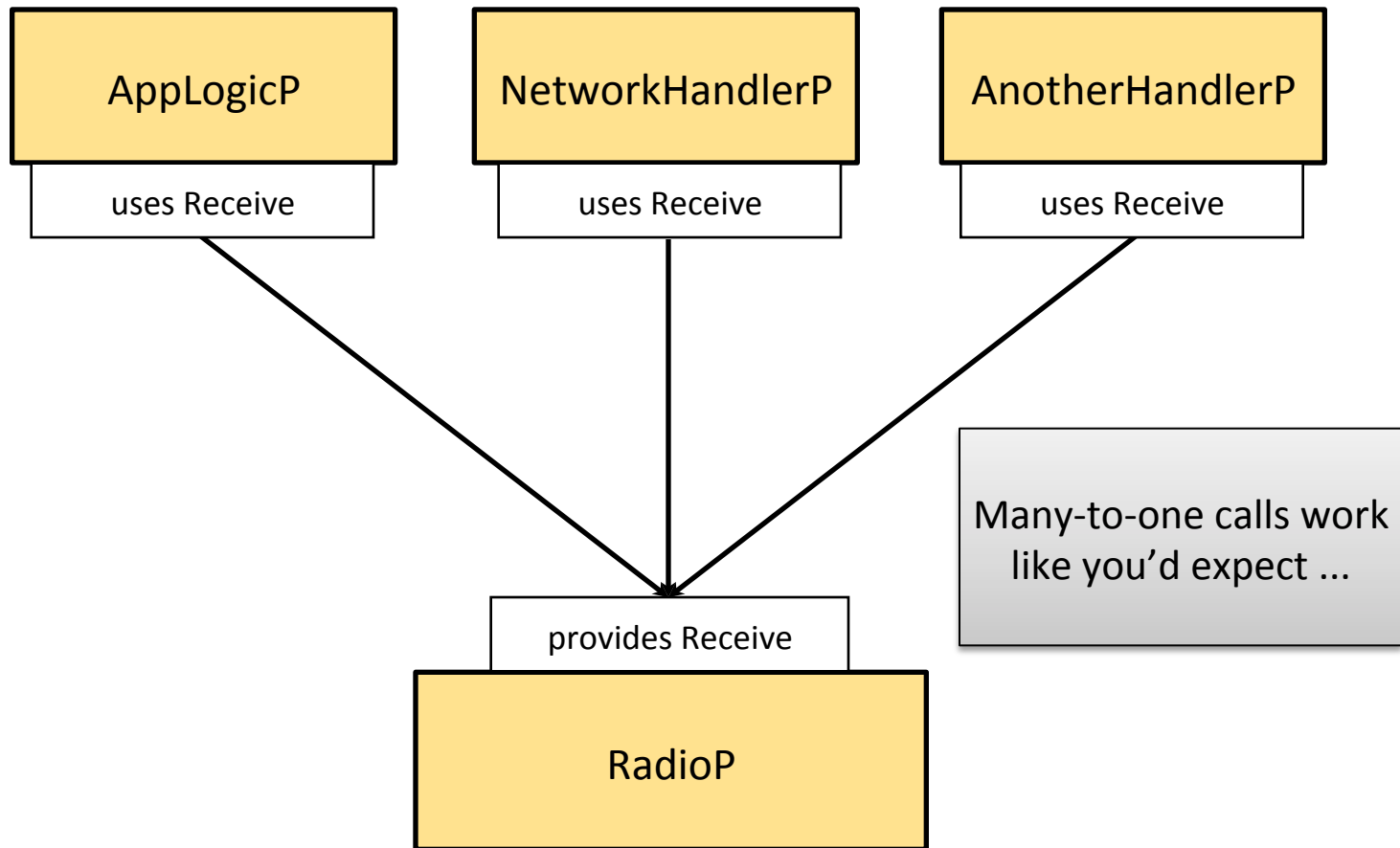
```
interface Read<type> {  
    command error_t read();  
    event void readDone(error_t error, type value);  
}
```

```
module SixteenBitSensorP {  
    provides interface Read<uint16_t>;  
}
```

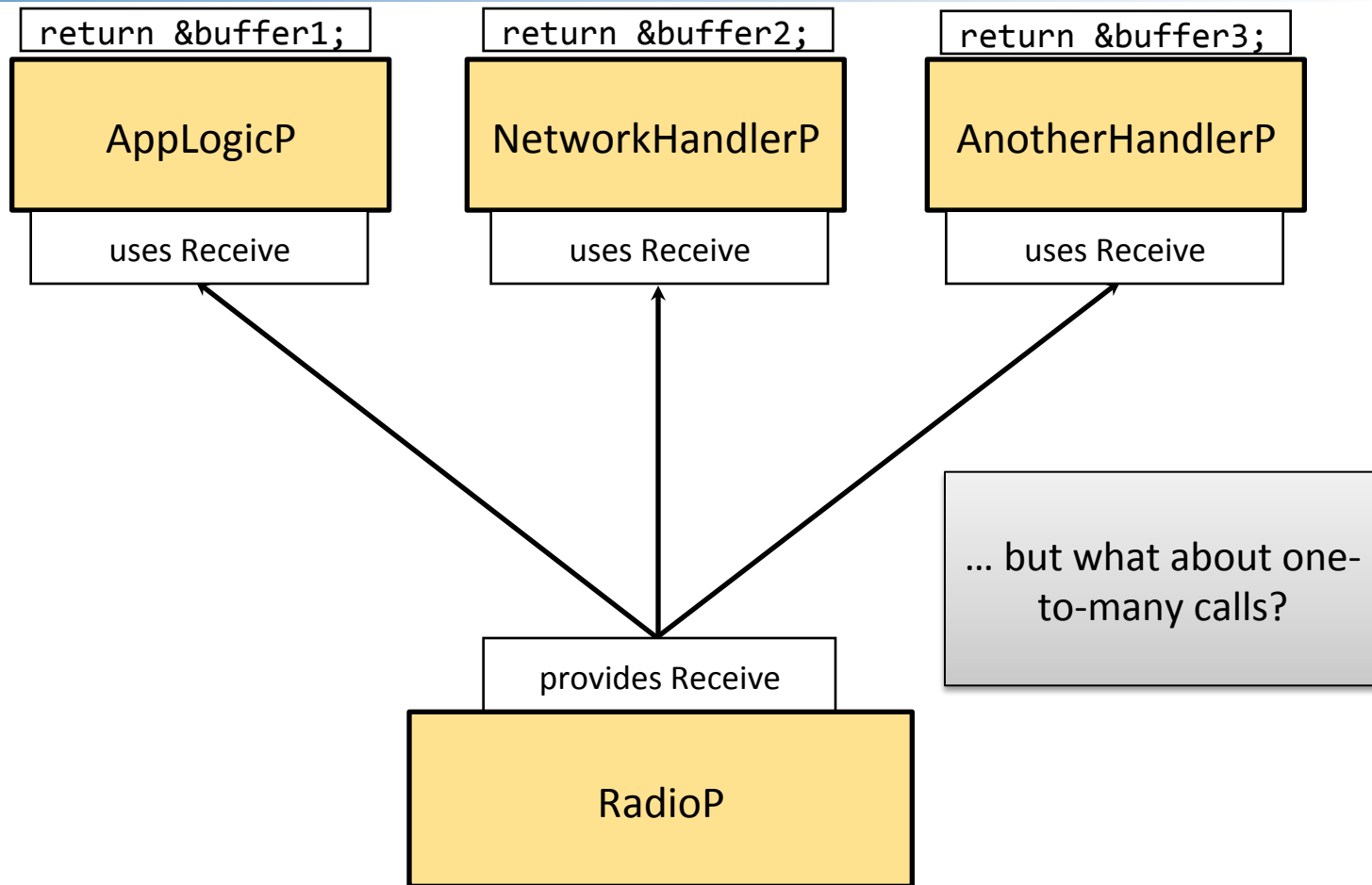
```
module BooleanSensorP {  
    provides interface Read<bool>;  
}
```



Fan-In: No Big Deal



Fan-Out: Bad Things Happen



Fan-Out: What Bad Things Happen?

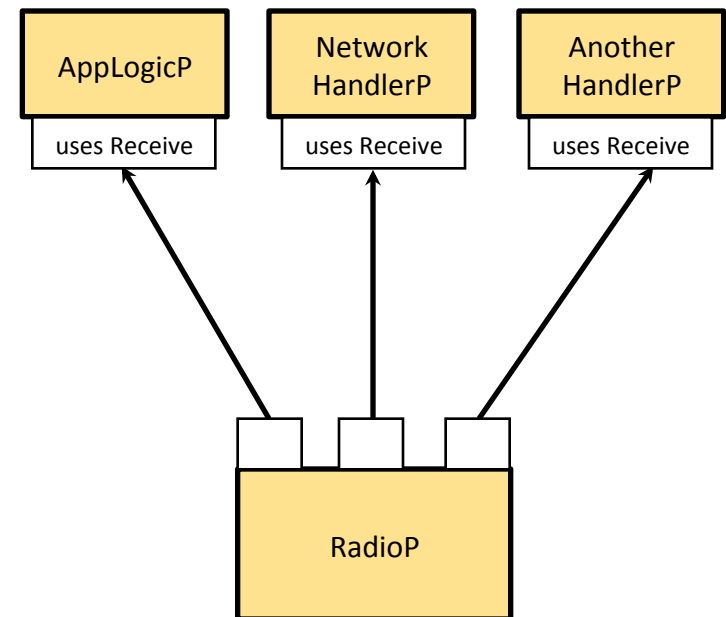
- If different return values come back, nesC may not be able to make sense of the contradiction and will *arbitrarily* pick one
- Avoid designs where this is possible
- If you can't avoid it, see TinyOS Programming Guide 5.2 for more info on combining return values



Parameterized Wiring

➤ Consider the following way to avoid fan-out:

```
module RadioP {  
  provides interface Receive as Receive0;  
  provides interface Receive as Receive1;  
  provides interface Receive as Receive2;  
  uses interface LowLevelRadio;  
  ...  
}  
implementation {  
  event void LowLevelRadio.packetReceived(  
    uint8_t * rawPacket) {  
    ...  
    uint8_t type = decodeType(rawPacket);  
    if(type == 0)  
      signal Receive0.receive(...);  
    else if(type == 1)  
      signal Receive1.receive(...);  
    ...  
  }  
  ...  
}
```



Parameterized Wiring

- The idea works in concept, but isn't maintainable in practice
- But nesC can approximate the behavior in a much more maintainable way:

```
module RadioP {  
    provides interface Receive[uint8_t id];  
    ...  
}  
implementation {  
    event void LowLevelRadio.packetReceived(uint8_t * rawPacket) {  
        ...  
        uint8_t type = decodeType(rawPacket);  
        signal Receive[type].received(...);  
    }  
    ...  
}
```



Using Parameterized Wiring

- You can wire parameterized interfaces like so:

```
AppLogicP -> RadioP.Receive[0];  
NetworkHandlerP -> RadioP.Receive[1];  
AnotherHandlerP -> RadioP.Receive[2];
```

- If each component is wired in with a unique parameter, then fan-out goes away

Unique Parameters

- In most cases, it's unreasonable to expect the user to count the number of times (s)he is using the interface and wire accordingly
- nesC can automatically generate a unique parameter for you using the `unique()` macro:

```
AppLogicP -> RadioP.Receive[unique("RadioP")];  
// unique("RadioP") expands to 0
```

```
NetworkHandlerP -> RadioP.Receive[unique("RadioP")];  
// unique("RadioP") expands to 1
```

```
AnotherHandlerP -> RadioP.Receive[unique("RaadioP")];  
// unique("RaadioP") expands to 0 (oops)
```

...

uniqueCount()

- What if your component needs to store different state for each unique parameter?

```
module RadioP {  
    ...  
}  
implementation {  
    int16_t state[uniqueCount("RadioP")];  
  
    ...  
}
```

uniqueCount(X)
expands to # of times
unique(X) appears in
the application

Defaults

- If you provide a parameterized interface and signal an event on it, you must also give a default event handler:

```
module SharedComponentP {  
    ...  
}  
implementation {  
    event void LowLevelRadio.packetReceived(uint8_t * rawPacket) {  
        ...  
        signal Receive[type].received(...);  
    }  
  
    default event void Receive.received[uint8_t id](...) {  
        // e.g., do nothing  
    }  
    ...  
}
```

Outline

- Installing TinyOS and Building Your First App
- Hardware Primer
- Basic nesC Syntax
- Advanced nesC Syntax
- **Network Communication**
- Sensor Data Acquisition
- Debugging Tricks and Techniques



Slight Diversion: App Bootstrapping

- Each app has a “main” configuration which wires together the app’s constituent components
- But how do these components start running?
- TinyOS includes a MainC component which provides the Boot interface:

```
interface Boot {  
    event void booted();  
}
```



Slight Diversion: App Bootstrapping

- Create one module which initializes your application, then wire MainC's Boot interface into it:

```
configuration MyAppC {  
}  
implementation {  
    components MyAppP;  
    components MainC;  
    ...  
    MyAppP.Boot -> MainC;  
}
```

```
module MyAppP {  
    uses interface Boot;  
}  
implementation {  
    event void Boot.booted() {  
        // Initialize app here  
    }  
    ...  
}
```



Slight Diversion #2: error_t Data Type

- TinyOS defines a special error_t data type that describes several different error codes
- Often given as return values to commands or event handlers
- Commonly used values:
 - ❑ SUCCESS (everything's OK)
 - ❑ FAIL (general error)
 - ❑ EBUSY (subsystem is busy with another request, retry later)
 - ❑ ERETRY (something weird happened, retry later)
- Others defined in `$TOSROOT/types/TinyError.h`



Message Addressing

- Each node has a unique 16-bit address (`am_addr_t`) specified by the make command
`make install.[address] platform`
- Two special address constants:
 - ❑ `TOS_BCAST_ADDR` (0xFFFF) is reserved for broadcast traffic
 - ❑ `TOS_NODE_ID` always refers to the node's own address
- Each message also has an 8-bit Active Message ID (`am_id_t`) analogous to TCP ports
 - ❑ Determines how host should handle received packets, not which host receives it



TinyOS Active Messages

- `message_t` structure defined in `$TOSROOT/tos/types/message.h`
- Each platform defines platform-specific header, footer, and metadata fields for the `message_t`
- Applications can store up to `TOSH_DATA_LENGTH` bytes payload in the data field (28 by default)

```
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;
```

Header

Payload (TOSH_DATA_LENGTH)

Footer

Metadata



Split-Phase Operation

- Many networking commands take a long time (ms) for underlying hardware operations to complete -- blocking would be bad
- TinyOS makes these long-lived operations split-phase

- ❑ Application issues
- ❑ An event is signaled

Error code here indicates whether TinyOS processing request

Error code here indicates whether TinyOS could *complete* processing request

```
interface SplitControl {  
  command error_t start();  
  event void startDone(error_t error);  
  
  command error_t stop();  
  event void stopDone(error_t error);  
}
```



Active Messaging Interfaces

```
interface AMSend {
```

```
    command error_t send(am_addr_t addr, message_t * msg,  
        uint8_t len);
```

```
    command error_t cancel(message_t * msg);
```

```
    event void sendDone(message_t * msg, error_t error);
```

```
    command uint8_t maxPayloadLength();
```

```
    command void* getPayload(message_t * msg, uint8_t len);
```

```
}
```

```
interface Receive {
```

```
    event message_t* receive(message_t * msg, void *  
        payload, uint8_t len);
```

```
}
```



Other Networking Interfaces

```
interface Packet {
```

```
    command void clear(message_t * msg);
```

```
    command void* getPayload(message_t * msg, uint8_t  
        len);
```

```
    command uint8_t payloadLength(message_t * msg);  
    command void setPayloadLength(message_t * msg, uint8_t  
        len);
```

```
    command uint8_t maxPayloadLength();
```

```
}
```



Other Networking Interfaces

```
interface AMPacket {
```

```
    command am_addr_t address();  
    command am_group_t localGroup();
```

```
    command am_addr_t destination(message_t* amsg);  
    command am_addr_t source(message_t* amsg);  
    command am_group_t group(message_t* amsg);  
    command bool isForMe(message_t* amsg);
```

```
    command am_id_t type(message_t* amsg);
```

```
}
```



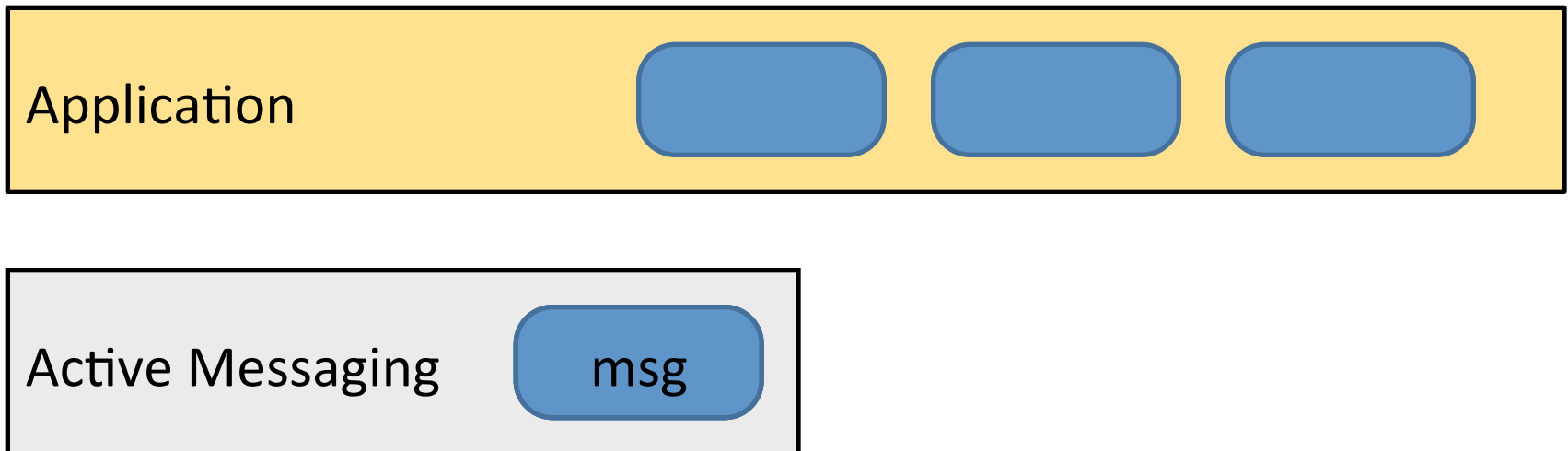
Other Networking Interfaces

```
interface PacketAcknowledgements {  
    async command error_t requestAck(message_t* msg);  
    async command error_t noAck(message_t* msg);  
    async command bool wasAcked(message_t* msg);  
}
```

- Default behavior: no ACKs
- Even with ACKs enabled, no automatic retransmissions
- Optional packet link layer can handle retransmissions;
#define PACKET_LINK and see TEP 127

Message Buffer Ownership

- Transmission: Radio driver gains ownership of the buffer until `sendDone(...)` is signaled
- Reception: Application's event handler gains ownership of the buffer, but it must return a free buffer for the next message



Network Types

- Radio standards like 802.15.4 mean that you could have communication among different types of motes with different CPUs
- nesC defines network types (`nx_uint16_t`, `nx_int8_t`, etc.) that transparently deal with endian issues for you
- nesC also defines an `nx_struct` analogous to C structs

```
typedef struct {  
    uint16_t field1;  
    bool field2;  
} bad_message_t;  
// Can have endianness problems  
// if sent to a host with a  
// different architecture
```

```
typedef nx_struct {  
    nx_uint16_t field1;  
    nx_bool field2;  
} good_message_t;  
// nesC will resolve endian  
// issues for you
```



Sending a Message

- First create a .h file with an nx_struct defining the message data format, and a unique active message ID (127–255)

```
enum {  
    AM_SENSORREADING = 240,  
};
```

```
typedef nx_struct sensor_reading {  
    nx_int16_t temperature;  
    nx_uint8_t humidity;  
} sensor_reading_t;
```



Sending a Message

- Declare a `message_t` variable in your module to store the packet's contents
- Get the packet's payload using the `Packet` interface; cast it to your message type; and store your data

```
implementation {
```

```
...
```

```
message_t output;
```

```
task void sendData() {
```

```
    sensor_reading_t * reading =  
        (sensor_reading_t *)call Packet.getPayload(&output,  
                                                    sizeof(sensor_reading_t));
```

```
    reading->temperature = lastTemperatureReading;  
    reading->humidity = lastHumidityReading;
```

```
...
```

```
}
```

```
}
```



Sending a Message

- Finally, use the AMSend interface to send the packet

```
task void sendData() {  
    ...  
  
    if(call AMSend.send(AM_BROADCAST_ADDR, &output,  
        sizeof(sensor_reading_t)) != SUCCESS)  
        post sendData();  
    // Try to send the message, and reschedule the task if it  
    // fails (e.g., the radio is busy)  
}
```



Sending a Message

- The AM subsystem will signal `AMSend.sendDone()` when the packet has been completely processed, successfully or not

```
event void AMSend.sendDone(message_t * msg, error_t err) {  
    if(err == SUCCESS) {  
        // Prepare next packet if needed  
    }  
    else {  
        post sendTask();  
        // Resend on failure  
    }  
}
```



Receiving a Message

- When messages with the correct AM ID are received, the Receive interface fires the `receive()` event

```
implementation {  
    ...  
    event message_t * Receive.receive(message_t * msg,  
        void * payload, uint8_t len) {  
        am_addr_t from = call AMPacket.source(msg);  
        sensor_reading_t * data = (sensor_reading_t *)payload;  
        ...  
        return msg;  
    }  
}
```



Networking Components

- Note that we didn't mention the packet's AM ID anywhere in the code
- That's because TinyOS includes generic components to manage the AM ID for you when you send/receive:

```
components new AMSenderC(AM_SENSORREADING);  
components new AMReceiverC(AM_SENSORREADING);
```

```
MyAppP.AMSend -> AMSenderC;
```

```
// AMSenderC provides AMSend interface
```

```
MyAppP.Receive -> AMReceiverC;
```

```
// AMReceiverC provides Receive interface
```

```
MyAppP.Packet -> AMSenderC;
```

```
MyAppP.AMPacket -> AMSenderC;
```

```
// AMSenderC and AMReceiverC provide Packet and AMPacket
```

```
// interfaces (pick one or the other)
```



Networking Components

- Before you can send/receive, you need to turn the radio on
- `ActiveMessageC` component provides a `SplitControl` interface to control the radio's power state

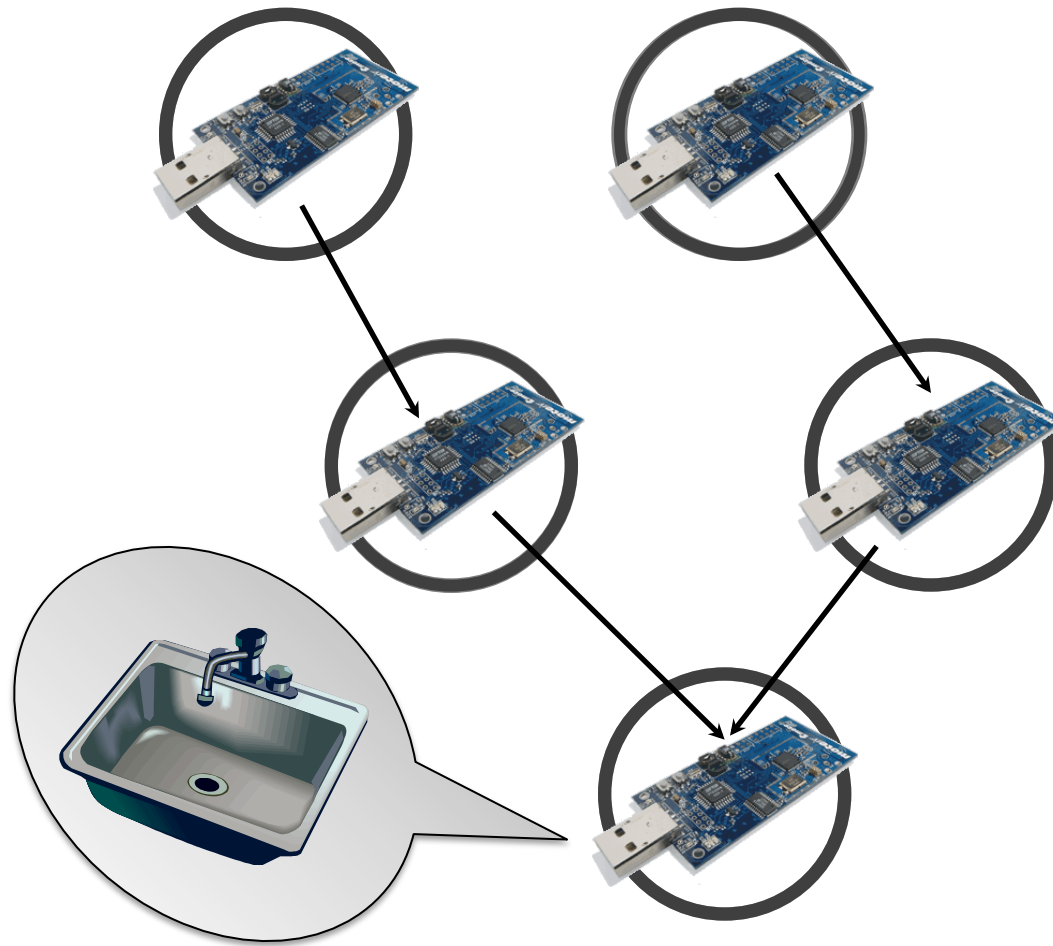
```
components ActiveMessageC;  
MyAppP.RadioPowerControl -> ActiveMessageC;
```



What About Multi-Hop?

- Until recently, TinyOS did not include a **general-purpose, point-to-point** multi-hop routing library
- Two special-purpose algorithms instead:
 - ❑ Collection Tree Protocol (CTP)
 - ❑ Dissemination
- Experimental TYMO point-to-point routing library added to TinyOS 2.1 (<http://docs.tinyos.net/index.php/Tymo>)
- blip: IPv6 stack added to TinyOS 2.1.1 ([http://docs.tinyos.net/index.php/BLIP Tutorial](http://docs.tinyos.net/index.php/BLIP_Tutorial))





Collection Tree Protocol (CTP)

Basic Operation



```

configuration MyCtpAppC {
}
implementation {
    components AppLogicP;
    components CollectionC;
    ...

    MyAppP.RoutingControl -> CollectionC;
    MyAppP.RootControl -> CollectionC;
    ...
}

```

```

module AppLogicP {
    uses interface StdControl as
        RoutingControl;
    uses interface RootControl;
    ...
}
implementation {
    ...

    event void RadioControl.startDone(
        error_t err) {
        ...
        if(TOS_NODE_ID == 100)
            call RootControl.setRoot();
            call RoutingControl.start();
        }

        ...
    }
}

```

Collection Tree Protocol (CTP)

Initializing CTP



```

configuration MyCtpAppC {
}
implementation {
    components AppLogicP;
    components CollectionC;
    ...

    MyAppP.Send -> CollectionC.
        Send[MY_MSG_ID];
    MyAppP.Receive -> CollectionC.
        Receive[MY_MSG_ID];
    MyAppP.Packet -> CollectionC;
    ...
}

```

```

module AppLogicP {
    ...
    uses interface Send;
    uses interface Receive;
    uses interface Packet;
    ...
}
implementation {
    ...

    task void sendPacket() {
        result_t err = call Send.send(
            &msg, sizeof(MyMsg));
        ...
    }

    event message_t * Receive.receive(
        message_t * msg, void * payload,
        uint8_t len) {
        // Only signaled on root node
        ...
    }
}

```

Collection Tree Protocol (CTP)

Sending/Receiving Packets

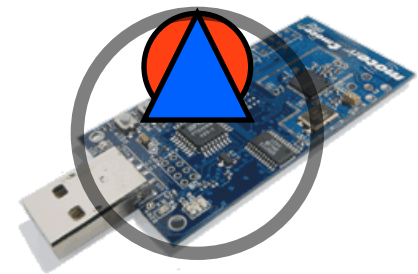
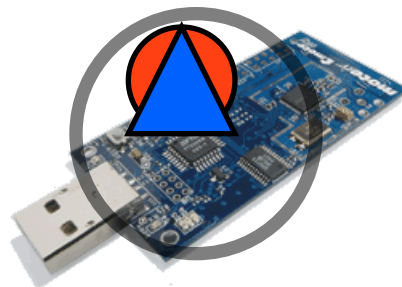
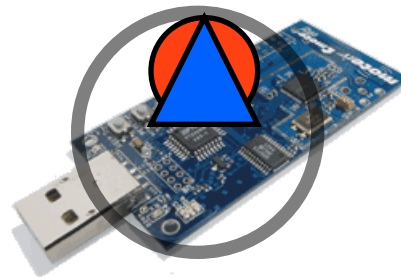
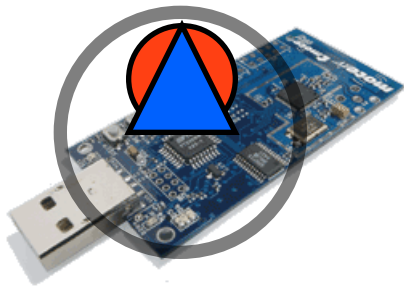


Collection Tree Protocol (CTP)

- To link into your app, include these lines in your Makefile:

```
CFLAGS += -I$(TOSDIR)/lib/net  
CFLAGS += -I$(TOSDIR)/lib/net/4bitle  
CFLAGS += -I$(TOSDIR)/lib/net/ctp
```

- CTP automatically turns on packet ACKs, retransmits up to 30 times at each hop
 - ❑ But no *end-to-end* acknowledgments;
PacketAcknowledgments.wasAcked() only tells you if the packet made it to the first hop



Dissemination

Basic Operation



For More Information

- TinyOS Tutorial 12: Network Protocols (http://docs.tinyos.net/index.php/Network_Protocols)
- TEP 123: Collection Tree Protocol (<http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html>)
- TEP 118: Dissemination (<http://www.tinyos.net/tinyos-2.x/doc/html/tep118.html>)



Sending Data to a PC

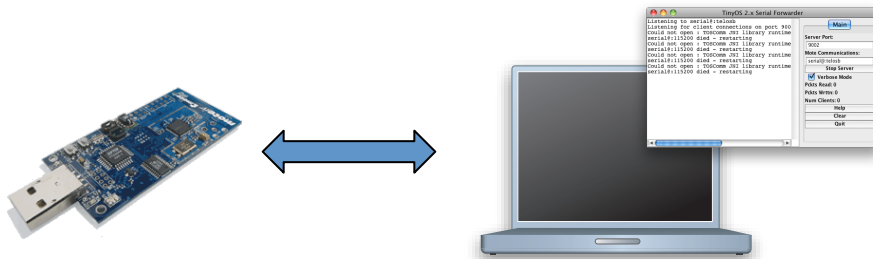
- TinyOS apps can also send or receive data over the serial/USB connection to an attached PC
- The SerialActiveMessageC component provides an Active Messaging interface to the serial port:

```
components SerialActiveMessageC;  
MyAppP.SerialAMSend ->  
    SerialActiveMessageC.Send[AM_SENSORREADING];  
MyAppP.SerialReceive ->  
    SerialActiveMessageC.Receive[AM_SENSORREADING];  
// SerialActiveMessageC provides parameterized AMSend and  
// Receive interfaces  
MyAppP.SerialPowerControl -> SerialActiveMessageC;
```



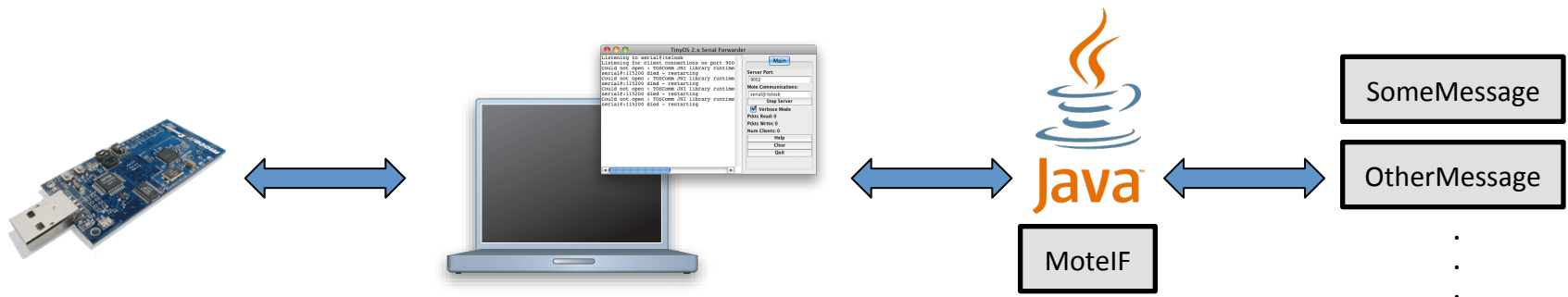
Interfacing With Motes

- TinyOS includes a Java-based SerialForwarder utility that implements PC side of TEP 113
 - ❑ `java net.tinyos.sf.SerialForwarder -comm serial@[port]:[speed]`
 - ❑ `[speed]` may be a specific baud rate or a platform name (e.g., `telosb`)
- Listens on TCP port and sends/receives TinyOS messages from local or remote applications



Interfacing With Motes

- Java SDK connects to SerialForwarder and converts TinyOS messages to/from native Java objects
- mig application auto-generates these classes from your app's header files
 - ❑ `mig java -java-classname=[classname] [header.h] [message-name] -o [classname].java`



SDK Support for Other Languages

➤ C/C++

- ❑ C reimplement of SerialForwarder (sf) and a few test apps found in `$TOSROOT/support/sdk/c/sf`
- ❑ Building sf also builds `libmote.a` for accessing the motes in your own code
- ❑ See `sfsource.h` and `serialsource.h` to get started



SDK Support for Other Languages

➤ Python

- ❑ Python classes in `$TOSROOT/support/sdk/python` closely mirror Java SDK
- ❑ Not completely stand-alone; Python MoteIF implementation talks to Java or C SerialForwarder
- ❑ See `tinyos/message/MoteIF.py` to get started

➤ C#

- ❑ `mig` can generate C# classes to parse/generate raw TinyOS packets
- ❑ But it's up to the user to actually get those packets from the serial port or SerialForwarder



CC2420Config Interface

```
interface CC2420Config {  
    command uint8_t getChannel();  
    command void setChannel(uint8_t channel);  
  
    async command uint16_t getShortAddr();  
    command void setShortAddr(uint16_t address);  
  
    async command uint16_t getPanAddr();  
    command void setPanAddr(uint16_t address);  
  
    command error_t sync();  
    event void syncDone(error_t error);  
}
```

(Provided by CC2420ControlC component)



Outline

- Installing TinyOS and Building Your First App
- Hardware Primer
- Basic nesC Syntax
- Advanced nesC Syntax
- Network Communication
- **Sensor Data Acquisition**
- Debugging Tricks and Techniques



Obtaining Sensor Data

- Each sensor has components that provides one or more split-phase Read interfaces

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

- Some sensor drivers provide additional interfaces for bulk (ReadStream) or low-latency (ReadNow) readings
 - ❑ See TEPs 101 and 114 for details

Sensor Reading Example

```
configuration MyAppC {  
}  
implementation {  
  components MyAppP;  
  ...  
}
```

```
components new AccelXC();  
// X axis accelerator component  
// defined by mts300 sensorboard  
MyAppP.AccelX -> AccelXC;
```

```
module MyAppP {  
  uses interface Read<uint16_t> as AccelX;  
  ...  
}  
implementation {  
  ...  
  task void readAccelX() {  
    if(call AccelX.read()) != SUCCESS  
      post readAccelX();  
  }  
  event void AccelX.readDone(error_t err,  
    uint16_t reading) {  
    if(err != SUCCESS) {  
      post readAccelX();  
      return;  
    }  
    // Handle reading here  
  }  
  ...  
}
```



Sensor Components

- Sensor components are stored in:
 - ❑ `$TOSROOT/tos/platform/[platform]` (for standard sensors)
 - Note that `telosb` “extends” `telosa`, so look in both directories if you’re using a TelosB or Tmote Sky mote!
 - ❑ `$TOSROOT/tos/sensorboard/[sensorboard]` (for add-on sensor boards)

- Additional sensor board components may be available from TinyOS CVS in `tinys-2.x-contrib`
 - ❑ Unfortunately, some third-party sensor board drivers have yet to be ported from TinyOS 1.x to 2.x



External Sensors

```

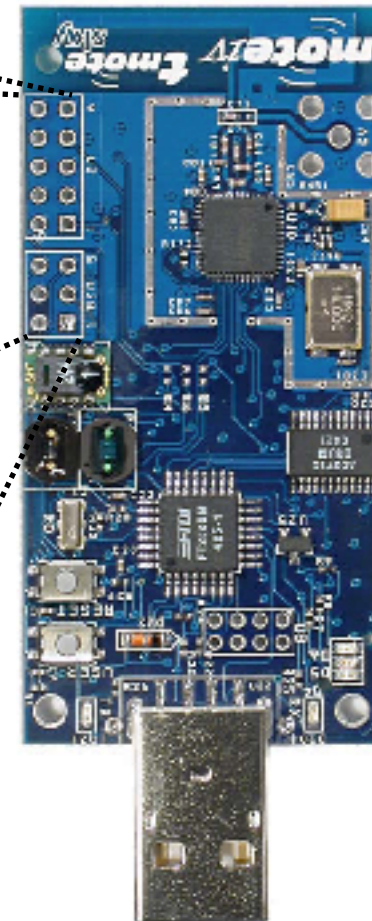
interface HplMsp430GeneralIO {
  command void makeInput();
  command void makeOutput();

  Analog Input 0 (ADC0)
  Analog Input 1 (ADC1)
  command bool get();
  Analog Input 2 (ADC2)
  Exclusive Digital I/O 1 (GIO1)
  Analog Ground (Gnd)
  command void clr();
  command void set();
  command void toggle();
}
  
```

Analog VCC (AVcc)	1	2	UART Receive (UART0RX)
Analog Input 0 (ADC0)	3	4	UART Transmit (UART0TX)
Analog Input 1 (ADC1)	5	6	I2C Clock (I2C_SCL)
Analog Input 2 (ADC2)	7	8	Shared Digital I/O 4 (GIO4)
Exclusive Digital I/O 1 (GIO1)			I2C Data (I2C_SDA)
Analog Ground (Gnd)	9	10	Shared Digital I/O 5 (GIO5)
			Analog Input 3 (ADC3)
			Exclusive Digital I/O 0 (GIO0)

Figure 20 : Functionality of the 10-pin expansion connector (U2).
Alternative pin uses are shown in gray.

Analog Input 6 (ADC6)	1	2	Analog Input 7 (ADC7)
DAC0			DAC 1 / SVS in
Exclusive Digital I/O 2 (GIO2)	3	4	Exclusive Digital I/O 3 (GIO3)
Timer A Capture (TA1)			External DMA Trigger (DMAE0)
User Interrupt (UserInt)	5	6	Reset



External Sensors

- Digital I/O: wire directly into HplMsp430GeneralIOC component

```
component HplMsp430GeneralIOC {  
    provides interface HplMsp430GeneralIO as ADC0;  
    provides interface HplMsp430GeneralIO as ADC1;  
    provides interface HplMsp430GeneralIO as ADC2;  
    provides interface HplMsp430GeneralIO as ADC3;  
    provides interface HplMsp430GeneralIO as ADC4;  
    provides interface HplMsp430GeneralIO as ADC5;  
    provides interface HplMsp430GeneralIO as ADC6;  
    provides interface HplMsp430GeneralIO as ADC7;  
    provides interface HplMsp430GeneralIO as DAC0;  
    provides interface HplMsp430GeneralIO as DAC1;  
    ...  
}
```

- I²C: read TEP 117 (Low-Level I/O)
- Analog I/O: read TEP 101 (Analog-to-Digital Converters)

Outline

- Installing TinyOS and Building Your First App
- Hardware Primer
- Basic nesC Syntax
- Advanced nesC Syntax
- Network Communication
- Sensor Data Acquisition
- **Debugging Tricks and Techniques**

Hard-Learned Lessons

- Be sure to check return values -- don't assume SUCCESS!
 - ❑ At the very least, set an LED when something goes wrong

- The TinyOS toolchain doesn't always warn about overflowing integers

```
uint8_t i;  
for(i = 0; i < 1000; i++) { ... }  
// This loop will never terminate
```

- Not all the Tmote Sky motes have sensors



msp430-gcc Alignment Bugs

- If you're unlucky, msp430-gcc will crash with internal errors like these:

```
/opt/tinyos-2.x/tos/interfaces/TaskBasic.nc: In function `SchedulerBasicP$TaskBasic
$runTask':
/opt/tinyos-2.x/tos/interfaces/TaskBasic.nc:64: unable to generate reloads for:
(call_insn 732 3343 733 (set (reg:SI 15 r15)
      (call (mem:HI (symbol_ref:HI ("AsyncQueueC$1$Queue$dequeue")) [0 S2 A8])
        (const_int 0 [0x0])))) 14 {*call_value_insn} (nil)
      (nil)
      (nil))
/opt/tinyos-2.x/tos/interfaces/TaskBasic.nc:64: Internal compiler error in
find_reloads, at reload.c:3590
```

- It's almost always because of *alignment* bugs (msp430-gcc doesn't always like it when fields straddle 16-bit boundaries)

```
typedef nx_struct my_msg
{
    nx_uint8_t field1;
    nx_uint8_t pad;
    nx_uint16_t field2;
} my_msg_t;
```

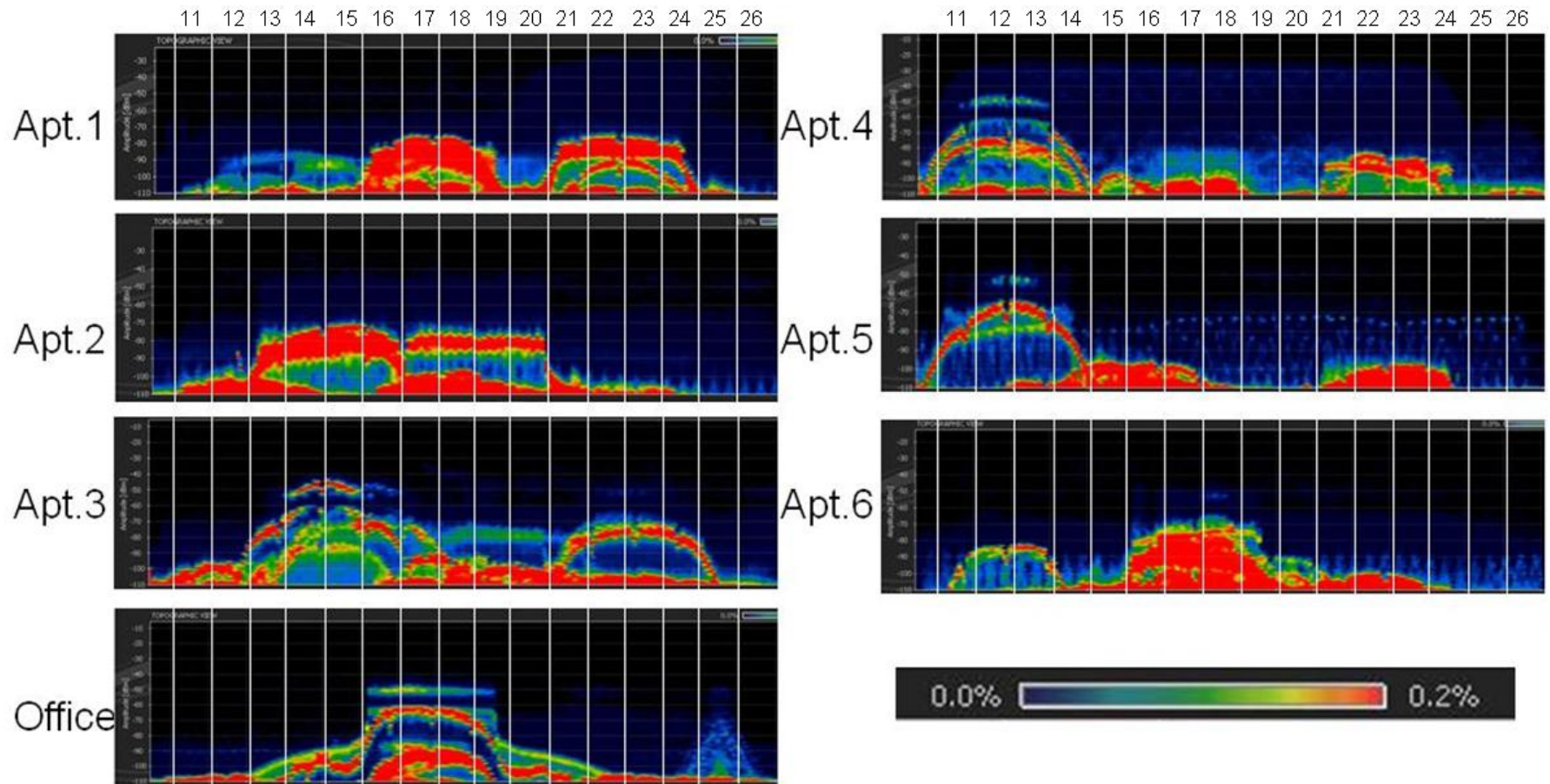


802.15.4 Radio Channels

- The CC2420 chip on the Tmote and MicaZ supports 802.15.4 channels 11 - 26
- 802.15.4 uses 2.4 GHz spectrum
- This can lead to interference between motes and with 802.11, Bluetooth, and all sorts of other things



802.15.4 Radio Channels



802.15.4 Radio Channels

- If you're seeing weird network behavior, set your CC2420 channel to something else:
 - ❑ Defaults to 26
 - ❑ Command-line: `CC2420_CHANNEL=xx make ...`
 - ❑ Makefile: `PFLAGS = -DCC2420_DEF_CHANNEL=xx`



Active Message Groups

- To avoid address collision with other applications or networks, you can also change the AM group:
 - ❑ Defaults to 0x22
 - ❑ Makefile: `DEFAULT_LOCAL_GROUP=xx` (any 16-bit value)
- On 802.15.4 compliant chips, maps to PAN ID
- Does not prevent *physical* interference of packets: only instructs radio chip/driver to filter out packets addressed to other groups



LEDs

- The easiest way to display runtime information is to use the mote's LEDs:

```
interface Leds {  
    async command void led0On();  
    async command void led0Off();  
    async command void led0Toggle();  
    async command void led1On();  
    async command void led1Off();  
    async command void led1Toggle();  
    async command void led2On();  
    async command void led2Off();  
    async command void led2Toggle();  
    async command uint8_t get();  
    async command void set(uint8_t val);  
}
```

- Provided by the components LedsC and NoLedsC



printf()

- You can use `printf()` to print debugging messages to the serial port
- Messages are buffered and sent to serial port in bulk; `printf fflush()` asks TinyOS to flush buffer
- **DON'T USE `printf()` FOR CRITICAL MESSAGES**
- When its buffer fills up, `printf()` starts throwing away data

printf()

- To enable the printf library, add the following line to your Makefile:

```
CFLAGS += -I$(TOSDIR)/lib/printf
```

- Note: this automatically turns on SerialActiveMessageC subsystem

- Included PrintfClient utility displays printed messages to console

```
java net.tinyos.tools.PrintfClient  
  [-comm serial@[port]:[speed]]
```

BaseStation

- The BaseStation app in \$TOSROOT/apps/BaseStation will sniff all wireless traffic and forward it to the serial port
- Listen tool prints hex-dump of packets to console:

```
java net.tinyos.tools.Listen  
  [-comm serial@[port]:[speed]]
```
- Extremely helpful for figuring out what data is being sent!



gdb

- The CPU on the Tmote Sky motes supports interactive debugging using gdb
- Set breakpoints, inspect the contents of variables, etc.
- The catch: it needs a special cable and modified motes -- and they don't make the motes anymore
 - ❑ We have 5 motes and one cable



TOSSIM

- make `micaz sim` compiles application to native C code for your own machine, which can be loaded into Python or C++ simulator (“TOSSIM”)
- Good way to rapidly test application logic, at the cost of some realism
 - ❑ e.g., does not emulate sensing and does not reproduce timing of real microcontrollers
- Besides app code, need two configuration details:
 - ❑ *Topology* of simulated network
 - ❑ *Noise trace* from simulated environment



TOSSIM Configuration: Topology

- List of links in the network and associated gain (signal strength in dBm)

- Several sources:

- ❑ Real measurements
- ❑ Samples included in TinyOS (`$TOSDIR/lib/tossim/topologies`)
- ❑ Generate one based on various parameters (<http://www.tinyos.net/tinyos-2.x/doc/html/tutorial/usc-topologies.html>)

0	1	-90.80
1	0	-95.95
0	2	-97.48
2	0	-102.10
0	3	-111.33
3	0	-115.49
0	4	-104.82
4	0	-110.09
...		

(from 15-15-sparse-mica2-grid.txt)

TOSSIM Configuration: Noise Trace

- Trace of ambient noise readings in dBm
 - Must contain at least 100 entries; more is better, but RAM consumption increases with larger traces
 - 39
 - 98
 - 98
 - Two sources:
 - ❑ Real measurements
 - 98
 - 99
 - ❑ Samples included in TinyOS (`$TOSDIR/lib/tossim/noise`)
 - 98
 - 94
 - 98
- ...

(from meyer-heavy.txt)

Other TOSSIM Features

- Log debug messages to console or to a file
- Inject packets into network
- Debugging support
 - ❑ Python TOSSIM: read variables' contents
 - ❑ C++ TOSSIM: use gdb
- TOSSIM Live fork: TOSSIM acts as SerialForwarder, send/receive serial packets to simulated motes
 - ❑ http://docs.tinyos.net/index.php/TOSSIM_Live
- See TinyOS Tutorial 11 for more details



Avrora + MSPsim

- Avrora: cycle-accurate Mica2 and MicaZ emulator
<http://compilers.cs.ucla.edu/avrora/>
- MSPsim: MSP430 (TelosB) emulator
<http://www.sics.se/project/mspsim/>
- Profile and benchmark apps, monitor packet transmissions, or interface with gdb
- Slower than TOSSIM, but highly accurate



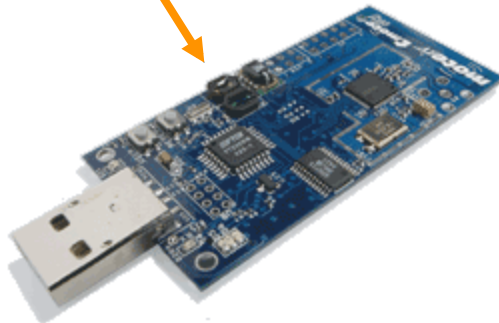
Safe TinyOS

- New in TinyOS 2.1: make [platform] safe
- Augments code to enforce pointer and type safety at runtime (bad casts, out-of-bounds array accesses, NULL pointer dereferences, etc.)
- When safety violations detected, LEDs blink error code
- <http://www.cs.utah.edu/~coop/safetinyos/>

Nathan Coopridier, Will Archer, Eric Eide, David Gay, and John Regehr, "Efficient Memory Safety for TinyOS," Proceedings of 5th ACM Conference on Embedded Networked Sensor Systems (SenSys 2007), 2007.



Demo: Putting it All Together



Demo Example

Three files:

1. Makefile

2. DemoMessage.h

3. DemoP.nc

4. DemoAppC.nc



Makefile

```
COMPONENT=DemoAppC  
include $(MAKERULES)
```



DemoMessage.h

```
#ifndef __DEMOMESSAGE_H
#define __DEMOMESSAGE_H

enum
{
    AM_DEMO_MSG = 231,
};

typedef nx_struct demo_msg
{
    nx_uint16_t lastReading;
} demo_msg_t;

#endif
```



DemoP.nc

```
module DemoP  
{  
    uses interface Boot;  
  
    uses interface Leds;  
  
    uses interface Read<uint16_t>;  
  
    uses interface SplitControl as RadioControl;  
    uses interface AMSend;  
    uses interface Receive;  
    uses interface Packet;  
  
    uses interface Timer<TMilli>;  
  
}
```



DemoP.nc

implementation

```
{  
  message_t buf;  
  task void readSensor();  
  task void sendBuffer();  
  event void Boot.booted()  
  {  
    if(call RadioControl.start() != SUCCESS)  
      call Leds.led0On();  
  }  
  
  event void RadioControl.startDone(error_t err)  
  {  
    if(err != SUCCESS)  
      call Leds.led0On();  
  
    if(TOS_NODE_ID == 0)  
      call Timer.startPeriodic(64);  
  }  
}
```

DemoP.nc

```
event void Timer.fired()
{
    post readSensor();
}
```

```
task void readSensor()
{
    if(call Read.read() != SUCCESS)
        post readSensor();
}
```

```
event void Read.readDone(error_t err, uint16_t val)
{
    demo_msg_t * payload = (demo_msg_t *)call Packet.getPayload(&buf, sizeof(
    payload->lastReading = val;

    post sendBuffer();
}
```

DemoP.nc

```
task void sendBuffer()
{
  if(call AMSend.send(AM_BROADCAST_ADDR,
    &buf, sizeof(demo_msg_t)) != SUCCESS)
    post sendBuffer();
}

event void AMSend.sendDone(message_t * jkdsakljads, error_t err)
{
  if(err != SUCCESS)
    post sendBuffer();
}

event message_t * Receive.receive(message_t * m,void * payload,uint8_t size)
{
  demo_msg_t * dpayload = (demo_msg_t *)payload;
  call Leds.set(dpayload->lastReading / 200);

  return m;
}

event void RadioControl.stopDone(error_t err) {}
```

DemoAppC.nc

```
#include "DemoMessage.h"
configuration DemoAppC{}
implementation{
    components DemoP, MainC;
    DemoP.Boot -> MainC.Boot;
    components LedsC;
    DemoP.Leds -> LedsC;
    components new HamamatsuS10871TsrC() as PhotoSensor;
    DemoP.Read -> PhotoSensor;
    components ActiveMessageC;
    DemoP.RadioControl -> ActiveMessageC;
    components new AMSenderC(AM_DEMO_MSG),
new AMReceiverC(AM_DEMO_MSG);
    DemoP.AMSend -> AMSenderC;
    DemoP.Receive -> AMReceiverC;
    DemoP.Packet -> AMSenderC;
    components new TimerMilliC();
    DemoP.Timer -> TimerMilliC;
}
```