TinyOS 2.1

IPSN 2009

Stephen Dawson-Haggerty, Omprakash Gnawali,
David Gay, Philip Levis, Răzvan Musăloiu-E.,
Kevin Klues, and John Regehr

San Francisco, CA - April 16, 2009

Agenda

- 8:33: Overview (Om)
- 8:40: Basics (Phil and David)
- 9:30: TOSSIM (Razvan)
- 9:45: Safe TinyOS (John)
- 10:00: Threads (Kevin)
- 10:15: break
- 10:20: Protocols (Om)
- 10:40: Upcoming (Stephen)
- 10:50: Hands-on (Razvan, Om, et al.)
- 11:30: End

What?

- An operating system for low power, embedded, wireless devices
  - Wireless sensor networks (WSNs)
  - Sensor-actuator networks
  - Embedded robotics
- Open source, open developer community
- http://www.tinyos.net

Who are we?

- Some principal developers and designers
  - Stephen Dawson-Haggerty: network protocols
  - David Gay: language design
  - Omprakash Gnawali: network protocols
  - Kevin Klues: core system
  - Philip Levis: core system
  - Răzvan Musăloiu-E.: network protocols
  - John Regehr: compilation tools
- There are many contributors besides us, they all deserve credit

Why?

- TinyOS is very powerful
  - Modern operating system and language techniques in an embedded system
  - A lot of libraries, support code, and community development
- TinyOS has a steep learning curve
  - It can take time to use all of its capabilities
- Give a jump-start on high level concepts and how to write applications

Goals

- Give you a high-level understanding of TinyOS's structure and ideas
- Explain how to build and install applications
- Survey important libraries
  - Focus on very recent additions
- Give you the experience of writing a networked sensing application

Schedule

- 8:33: Overview (Om)
- 8:40: Basics (Phil and David)
- 9:30: TOSSIM (Razvan)
- 9:45: Safe TinyOS (John)
- 10:00: Threads (Kevin)
- 10:15: break
- 10:20: Protocols (Om)
- 10:40: Upcoming (Stephen)
- 10:50: Hands-on (Razvan, Om, et al.)
- 11:30: End

Basics

Philip Levis (Stanford)

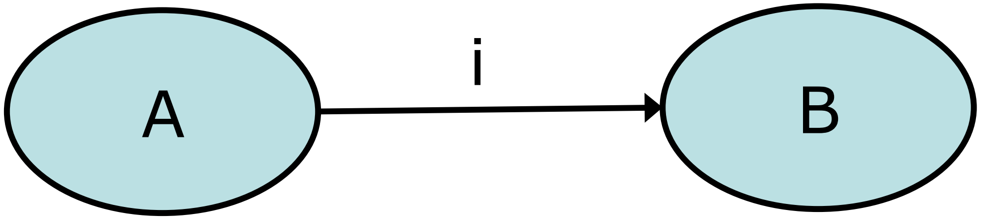
David Gay (Intel Research)

Outline

- *Components and interfaces*
  - Basic example
- Tasks
  - More complex example
- Compiling and toolchain

TinyOS Components

- TinyOS and its applications are in nesC
  - C dialect with extra features
- Basic unit of nesC code is a component
- Components connect via interfaces
  - Connections called "wiring"

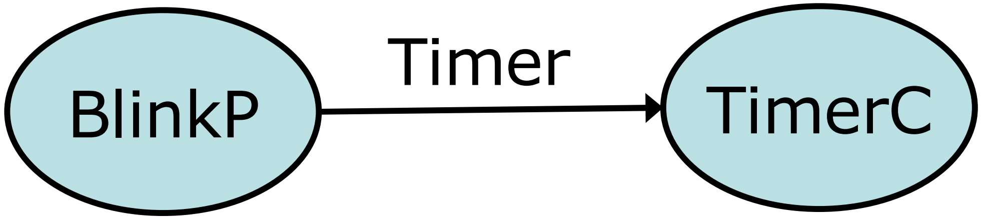


Components

- A component is a file (names must match)
- Modules are components that have variables and executable code
- Configurations are components that wire other components together

Component Example

- BlinkC wires BlinkP.Timer to TimerC.Timer



```
module BlinkP { … }
implementation {
  int c;
  void increment() {c++;}
}
```

```
configuration BlinkC { … }
implementation {
  components new TimerC();
  components BlinkC;

  BlinkC.Timer -> TimerC;
}
```

Singletons and Generics

- Singleton components are unique: they exist in a global namespace
- Generics are instantiated: each instantiation is a new, independent copy

```
configuration BlinkC { … }
implementation {
  components new TimerC();
  components BlinkC;

  BlinkC.Timer -> TimerC;
}
```

Interfaces

- Collections of related functions
- Define how components connect
- Interfaces are bi-directional: for A->B
  - Commands are from A to B
  - Events are from B to A
- Can have parameters (types)

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

Outline

- Components and interfaces
  - *Basic example*
- Tasks
  - More complex example
- Compiling and toolchain

Basic Example

- Goal: write an anti-theft device. Let's start simple.
- Two parts:
  - Detecting theft.
    - Assume: thieves put the motes in their pockets.
    - So, a "dark" mote is a stolen mote.
    - Every N ms check if light sensor is below some threshold
  - Reporting theft.
    - Assume: bright flashing lights deter thieves.
    - Theft reporting algorithm: light the **red LED** for a little while!
- What we'll see
  - Basic components, interfaces, wiring
  - Essential system interfaces for startup, timing, sensor sampling

The Basics – Let’s Get Started

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<TMilli> as Check;
  uses interface Read<uint16_t>;
}
implementation {
  event voi
    call Che
  }
  event voi
    call Rea
  }
  event voi
    if (ok == SUCCESS && val < 20
      theftLed();
  }
}
```

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

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

Components start with a *signature* specifying
- the interfaces *provided* by the component
- the interfaces *used* by the component

A module is a component implemented in C
- with functions implementing commands and events
- and extensions to call commands, events

The Basics – Split-Phase Ops

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

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

- A command starts the op: read
- An event signals op completion: readDone

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

The Basics – Split-Phase Ops

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

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

- A command starts the op: read
- An event signals op completion: readDone

Errors are signalled using the error_t type, typically

- Commands only allow one outstanding request
- Events report any problems occurring in the op

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

The Basics – Configurations

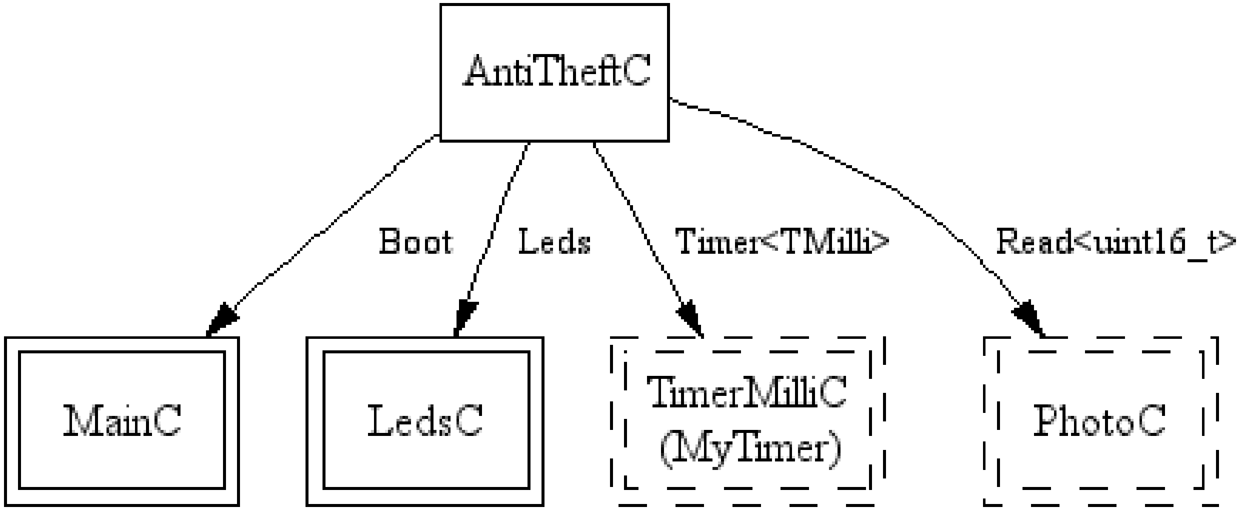
```
configuration AntiTheftAppC { }
implementation
{
  components AntiThe

  AntiTheftC.Boo
  AntiTheftC.Led

  components ne
  AntiTheftC.Che

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

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

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

A configuration is a component built out of other components.
It *wires* “used” to “provided” interfaces.
It can instantiate *generic* components
It can itself provide and use interfaces

Components


AntiTheftC
Boot
Leds
Timer<TMilli>
Read<uint16_t>
MainC
LedsC
TimerMilliC
(MyTimer)
PhotoC

Outline

- Components and interfaces
  - Basic example
- *Tasks and concurrency*
  - More complex example
- Compiling and toolchain

Tasks

- TinyOS has a single stack: long-running computation can reduce responsiveness
- Tasks: mechanism to defer computation
  - Tells TinyOS “do this later”
- Tasks run to completion
  - TinyOS scheduler runs them one by one in the order they post
  - Keep them short!
- Interrupts run on stack, can post tasks

Outline

- Components and interfaces
  - Basic example
- Tasks and concurrency
  - *More complex example*
- Compiling and toolchain

More Complex Application

- Let’s improve our anti-theft device. A clever thief could still steal our motes by keeping a light shining on them!
  - But the thief still needs to pick up a mote to steal it.
  - Theft Detection Algorithm 2: Every N ms, sample acceleration at 100Hz and check if variance above some threshold
- What we’ll see
  - (Relatively) high frequency sampling support
  - Use of tasks to defer computation-intensive activities
  - TinyOS execution model

Advanced Sensing, Tasks

```
uses interface ReadStream;
uint16_t accelSamples[ACCEL_SAMPLES];
event void Timer.fired() {
  call ReadStream.postBuffer(accelSamples, ACCEL_SAMPLES);
  call ReadStream.read(10000);
}

event void ReadStream.readD
  if (ok == SUCCESS)
    post checkAcceleration();
}

task void checkAcceleration() {
    ... check acceleration and re
}
```

ReadStream is an interface for periodic sampling of a sensor into one or more buffers.

- postBuffer adds one or more buffers for sampling
- read starts the sampling operation
- readDone is signalled when the last buffer is full

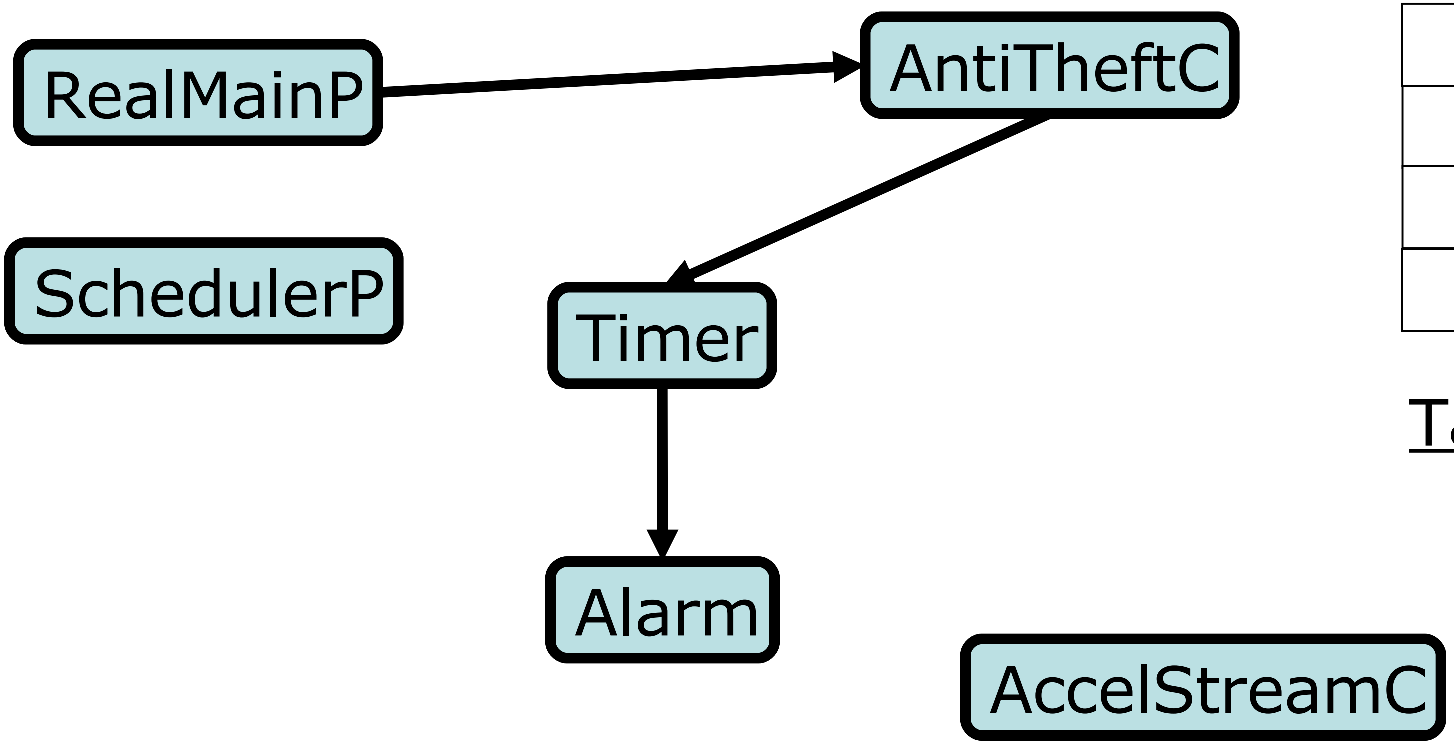
```
interface ReadStream<val_t> {
  command error_t postBuffer(val_t* buf, uint16_t count);
  command error_t read(uint32_t period);
  event void readDone(error_t ok, uint32_t actualPeriod);
}
```

Advanced Sensing, Tasks

```
uint16_t accelSamples[SAMPLES];
event void ReadStream.readDone(error_t ok, uint32_t actualPeriod) {
  if (ok == SUCCESS)
    post checkAcceleration();
}
task void checkAcceleration() {
   uint16_t i, avg, var;

   for (avg = 0, i = 0; i < SAMPLES; i++)
     avg += accelSamples[i];
   avg /= SAMPLES;

   for (var = 0, i = 0; i < SAMPL
     {
       int16_t diff = accelSample
       var += diff * diff;
     }
   if (var > 4 * SAMPLES) theftL
}
```

In readDone, we need to compute the variance of the sample. We defer this “computationally-intensive” operation to a separate *task*, using post. We then compute the variance and report theft.

TinyOS Execution Model



Stack

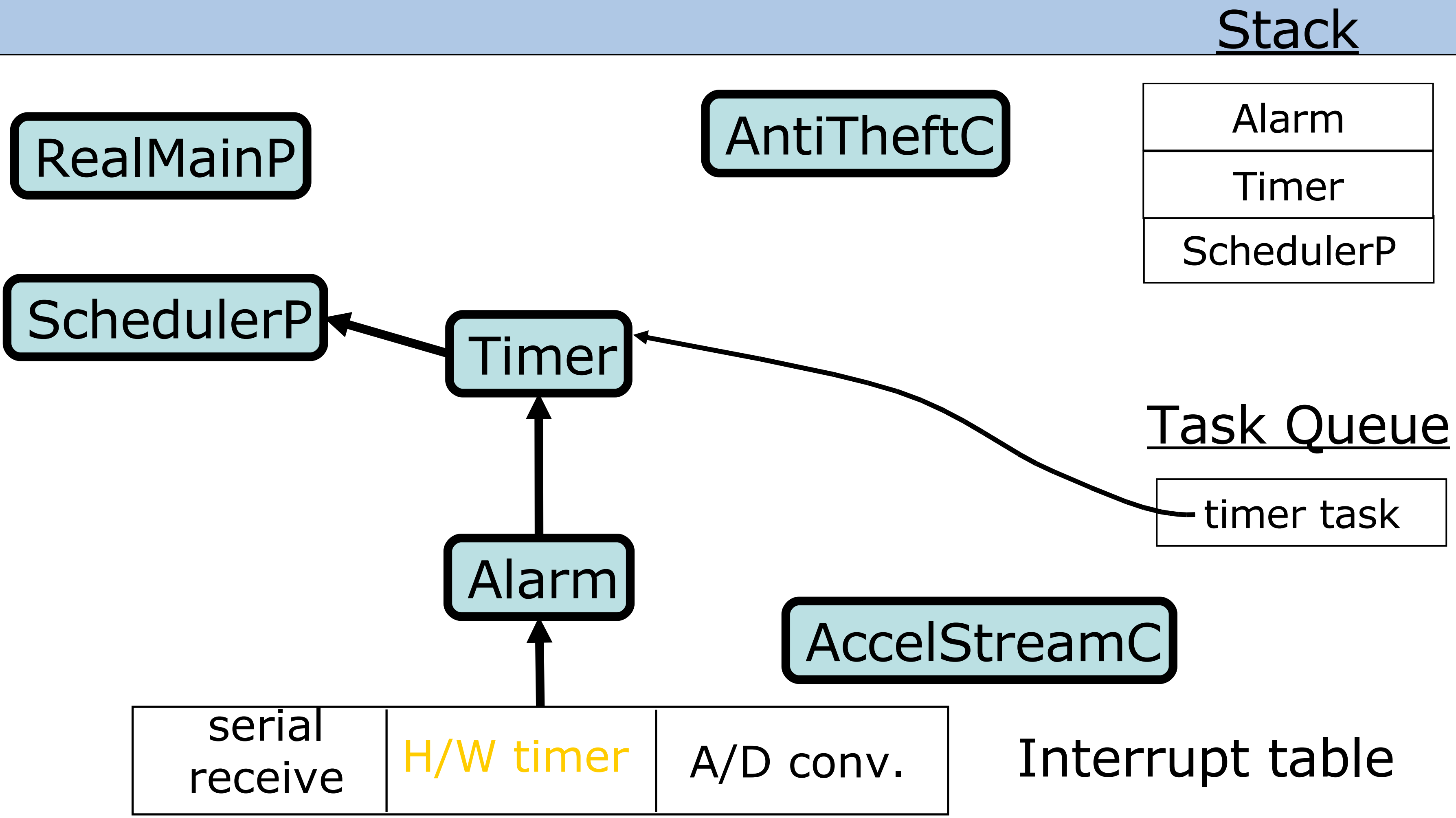
RealMainP
AntiTheftC
Timer
Alarm

Task Queue

serial receive	H/W timer	A/D conv.

Interrupt table

TinyOS Execution Model



RealMainP

AntiTheftC

SchedulerP

Timer

Alarm

AccelStreamC

Stack

- Alarm
- Timer
- SchedulerP

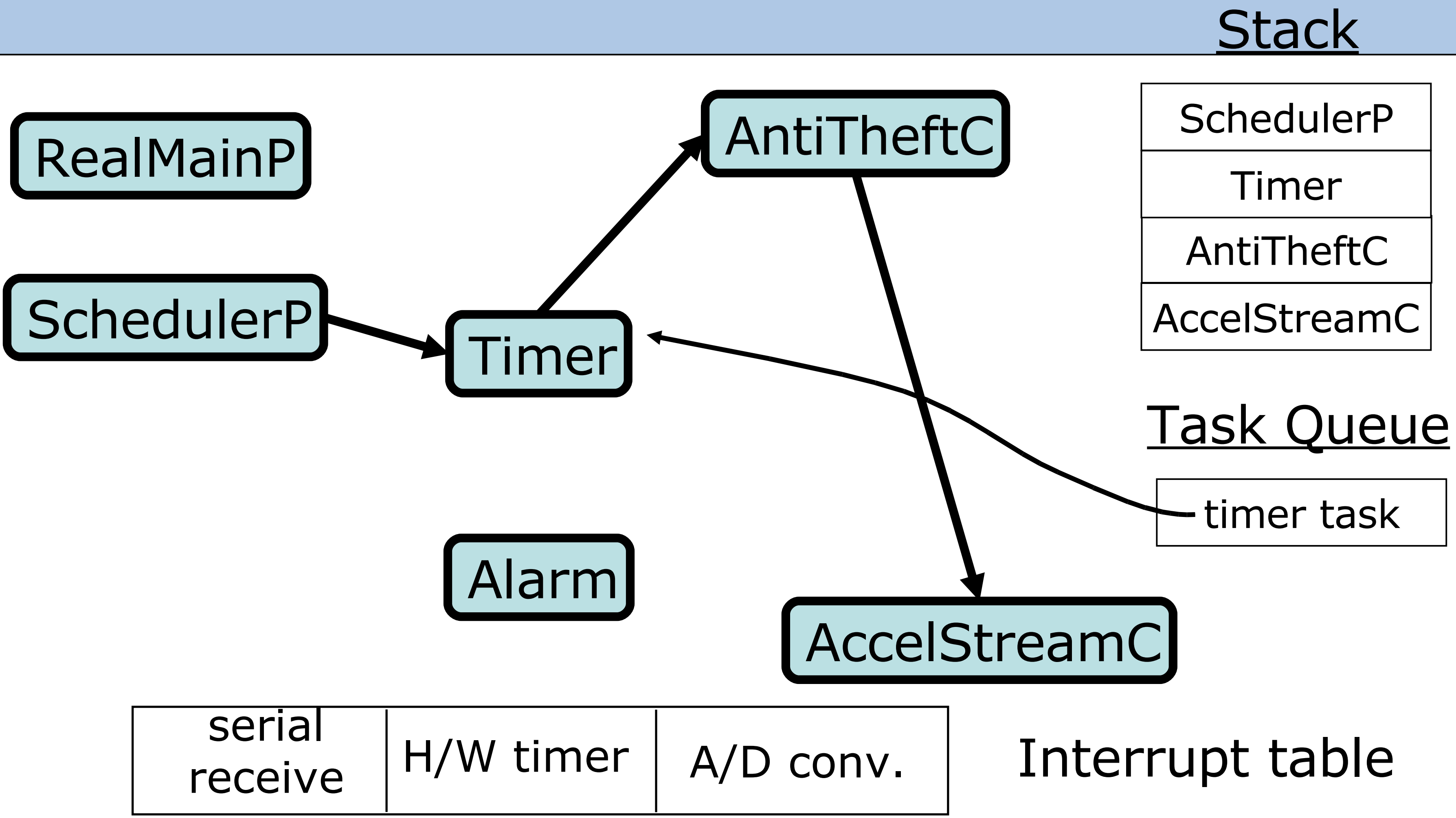
Task Queue

- timer task

Interrupt table

serial receive	H/W timer	A/D conv.

TinyOS Execution Model



Stack

SchedulerP

Timer

AntiTheftC

AccelStreamC

Task Queue

timer task

serial receive	H/W timer	A/D conv.

Interrupt table

Networking – “External” Types

```
#include “antitheft.h”
module AntiTheftC {
  ... uses interface DisseminationValue<settings_t> as SettingsValue;
gsValue.get();
ckInterval);
   if (settings.detect & DETECT_
      call ReadStream.postBuffe
      call ReadStream.read(1000
   }
 }
```

```
#ifndef ANTITHEFT_H
#define ANTITHEFT_H
typedef nx_struct {
  nx_uint8_t alert, detect;
  nx_uint16_t checkInterval;
} settings_t;
#endif
```

External types (nx_...) provide C-like access, but:

- platform-independent layout and endianness gives interoperability
- no alignment restrictions means they can easily be used in network buffers
- compiled to individual byte read/writes

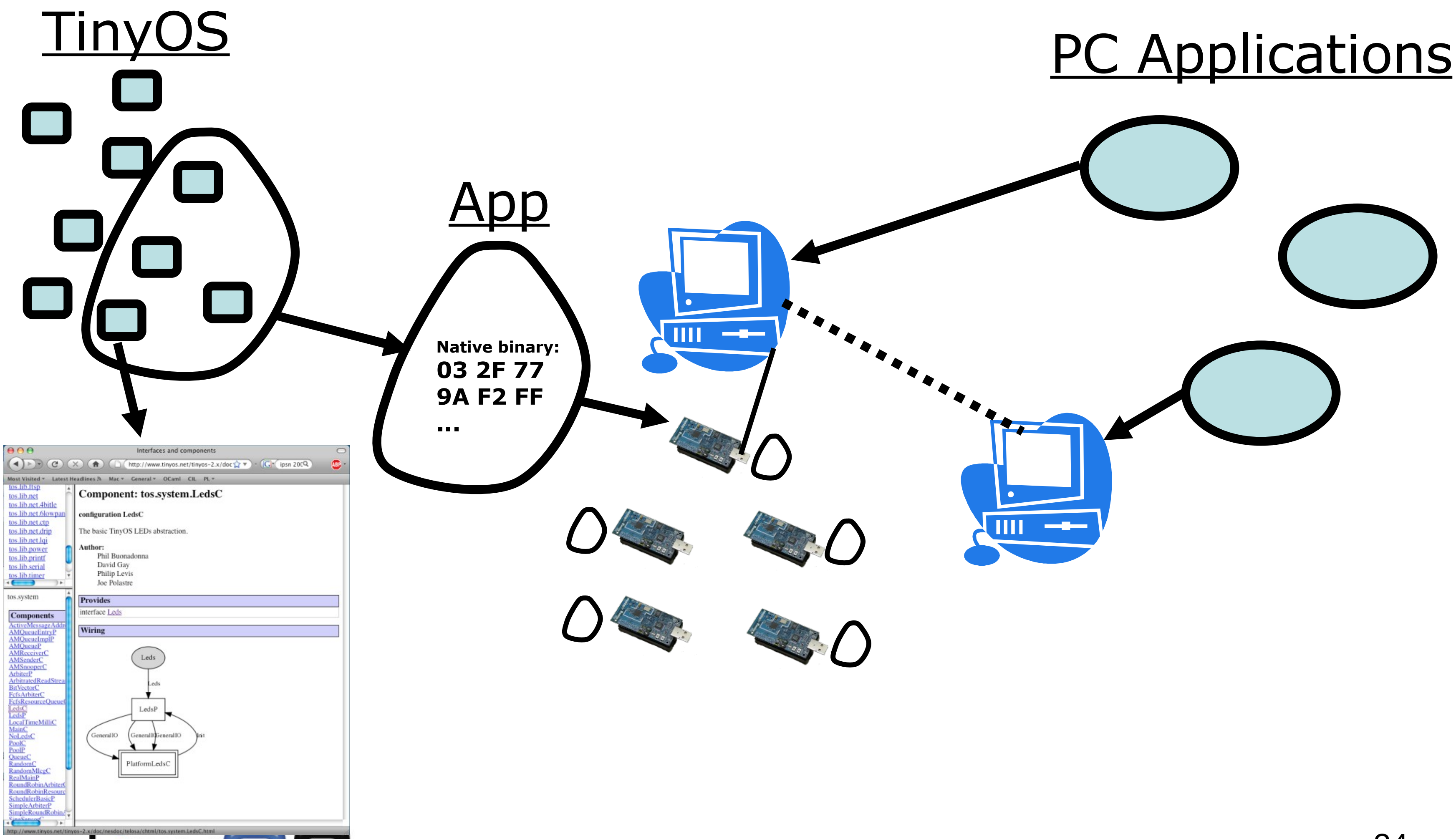
TinyOS/nesC Summary

- Components and Interfaces
  - Programs built by writing and wiring components
    - modules are components implemented in C
    - configurations are components written by assembling other components
- Execution model
  - Execution happens in a series of tasks (atomic with respect to each other) and interrupt handlers
  - No threads
- System services: startup, timing, sensing (so far)
  - (Mostly) represented by instantiatable generic components
    - This instantiation happens at compile-time! (think C++ templates)
  - All slow system requests are split-phase

Outline

- Components and interfaces
  - Basic example
- Tasks
  - More complex example
- *Compiling and toolchain*

The Toolchain



TinyOS

PC Applications

App

Native binary:
03 2F 77
9A F2 FF
...

Interfaces and components

http://www.tinyos.net/tinyos-2.x/doc

tos.lib.ftsp
tos.lib.net
tos.lib.net.4bitle
tos.lib.net.6lowpan
tos.lib.net.ctp
tos.lib.net.drip
tos.lib.net.lqi
tos.lib.power
tos.lib.printf
tos.lib.serial
tos.lib.timer

tos.system

Components

ActiveMessageAddr
AMQueueEntryP
AMQueueImplP
AMQueueP
AMReceiverC
AMSenderC
AMSnooperC
ArbiterP
ArbitratedReadStrea
BitVectorC
FcfsArbiterC
FcfsResourceQueue
LedsC
LedsP
LocalTimeMilliC
MainC
NoLedsC
PoolC
PoolP
QueueC
RandomC
RandomMlcgC
RealMainP
RoundRobinArbiter
RoundRobinResourc
SchedulerBasicP
SimpleArbiterP
SimpleRoundRobin

Component: tos.system.LedsC

configuration LedsC

The basic TinyOS LEDs abstraction.

Author:
Phil Buonadonna
David Gay
Philip Levis
Joe Polastre

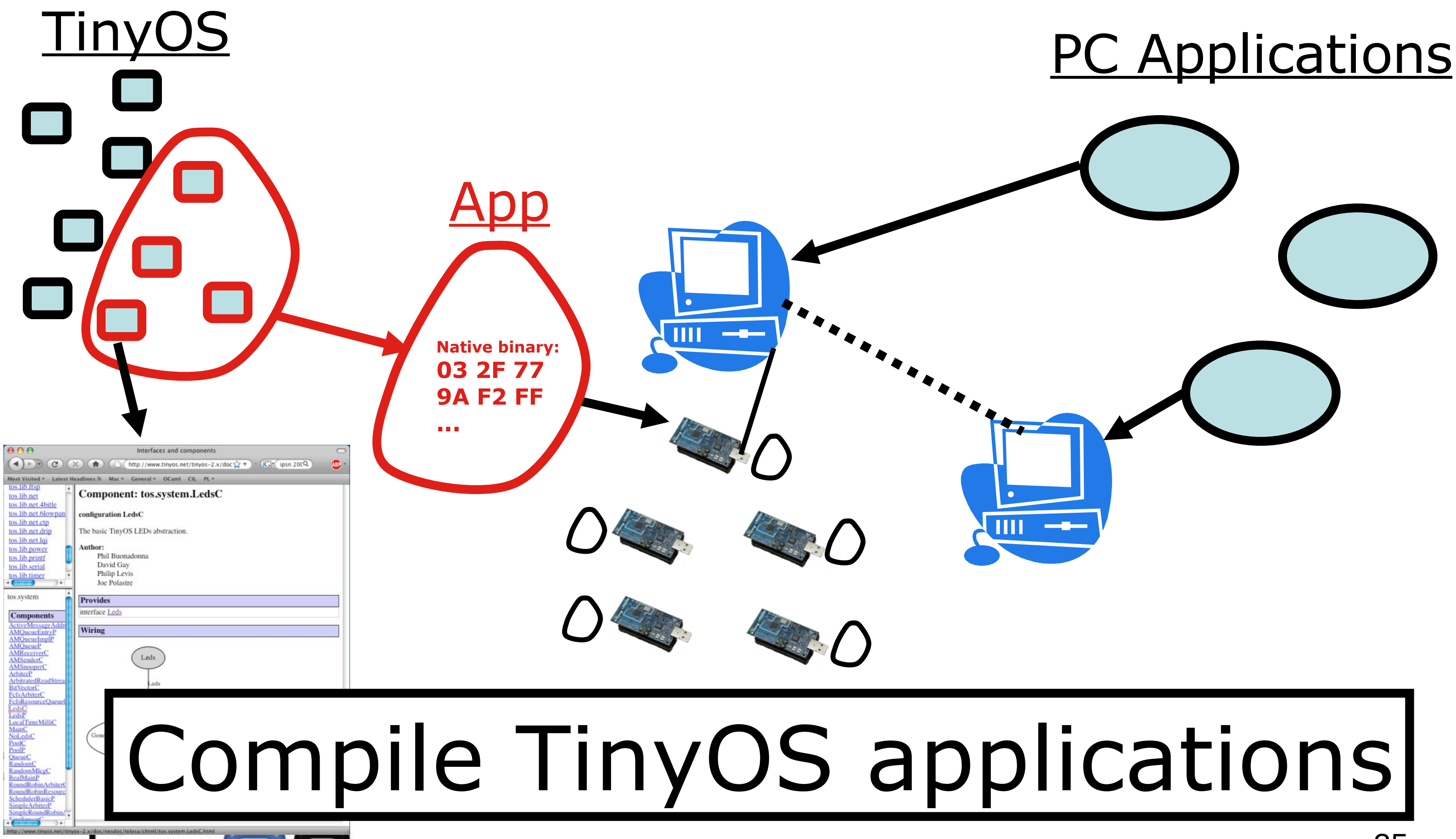
Provides

interface Leds

Wiring

http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosa/chtml/tos.system.LedsC.html

The Toolchain



Interfaces and components

http://www.tinyos.net/tinyos-2.x/doc

Component: tos.system.LedsC

configuration LedsC

The basic TinyOS LEDs abstraction.

Author:
Phil Buonadonna
David Gay
Philip Levis
Joe Polastre

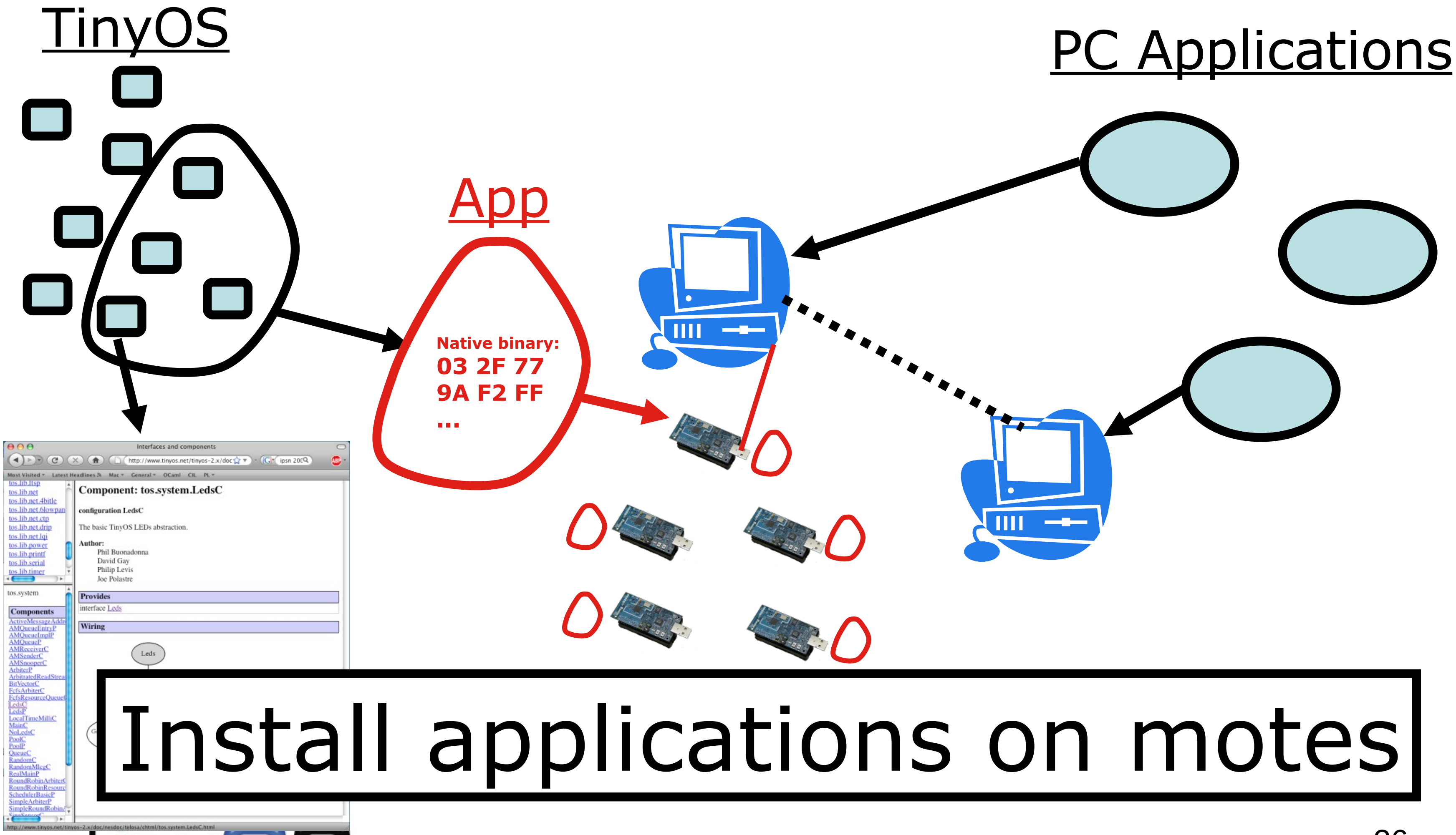
Provides

interface Leds

Wiring

Compile TinyOS applications

The Toolchain



tos.lib.ltsp
tos.lib.net
tos.lib.net.4bitle
tos.lib.net.6lowpan
tos.lib.net.ctp
tos.lib.net.drip
tos.lib.net.lqi
tos.lib.power
tos.lib.printf
tos.lib.serial
tos.lib.timer

tos.system

Components

ActiveMessageAddr
AMQueueEntryP
AMQueueImplP
AMQueueP
AMReceiverC
AMSenderC
AMSnooperC
ArbiterP
ArbitratedReadStrea
BitVectorC
FcfsArbiterC
FcfsResourceQueue
LedsC
LedsP
LocalTimeMilliC
MainC
NoLedsC
PoolC
PoolP
QueueC
RandomC
RandomMlcgC
RealMainP
RoundRobinArbiter
RoundRobinResourc
SchedulerBasicP
SimpleArbiterP
SimpleRoundRobin

Component: tos.system.LedsC

configuration LedsC

The basic TinyOS LEDs abstraction.

Author:
Phil Buonadonna
David Gay
Philip Levis
Joe Polastre

Provides

interface Leds

Wiring

http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosa/chtml/tos.system.LedsC.html

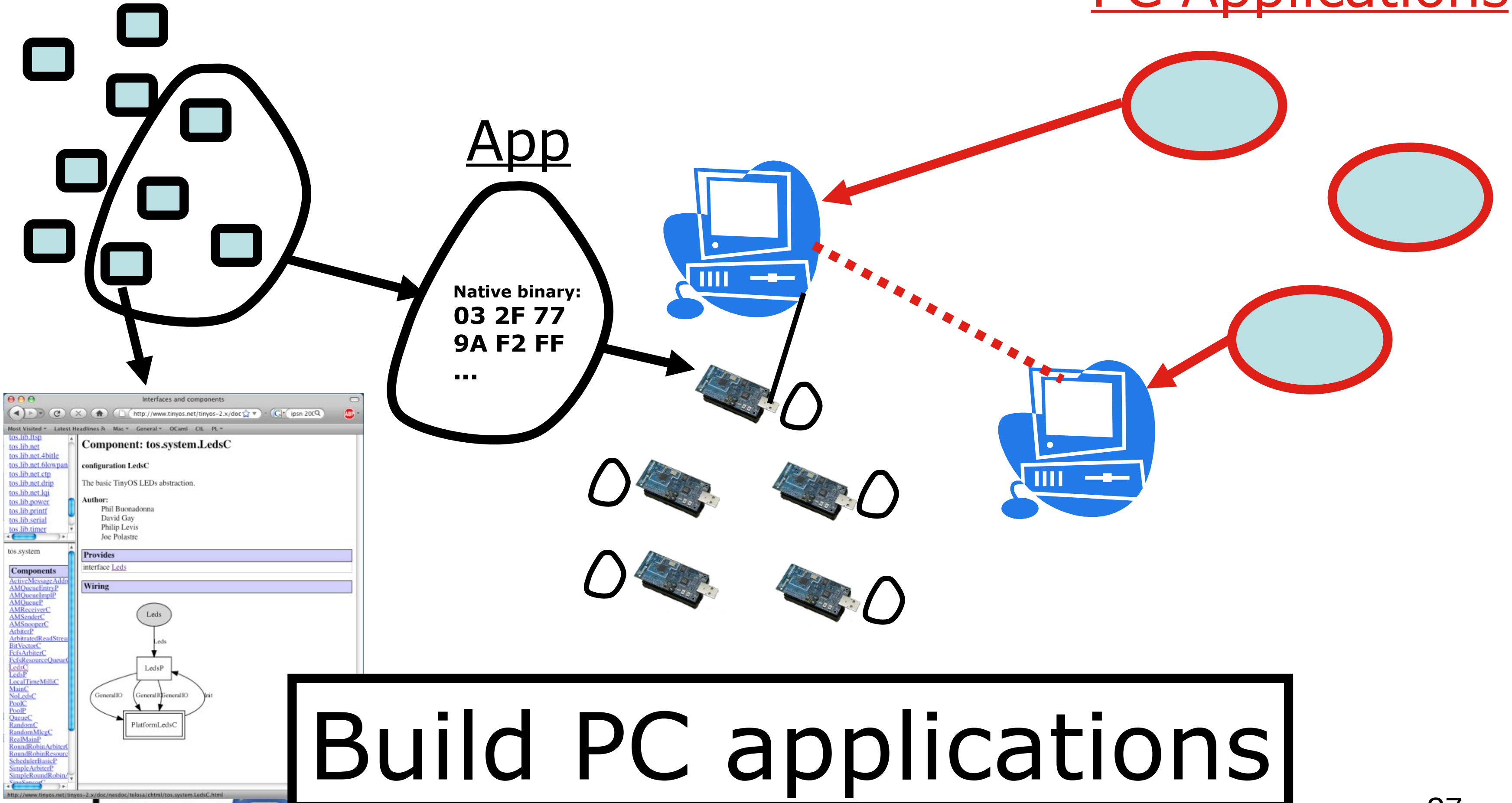
Install applications on motes

The Toolchain

TinyOS

PC Applications

App



Interfaces and components

http://www.tinyos.net/tinyos-2.x/doc

tos.lib.ftsp
tos.lib.net
tos.lib.net.4bitle
tos.lib.net.6lowpan
tos.lib.net.ctp
tos.lib.net.drip
tos.lib.net.lqi
tos.lib.power
tos.lib.printf
tos.lib.serial
tos.lib.timer

tos.system

Components

ActiveMessageAddr
AMQueueEntryP
AMQueueImplP
AMQueueP
AMReceiverC
AMSenderC
AMSnooperC
ArbiterP
ArbitratedReadStrea
BitVectorC
FcfsArbiterC
FcfsResourceQueue
LedsC
LedsP
LocalTimeMilliC
MainC
NoLedsC
PoolC
PoolP
QueueC
RandomC
RandomMlcgC
RealMainP
RoundRobinArbiter
RoundRobinResourc
SchedulerBasicP
SimpleArbiterP
SimpleRoundRobin
SineSensorC

Component: tos.system.LedsC

configuration LedsC

The basic TinyOS LEDs abstraction.

Author:
Phil Buonadonna
David Gay
Philip Levis
Joe Polastre

Provides

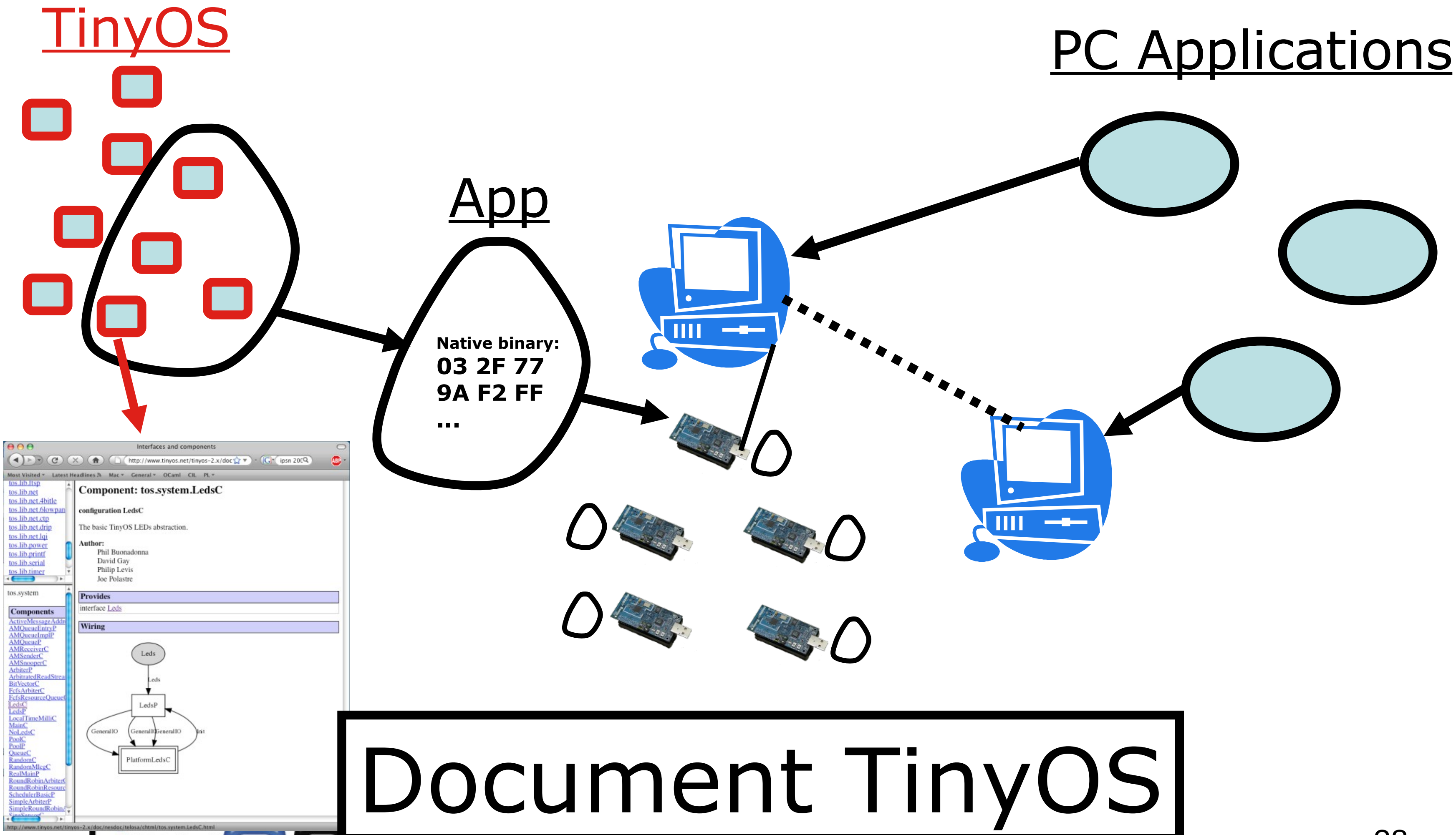
interface Leds

Wiring

http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosa/chtml/tos.system.LedsC.html

Build PC applications

The Toolchain



TinyOS

App

Native binary:
03 2F 77
9A F2 FF
…

PC Applications

Interfaces and components

http://www.tinyos.net/tinyos-2.x/doc

Most Visited · Latest Headlines · Mac · General · OCaml · CIL · PL

tos.lib.ftsp
tos.lib.net
tos.lib.net.4bitle
tos.lib.net.6lowpan
tos.lib.net.ctp
tos.lib.net.drip
tos.lib.net.lqi
tos.lib.power
tos.lib.printf
tos.lib.serial
tos.lib.timer

tos.system

Components

ActiveMessageAddr
AMQueueEntryP
AMQueueImplP
AMQueueP
AMReceiverC
AMSenderC
AMSnooperC
ArbiterP
ArbitratedReadStrea
BitVectorC
FcfsArbiterC
FcfsResourceQueue
LedsC
LedsP
LocalTimeMilliC
MainC
NoLedsC
PoolC
PoolP
QueueC
RandomC
RandomMlcgC
RealMainP
RoundRobinArbiter
RoundRobinResourc
SchedulerBasicP
SimpleArbiterP
SimpleRoundRobin

Component: tos.system.LedsC

configuration LedsC

The basic TinyOS LEDs abstraction.

Author:
Phil Buonadonna
David Gay
Philip Levis
Joe Polastre

Provides

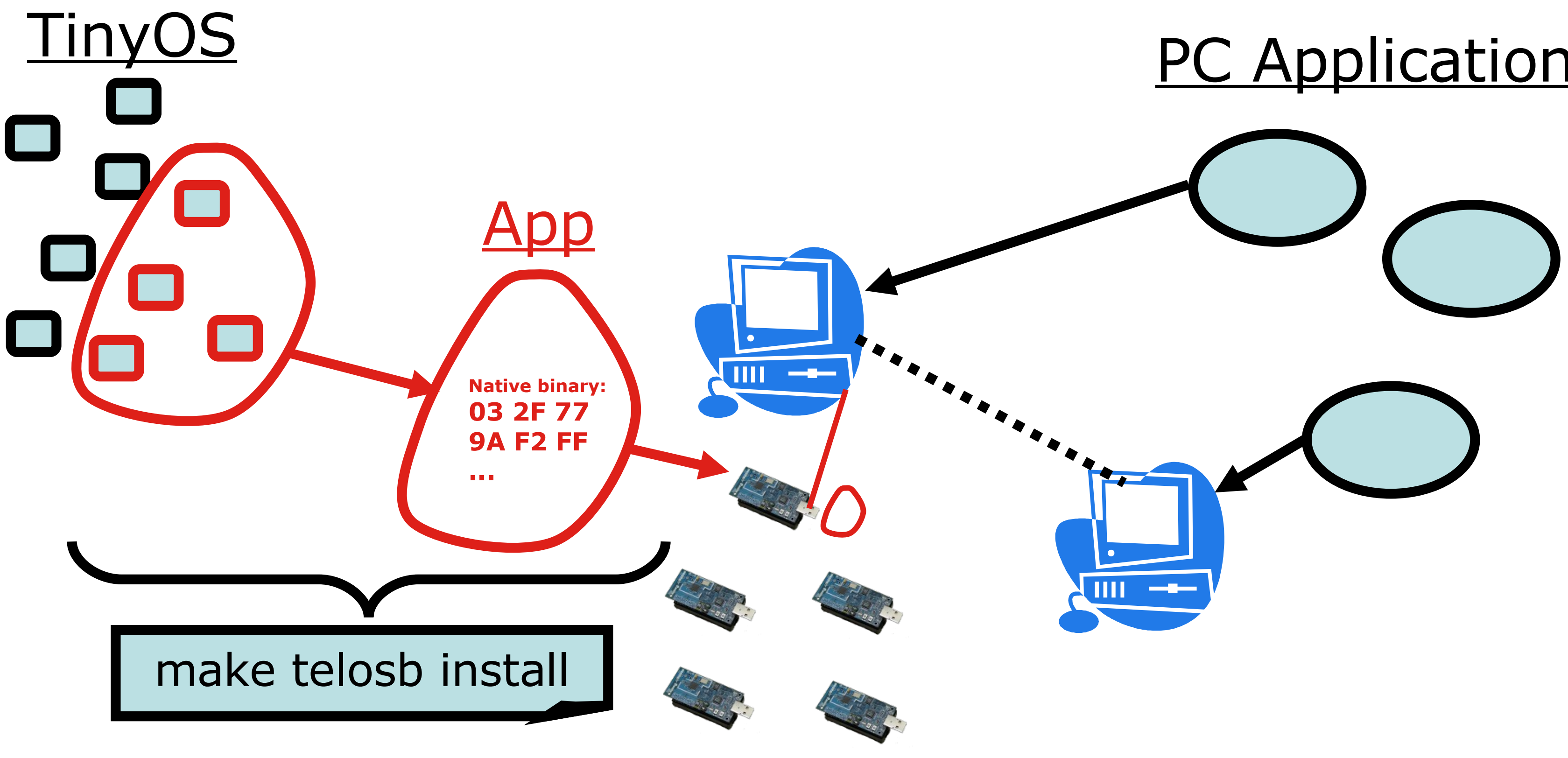
interface Leds

Wiring

http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosa/chtml/tos.system.LedsC.html

Document TinyOS

The “Make” System



TinyOS

PC Applications

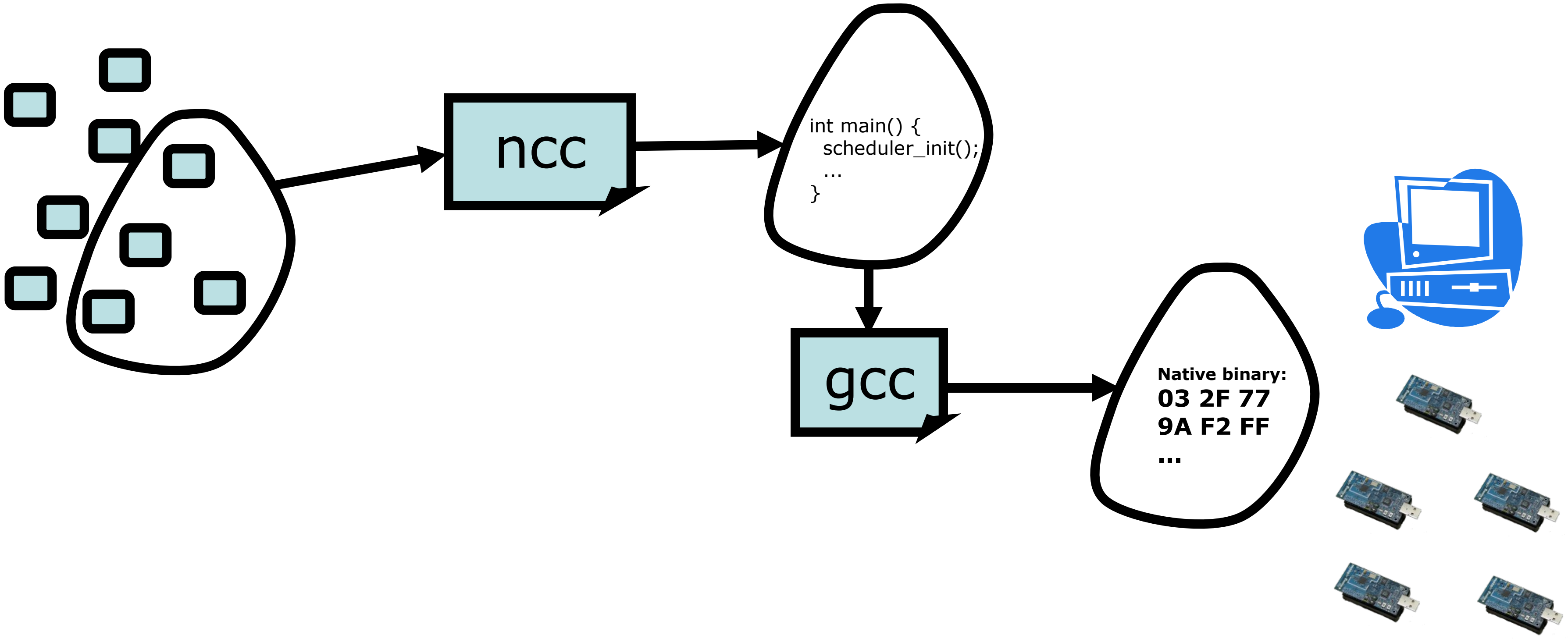
App

Native binary:
03 2F 77
9A F2 FF
...

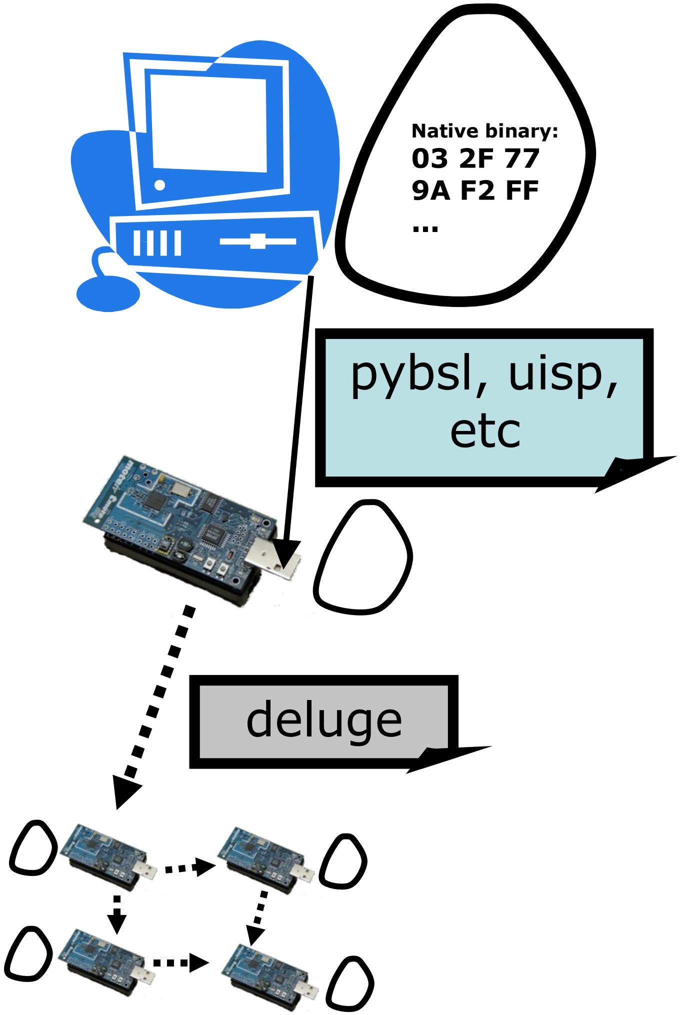
make telosb install

automates nesC, C compilation,
mote installation

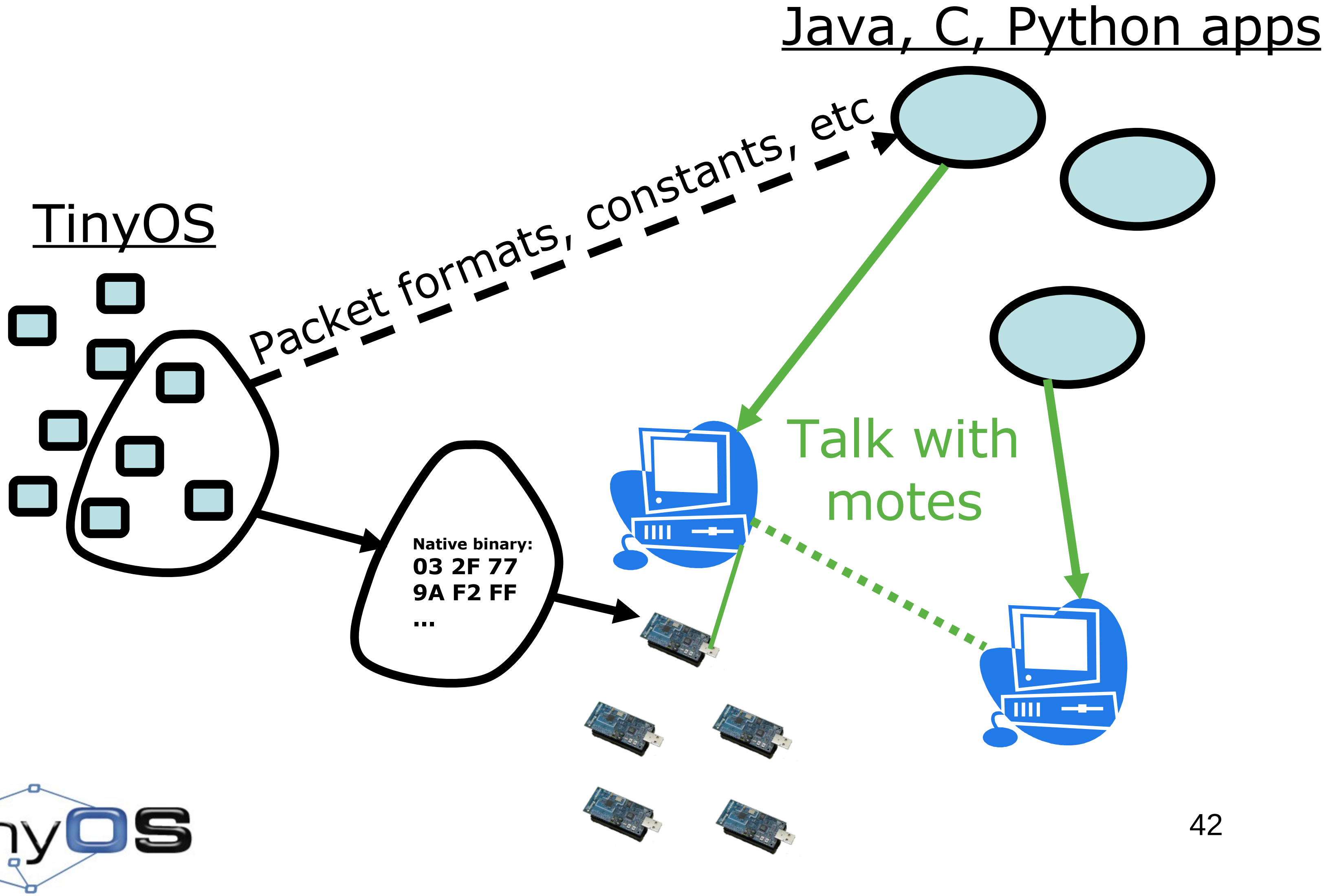
“Make”: Compile Applications



“Make”: Install Applications



Build PC Applications



Java, C, Python apps

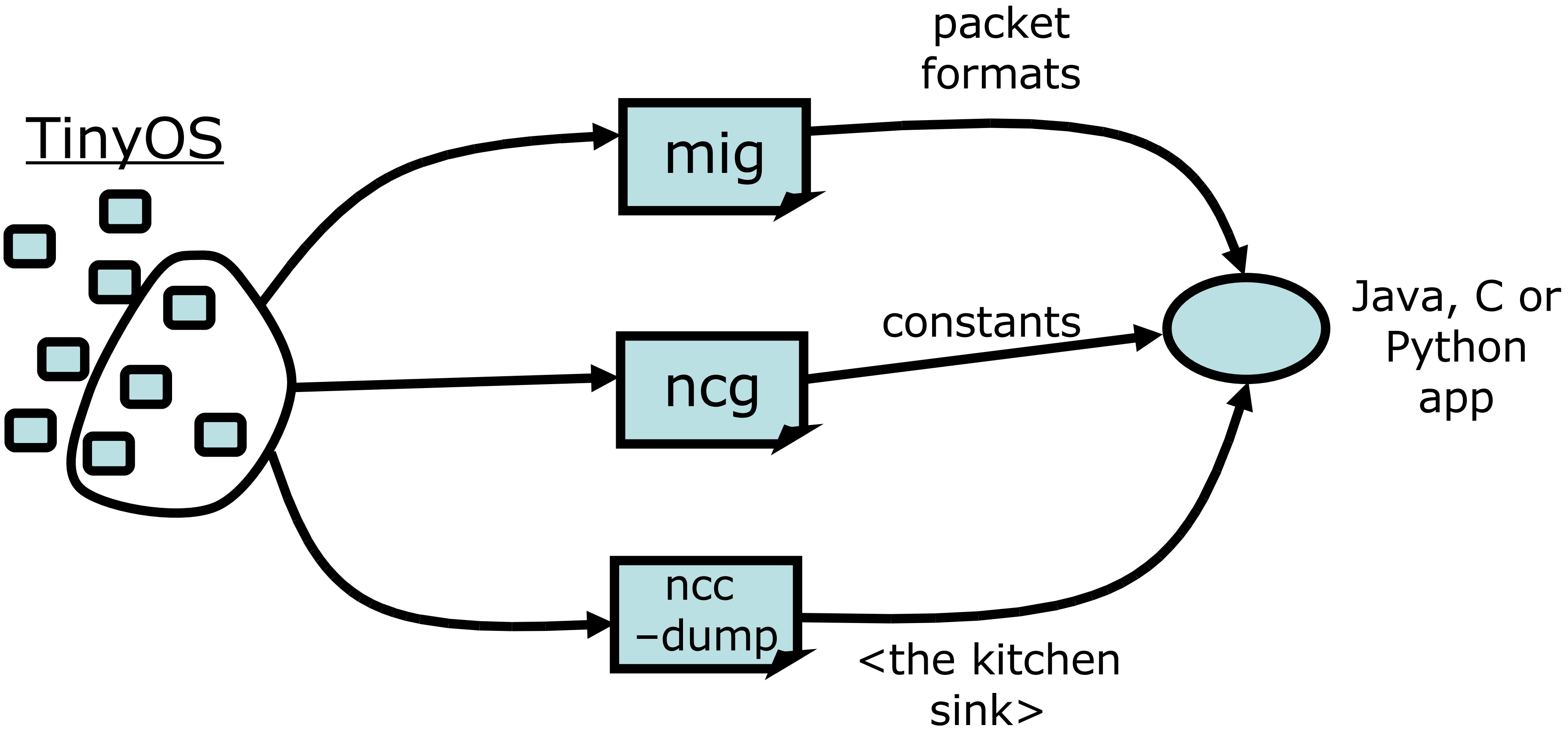
TinyOS

Packet formats, constants, etc

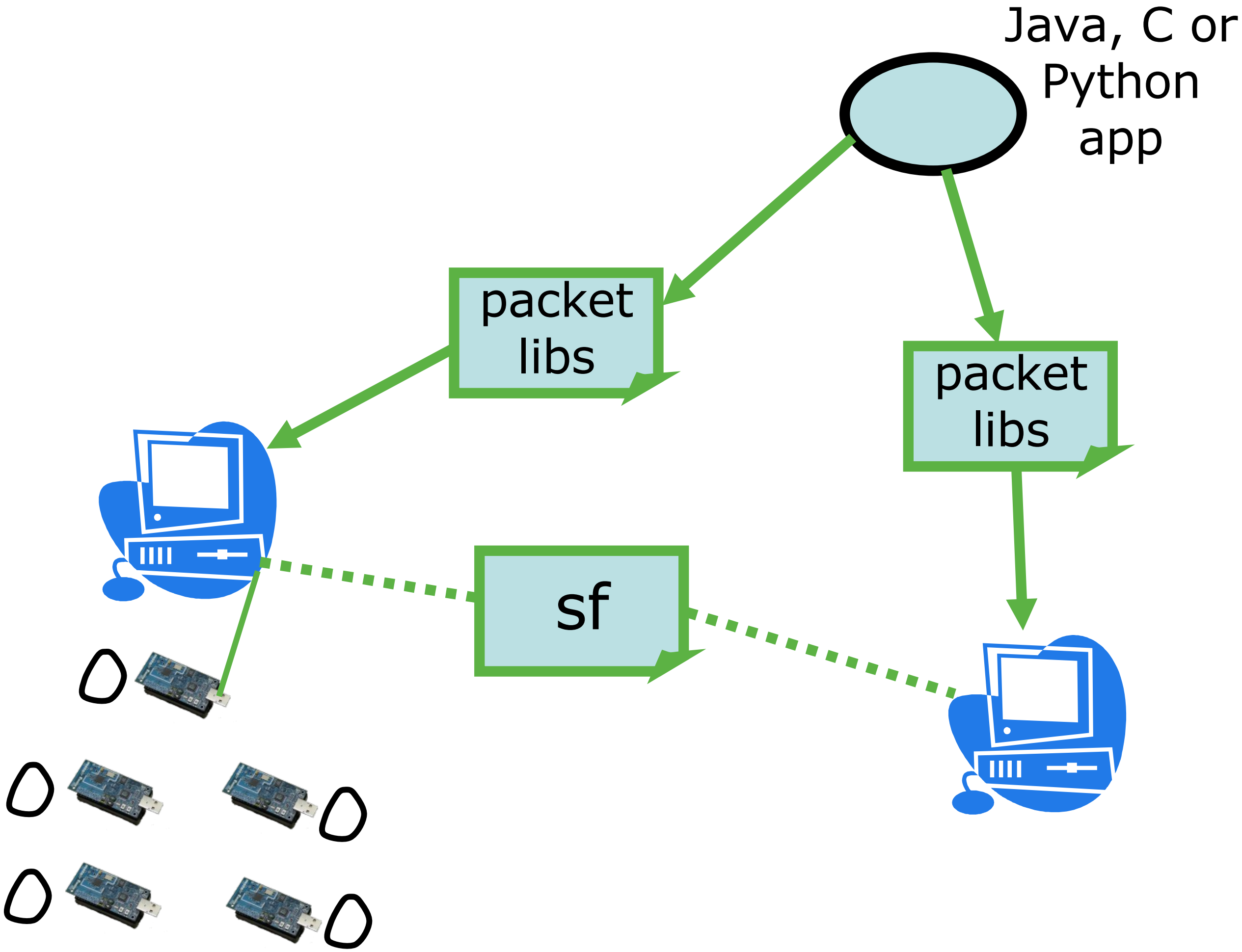
Native binary:
03 2F 77
9A F2 FF
...

Talk with motes

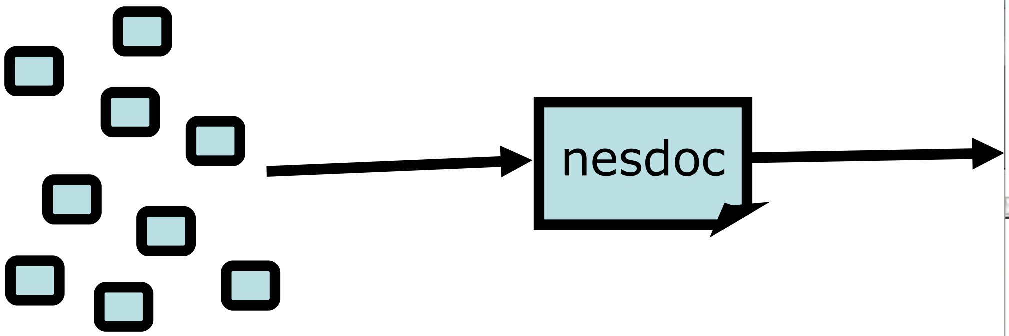
PC Applications: Extracting Information from TinyOS

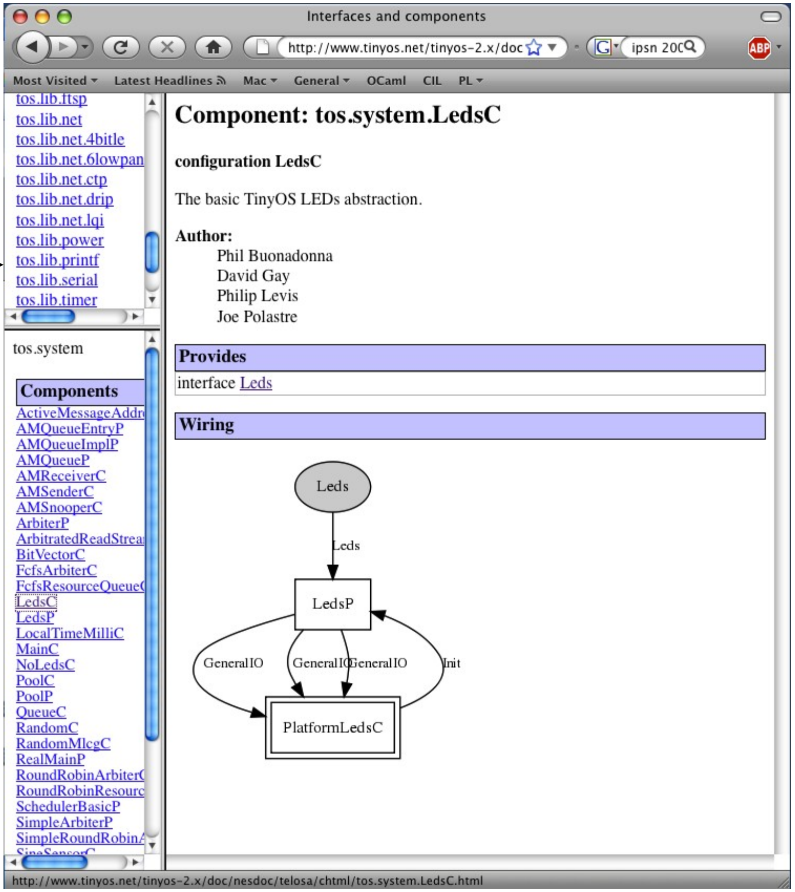


PC Applications: Talking to Motes



Document TinyOS




Interfaces and components
http://www.tinyos.net/tinyos-2.x/doc
ipsn 200
Most Visited
Latest Headlines
Mac
General
OCaml
CIL
PL

tos.lib.ftsp
tos.lib.net
tos.lib.net.4bitle
tos.lib.net.6lowpan
tos.lib.net.ctp
tos.lib.net.drip
tos.lib.net.lqi
tos.lib.power
tos.lib.printf
tos.lib.serial
tos.lib.timer

tos.system

Components

ActiveMessageAddr
AMQueueEntryP
AMQueueImplP
AMQueueP
AMReceiverC
AMSenderC
AMSnooperC
ArbiterP
ArbitratedReadStrea
BitVectorC
FcfsArbiterC
FcfsResourceQueue
LedsC
LedsP
LocalTimeMilliC
MainC
NoLedsC
PoolC
PoolP
QueueC
RandomC
RandomMlcgC
RealMainP
RoundRobinArbiter
RoundRobinResourc
SchedulerBasicP
SimpleArbiterP
SimpleRoundRobin

Component: tos.system.LedsC

configuration LedsC

The basic TinyOS LEDs abstraction.

Author:
Phil Buonadonna
David Gay
Philip Levis
Joe Polastre

Provides

interface Leds

Wiring

http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosa/chtml/tos.system.LedsC.html

TOSSIM

Răzvan Musăloiu-E. (JHU)

What is TOSSIM?

Discrete event simulator

ns2

Alternatives

Cycle-accurate simulators

Avrora, MSPSim

Two directions

Port

make PC a supported platform

TOSSIM
in tinyos-1.x

Virtualize

simulate one of the supported platforms

TOSSIM
in tinyos-2.x

Features

- Simulates a MicaZ mote
  - ATmega128L (128KB ROM, 4KB RAM)
  - CC2420
- Uses CPM to model the radio noise
- Supports two programming interfaces:
  - Python
  - C++

Anatomy

TOSSIM

```
os/lib/tossim
os/chips/atm128/sim
os/chips/atm128/pins/sim
os/chips/atm128/timer/sim
os/chips/atm128/spi/sim
os/platforms/mica/sim
os/platforms/micaz/sim
os/platforms/micaz/chips/cc2420/sim
```

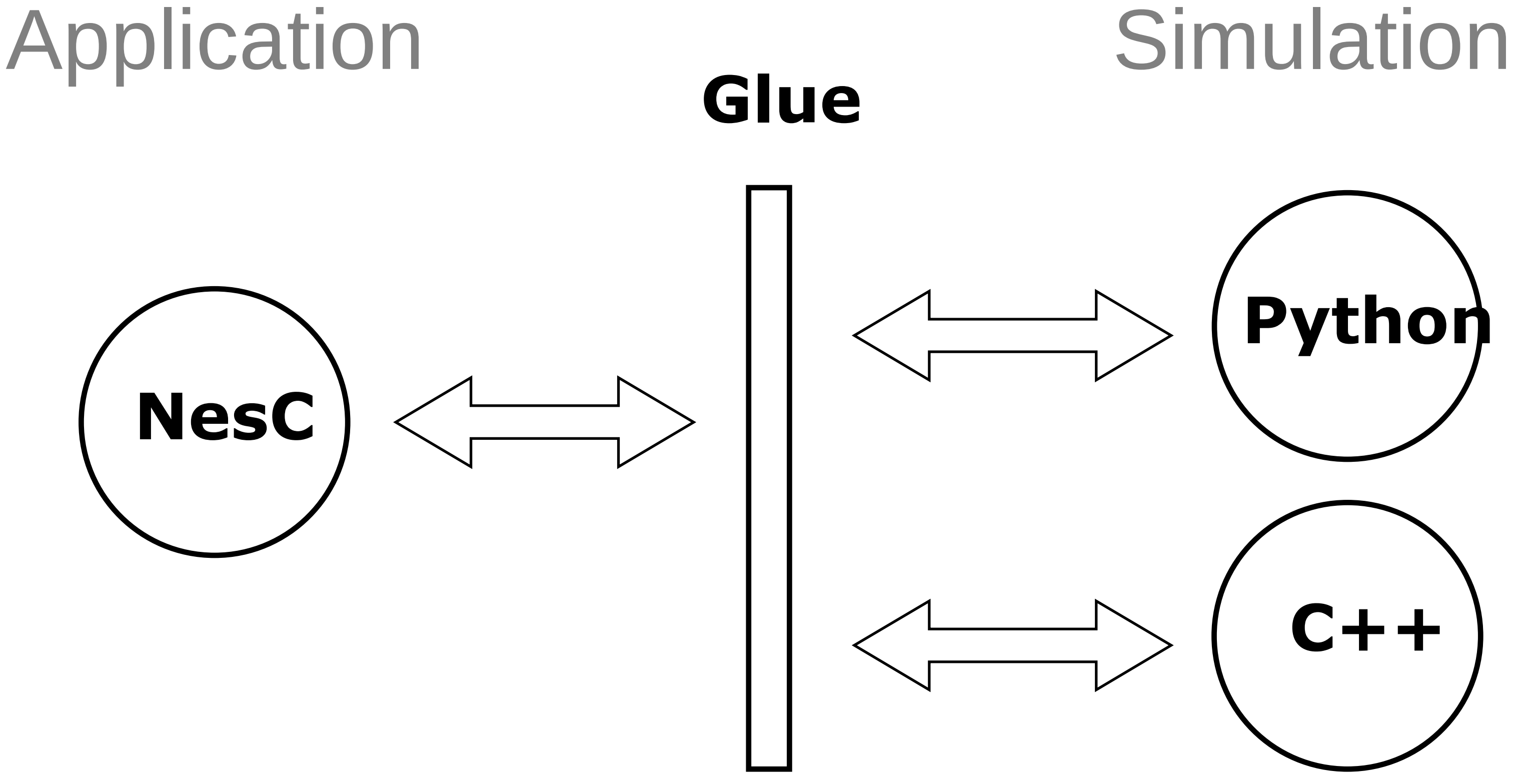
Application

```
Makefile
*.nc
*.h
```

Simulation Driver

```
*.py | *.cc
```

Quick Overview



The Building Process

```
$ make micaz sim
```

2. Generate an XML schema
3. Compile the application
4. Compile the Python support
5. Build a share object
6. Copying the Python support

```
$ ./sim.py
```

app.xml

sim.o

pytossim.o
tossim.o
c-support.o

_TOSSIMmodule.o

TOSSIM.py

TOSSIM.py

Tossim

Radio

Mote

Packet

Mac

TOSSIM.Tossim

.getNode() → TOSSIM.Mote

.radio() → TOSSIM.Radio

.newPacket() → TOSSIM.Packet

.mac() → TOSSIM.Mac

.runNextEvent()

.ticksPerSecond()

.time()

10 seconds

```
from TOSSIM import *

t = Tossim([])

  ...

while t.time() < 10*t.ticksPerSecond():
    t.runNextEvent()
```

dbg

Syntax

dbg(*tag, format, arg1, arg2, ...*);

Example

```
dbg(“Trickle”, “Starting time with time %u.\n”, timerVal);
```

Python

```
t = Tossim([])
t.addChannel(“Trickle”, sys.stdout)
```

Useful Functions

*char**	sim_time_string()
sim_time_t	sim_time()
int	sim_random()
sim_time_t	sim_ticks_per_sec()

typedef *long long int sim_time_t*;

Radio Model

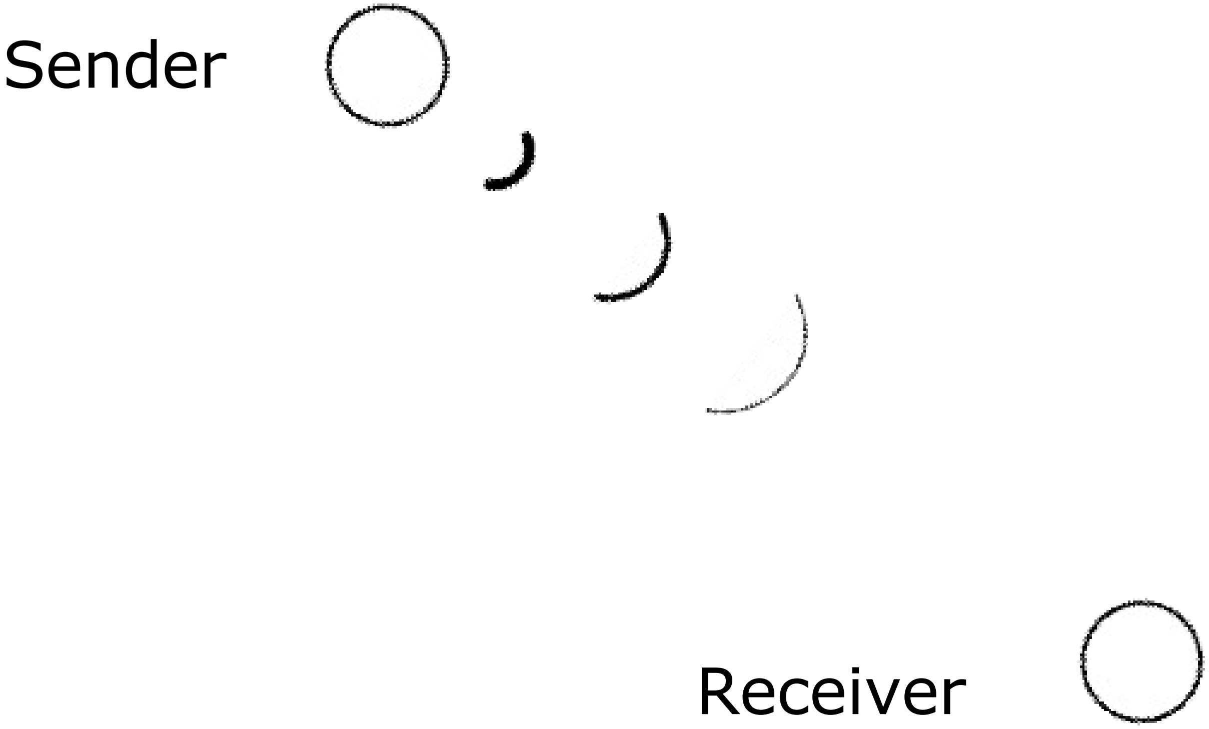
Closest-fit Pattern Matching (CPM)

Improving Wireless Simulation Through Noise Modeling

HyungJune Lee, Alberto Cerpa, and Philip Levis

IPSN 2007

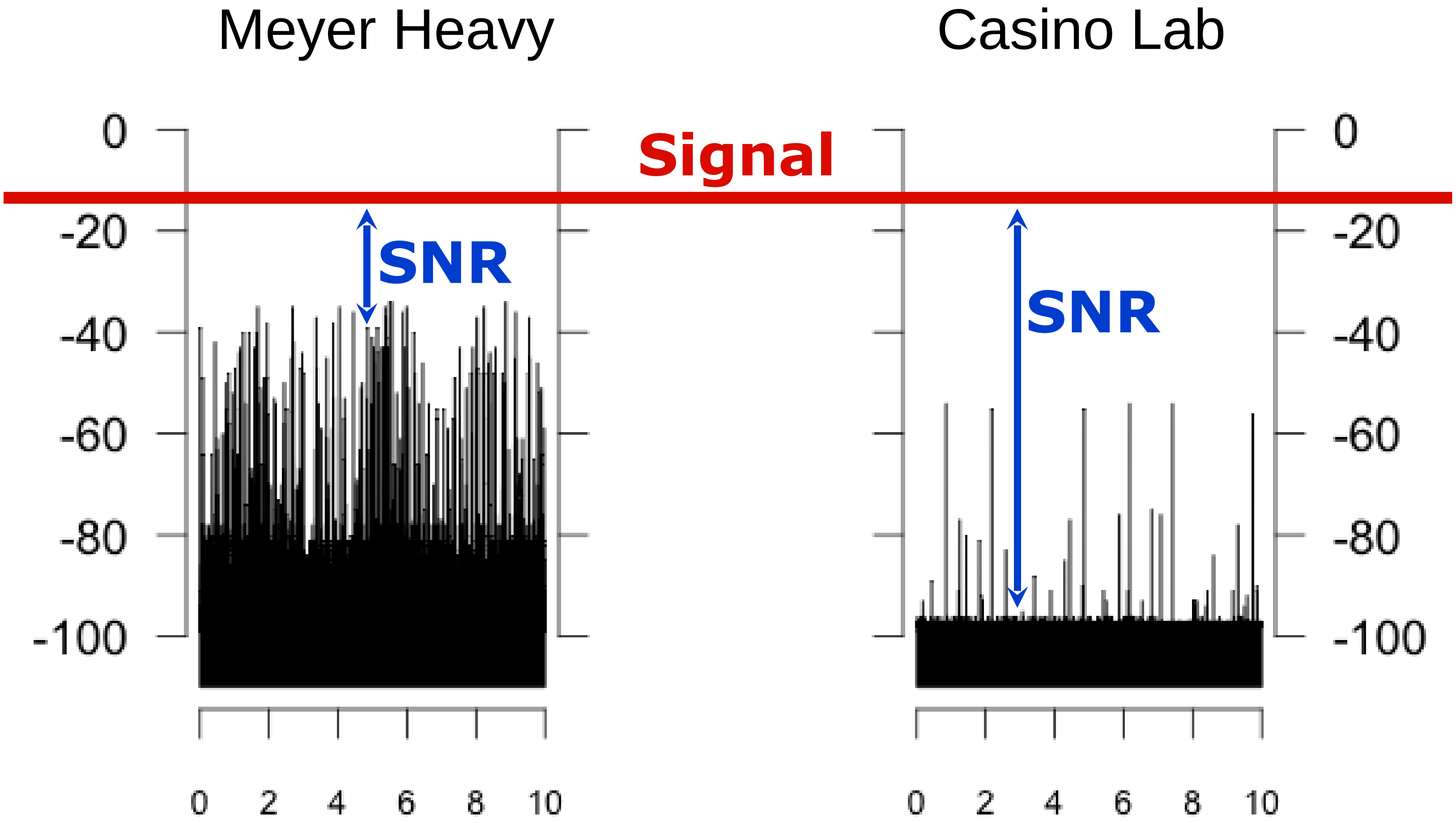
Radio Model



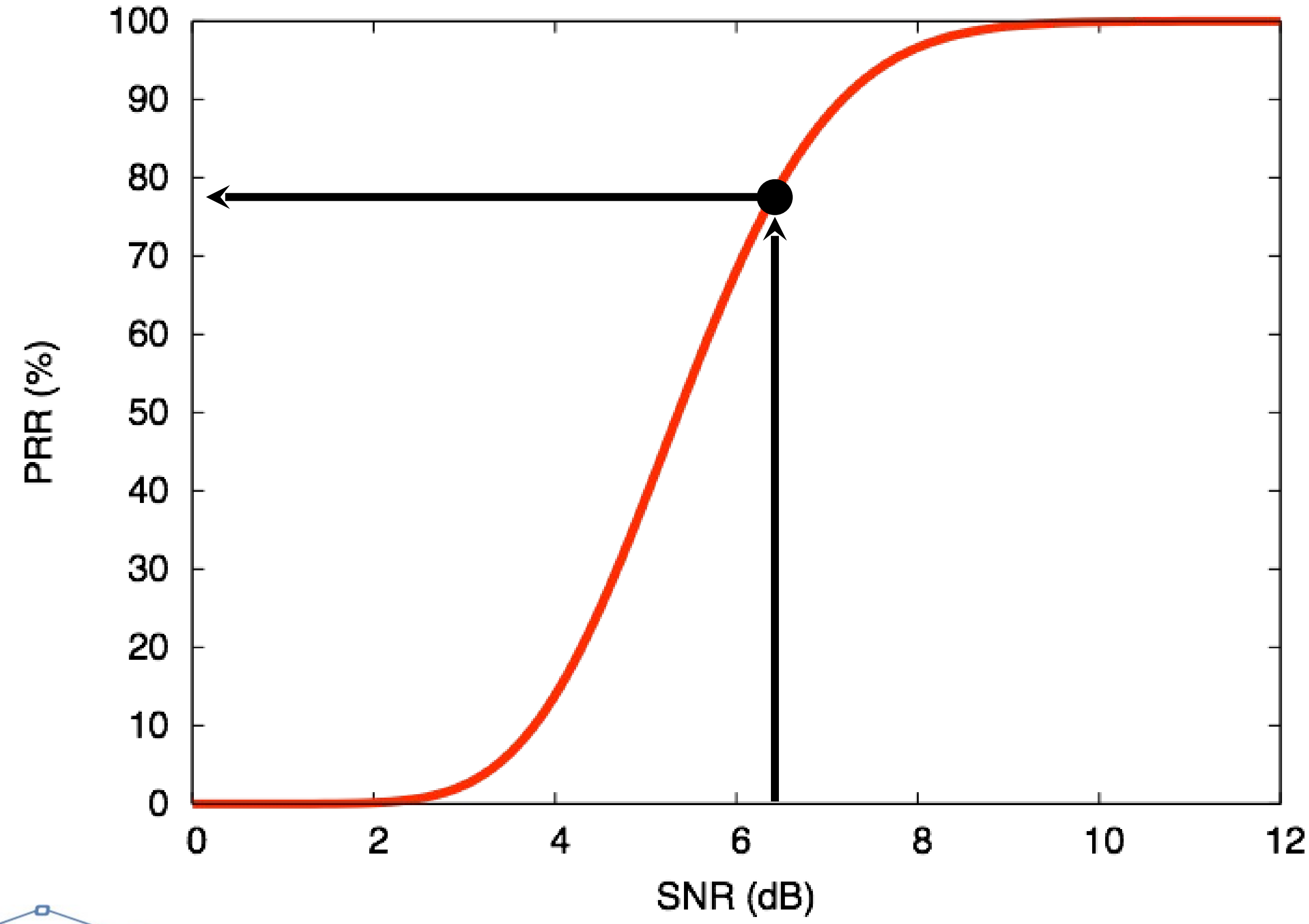
Sender

Receiver

Noise Level


Meyer Heavy
Casino Lab
Signal
SNR
SNR
0
-20
-40
-60
-80
-100
0
2
4
6
8
10

CC2420 SNR/PRR



TOSSIM.Radio

.add(*source, destination, gain*)

.connected(*source, destination*) → True/False

.gain(*source, destination*)

TOSSIM.Mote

.bootAtTime(*time*)

.addNoiseTraceReading(*noise*)

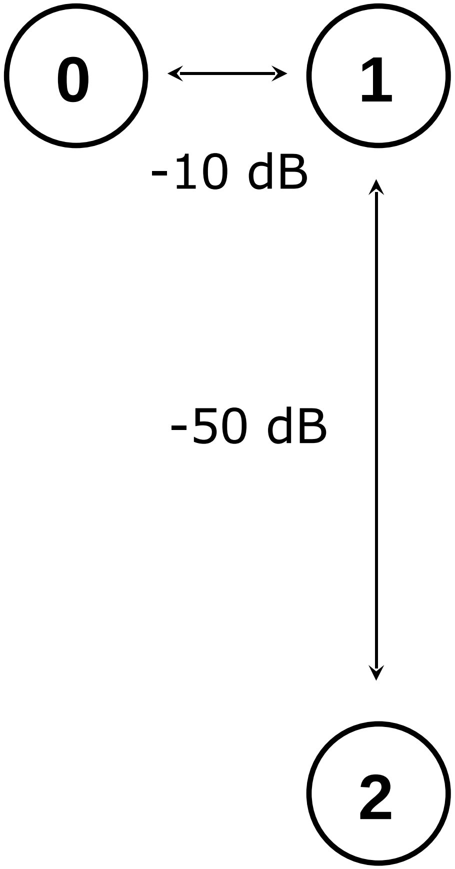
.createNoiseModel()

.isOn() → True/False

.turnOn()/.turnOff()

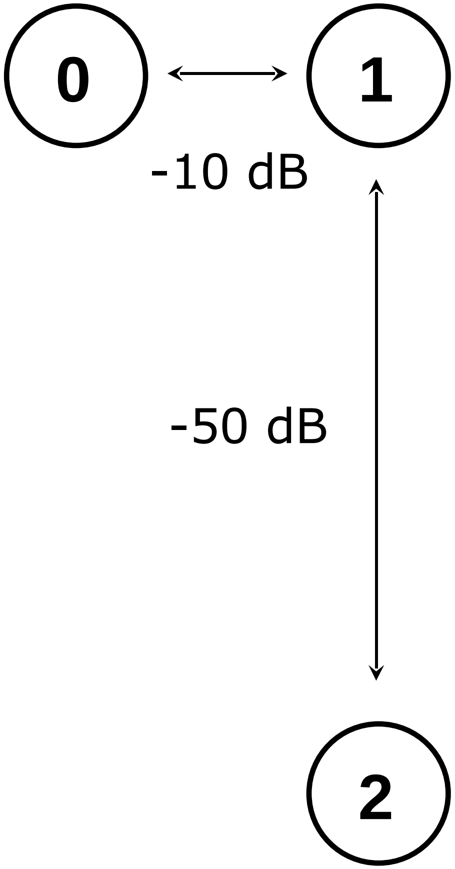
Example

```
from TOSSIM import *
t = Tossim([])
r = t.Radio()

mote0 = t.getNode(0)
mote1 = t.getNode(1)
mote2 = t.getNode(2)

r.add(0, 1, -10)
r.add(1, 0, -10)
r.add(1, 2, -50)
r.add(2, 1, -50)
```



Example (cont)

```
noise = file("meyer-short.txt")
lines = noise.readlines()
for line in lines:
  str = line.strip()
  if (str != ""):
    val = int(str)
    for m in [mote0, mote1, mote2]:
      m.addNoiseTraceReading(val)

for m in [mote0, mote1, mote2]:
    m.createNoiseModel()
```



Other Features

- Injecting packets
- Inspecting internal variables
- C++ interface
- Debuging using gdb

Improvements

- **TossimLive**
  - SerialActiveMessageC
- **CC2420sim**
  - Multiple channels
  - PacketLink
  - CC2420Packet: .getRSSI(), .getLQI()
  - ReadRssi()
  - Flash support

Future

Parametrized the PRR/SNR curve based on packet size *(in progress)*

Support for multiple binary images *(harder)*

Safe TinyOS

John Regehr (Utah)

What is Safe TinyOS?

- Memory safe execution for TinyOS 2.1 apps
  - Compiler inserts safety checks
  - These checks trap pointer / array errors before they can corrupt memory
- Behavior of memory-safe applications is unchanged
- Why use Safe TinyOS?
  - Debugging pointer and array problems on motes can be extremely difficult

Using Safe TinyOS

- Must explicitly request safe compilation

```
$ cd tinyos-2.x/apps/BaseStation
$ make micaz safe
   …
   18544 bytes in ROM
1724 bytes in RAM
$ make micaz
   …
   14888 bytes in ROM
   1724 bytes in RAM
```

Designed to Fail

- In TinyOS 2.1:

```
$ cd $TOSROOT/apps/tutorials/BlinkFail
$ make micaz install
```

- The application dies after a few seconds
  - BlinkFailC.nc has an obvious memory bug
- Next try this:

```
$ make micaz safe install
```

- After a few seconds the mote starts blinking its LEDs in funny patterns

FLIDs

- Default behavior on safety violation is to output a FLID (Fault Location IDentifier) using the LEDs
- A FLID is 8 digits in base-4
  - No LEDs lit = 0
  - 1 LED lit = 1
  - 2 LEDs lit = 2
  - 3 LEDs lit = 3
- A tool decodes FLIDs into error messages

Decoding a FLID

```
$ tos-decode-flid ./build/micaz/flids.txt 00001020
Deputy error message for flid 0x0048:


BlinkFailC__a <= BlinkFailC__a + BlinkFailC__i++ + 1
   (with no overflow): BlinkFailC.nc:70:
Assertion failed in CPtrArithAccess: BlinkFailC__a +
   BlinkFailC__i++ + 1 <= BlinkFailC__a + 10 (with no
   overflow)
```

Safe Components

- Safety is “opt in” at the level of nesC components
- This component is compiled as safe code:

```
generic module SimpleArbiterP() @safe() { … }
```

- These components are “trusted” code:

```
generic module SimpleArbiterP() @unsafe() { … }

generic module SimpleArbiterP() { … }
```

- Trusted code is compiled w/o safety checks

Porting Code to Safe TinyOS

- Recommended strategy
  - Annotate a component as **`@safe()`**
  - Compile application in safe mode
  - Fix warnings / errors
  - Repeat until no trusted components remain
- Arrays and pointers require annotations
  - Annotations are for Deputy, the safe C compiler behind Safe TinyOS
  - Purpose of annotations is to link memory regions with their bounds information

Annotation 1

- To declare **`msg`**, which always refers to a valid **`message_t`**

```
message_t* ONE msg = ...;
```

- Or if **`msg`** may be null

```
message_t* ONE_NOK msg;
```

- Most annotations have a **`_NOK`** form
  - But avoid using it when possible

Annotation 2

- To declare **`uartQueue`** as an array of 10 pointers to **`message_t`**
  - Where each element of the array must at all times refer to a valid **`message_t`**

```
message_t* ONE uartQueue[10];
```

Annotation 3

- To declare **`reqBuf`** as a pointer that always points to a valid block of at least **`reqBytes`** **`uint8_t`**s:

```
uint8_t *COUNT(reqBytes) reqBuf;
```

- Array dereferencing / pointer arithmetic can be done on **`reqBuf`**:
  - **`reqBuf[0]`** is legal
  - **`reqBuf[reqBytes-1]`** is legal
  - **`reqBuf[reqBytes]`** results in a safety violation

Annotation 4

- Multiple-indirect pointers require an annotation at each level:

```
int *ONE *ONE pp = ...;
```

- However, these are uncommon in TinyOS

Annotation 5

- If you get stuck, the “trusted cast” offers an escape hatch:

```
cc2420_header_t* ONE x =
  TCAST( cc2420_header_t* ONE,
  (uint8_t *)msg +
  offsetof(message_t, data) -
  sizeof(cc2420_header_t)

);
```

Interface Annotation 1

- The **`getPayload()`** command from the Packet interface might be annotated like this:

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

Interface Annotation 2

- However, `tinyos-2.x/tos/interfaces/Packet.nc` contains:

```
* @param 'message_t* ONE msg' …

* @param len …

* @return 'void* COUNT_NOK(len)' … */

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

- nesC allows you to put annotations in documentation comments

Safe TinyOS Summary

- Safe execution is useful
- Safety annotations are good documentation
- Most Mica2, MicaZ, TelosB apps and core services are safe
- Safe TinyOS Tutorial:
  - http://docs.tinyos.net/index.php/Safe_TinyOS

Threads

Kevin Klues (UCB)

The Great Divide

- Event-Based Execution
  - More efficient
  - Less RAM usage
  - More complex
- Thread-Based Execution
  - Less Efficient
  - More RAM Usage
  - Less Complex

```
void myFunc() {
    error_t e = read();
    //continue execution flow
}
void readDone(uint8_t val, error_t e) {
    //read() continuation code
}
```

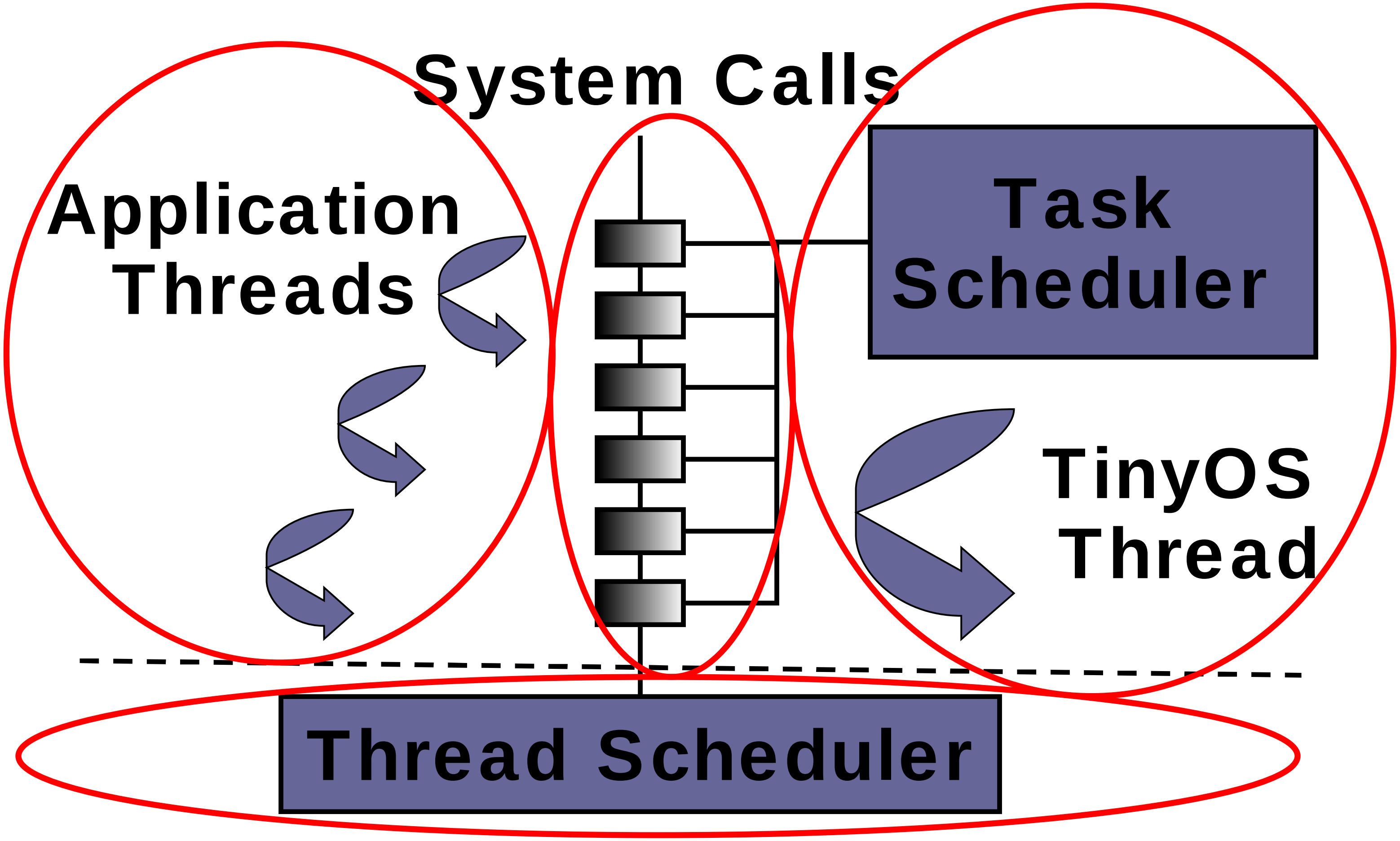
```
void myFunc() {
    error_t e;
    uint8_t val = read(&e);
    //read() continuation code
}
```

TOSThreads aims to resolve this fundamental tension

TOSThreads in a Nutshell

- Natural extension to the existing TinyOS concurrency model
- Implements Full-Fledged Threads Library
- Introduces Minimal Disruption to TinyOS
- Provides Flexible Event-based / Thread-based Code Boundary
- Enables Dynamic Linking and Loading of Application Binaries at Runtime
- Standard C and nesC based APIs

Architecture Overview



System Calls

Application Threads

Task Scheduler

TinyOS Thread

Thread Scheduler

Blink Example (nesC)

```
configuration BlinkAppC {
}
implementation {
  components MainC, BlinkC,  LedsC;
  components new ThreadC(STACK_SIZE);

  MainC.Boot <- BlinkC;
  BlinkC.Thread -> ThreadC;
  BlinkC.Leds -> LedsC;
}
```

```
module BlinkC {
  uses {
    interface Boot;
    interface Thread;
    interface Leds;
  }
}
implementation {
  event void Boot.booted() {
    call Thread.start(NULL);
  }
  event void Thread.run(void* arg) {
    for(;;) {
      call Leds.led0Toggle();
      call Thread.sleep(BLINK_PERIOD);
    }
  }
```

Blink Example (standard C)

```
#include "tosthread.h"
#include "tosthread_leds.h"

//Initialize variables associated with a thread
tosthread_t blink;
void blink_thread(void* arg);

void tosthread_main(void* arg) {
  tosthread_create(&blink, blink_thread, NULL, STACK_SIZE);
}
void blink_thread(void* arg) {
  for(;;) {
    led0Toggle();
    tosthread_sleep(BLINK_PERIOD);
  }
}
```

Modifications to TinyOS

- Change in boot sequence
- Small change is TinyOS task scheduler
- Additional post-amble in the interrupt sequence

Boot Sequence

Standard TinyOS Boot

```
event void TinyOS.booted() {
  atomic {
        platform_bootstrap();

        call Scheduler.init();

        call PlatformInit.init();
        while (call Scheduler.runNextTask());

        call SoftwareInit.init();
        while (call Scheduler.runNextTask());
  }
  signal Boot.booted();

  /* Spin in the Scheduler */
  call Scheduler.taskLoop();
}
```

Main

```
int main() {
  signal TinyOS.booted();

  //Should never get here
  return -1;
}
```

Boot Sequence

Thread Scheduler Boot

```
event void ThreadScheduler.booted() {
  setup_TinyOS_in_kernel_thread();
  signal TinyOSBoot.booted();
}
```

New Main

```
int main() {
  signal ThreadScheduler.booted();

  //Should never get here
  return -1;
}
```

Task Scheduler

Original

```
command void Scheduler.taskLoop()  {
  for (;;)  {
    uint8_t nextTask;
    atomic {
          while ((nextTask = popTask()) == NO_TASK))
            call McuSleep.sleep();
    }
    signal TaskBasic.runTask[nextTask]();
  }
}
```

New

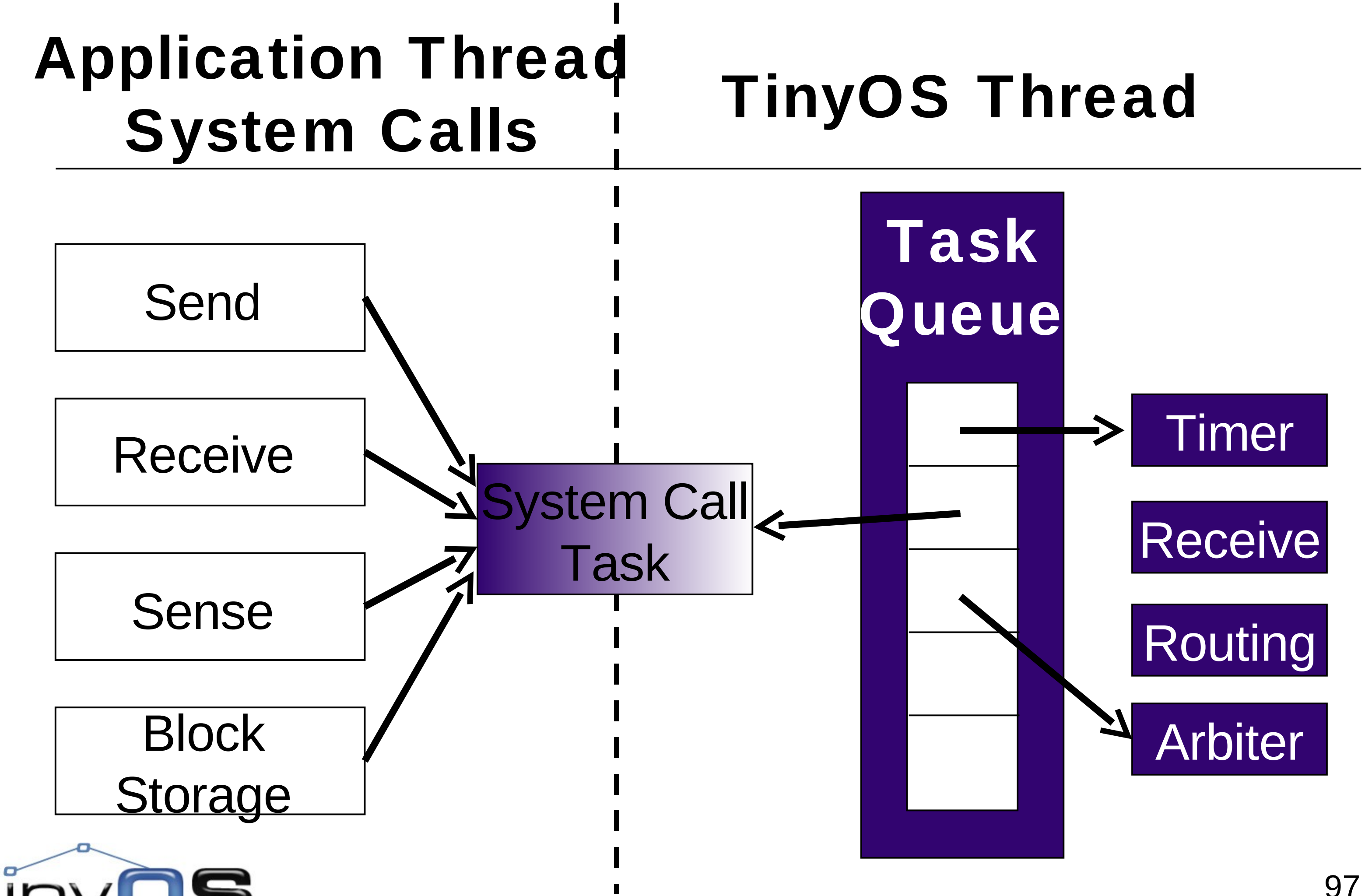
```
command void Scheduler.taskLoop()  {
  for (;;)  {
    uint8_t nextTask;
    atomic {
          while ((nextTask = popTask()) == NO_TASK)
            call ThreadScheduler.suspendThread(TOS_THREAD_ID);
    }
    signal TaskBasic.runTask[nextTask]();
  }
}
```

Interrupt Handlers

```
TOSH_SIGNAL(ADC_VECTOR) {
   signal SIGNAL_ADC_VECTOR.fired();
   atomic interruptCurrentThread();
 }
TOSH_SIGNAL(DACDMA_VECTOR) {
   signal SIGNAL_DACDMA_VECTOR.fired();
   atomic interruptCurrentThread();
 }
….
….
```

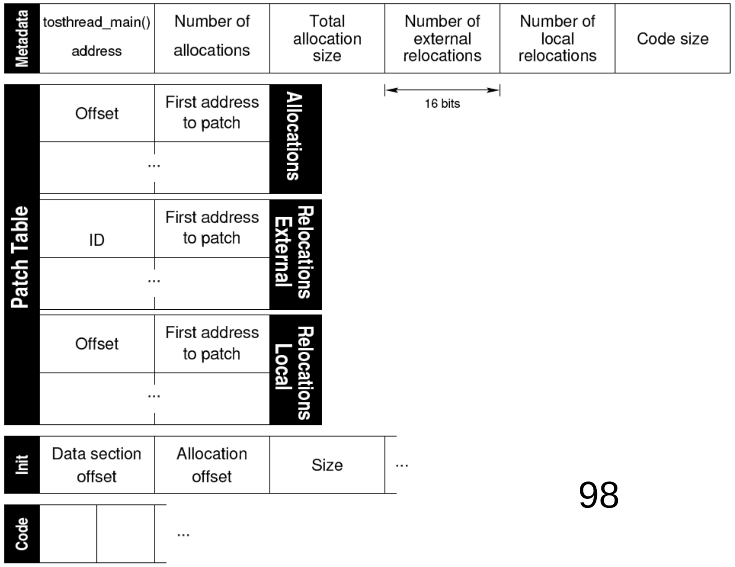
```
void interruptCurrentThread() {
  if( call TaskScheduler.hasTasks() ) {
    call ThreadScheduler.wakeupThread(TOS_THREAD_ID);
    call ThreadScheduler.interruptCurrentThread();
  }
}
```

System Calls



Linking and Loading

- Full applications written in standard C
- Custom MicroExe format
- Multiple concurrently running applications
- Generic TinyLD component supporting multiple APIs



Metadata
tosthread_main() address
Number of allocations
Total allocation size
Number of external relocations
Number of local relocations
Code size
16 bits

Patch Table
Allocations
Offset
First address to patch
...
External Relocations
ID
First address to patch
...
Local Relocations
Offset
First address to patch
...

Init
Data section offset
Allocation offset
Size
...

Code
...

Resources

- TOSThreads Tutorial

  http://docs.tinyos.net/index.php/TOSThreads_Tutorial
- TOSThreads TEP

  http://www.tinyos.net/tinyos-2.x/doc/html/tep134.html
- Source Code

System code:	tinyos-2.x/tos/lib/tosthreads
Example Applications:	tinyos-2.x/apps/tosthreads

Protocols

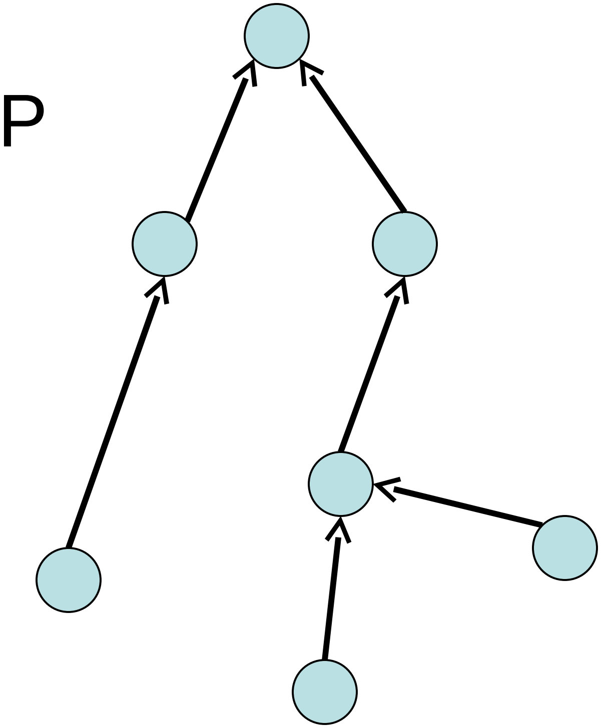
Omprakash Gnawali (USC)

Protocols in TinyOS 2.1

- Network Protocols
  - Collection: CTP, MultihopLQI
  - Dissemination: Drip, DIP
- Time Synchronization (FTSP)
- Over-the-air programming (Deluge)

Collection

- Collect data from the network to one or a small number of roots
- One of many traffic classes
- Available: MultihopLQI and CTP



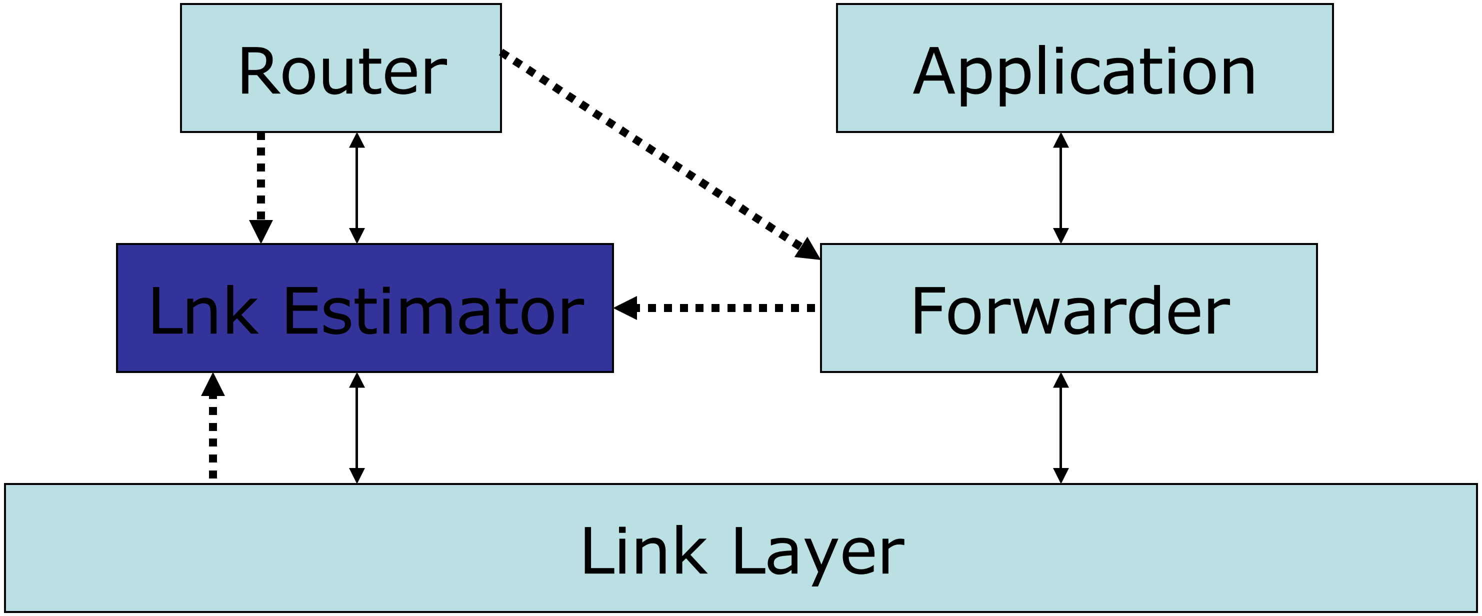
MultihopLQI

- Mostly tested and used on platforms with CC2420
  - MicaZ, TelosB, …
- Small code footprint
- tos/lib/net/lqi

CTP

- Platform independent
- More consistent performance than with MultihopLQI
- Code footprint can be a concern
- tos/lib/net/ctp

CTP Architecture



CTP Link Estimator

- Platform independent
  - Beacons and data packets
- Bi-directional ETX estimate
- Does not originate beacons itself
- Accurate but also agile

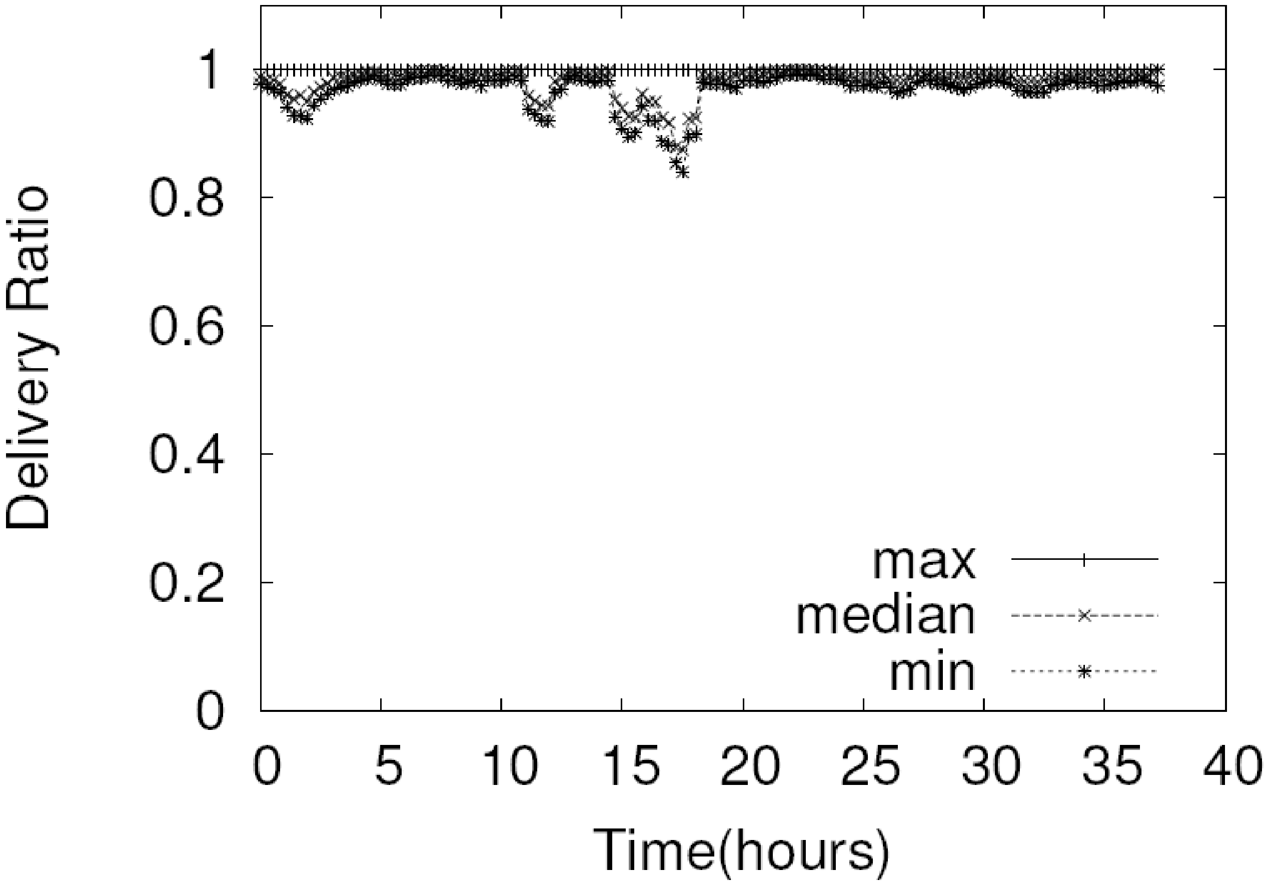
CTP Router

- ETX path metric
- Beacon interval can be 64 ms-x mins
- Select new path if better by at least 1.5 ETX
- Alternate parents

CTP Forwarder

- Duplicate suppression
- Retransmissions
- Loops trigger route updates
- Forward through alternate parents

CTP Reliability



Dissemination

- Send data to all the nodes
  - Commands, configuration parameters
- Efficient and fast
- Available protocols – Drip and DIP

Drip

- Fast and efficient for small number of items
- Trickle timers for advertisements
- Suppression
- tos/lib/net/drip

DIP

- Efficiently Disseminates large number of items (can not fit in one packet)
- Use hashes and version vectors to detect and identify updates to the values
- tos/lib/net/dip

Deluge

- Over-the-air programming
- Disseminates code
- Programs the nodes

Deluge Details

- Supports Tmote Sky/EPIC and MicaZ.
- Bulk dissemination on top of Drip
- Python tools
- Support for MIB600. **(new)**
- tos/lib/net/Deluge, tos/lib/tosboot

Time Synchronization

- Global time on all the nodes
- Node with smallest id becomes the root
- Flooding Time Synchronization Protocol (FTSP)
- tos/lib/ftsp

Upcoming Technologies

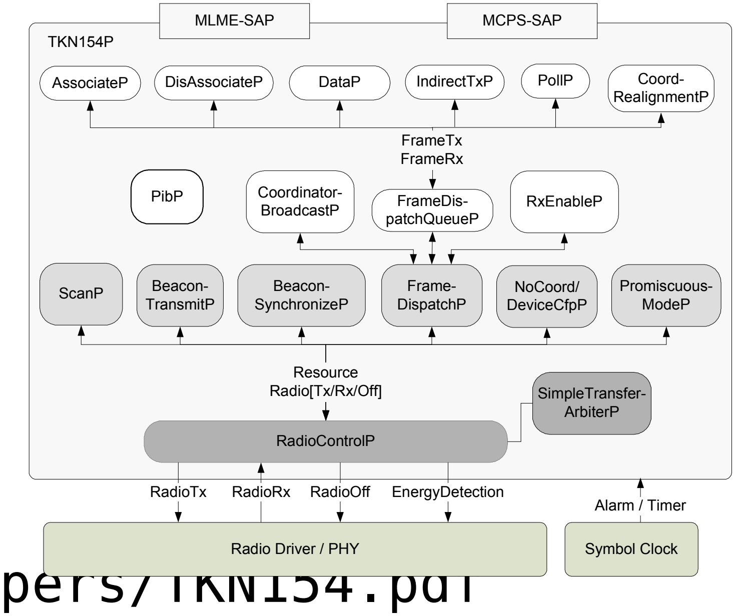
Stephen Dawson-Haggerty (UCB)

“Work in Progress”

- Proceeding in working groups
  - IEEE 802.15.4
  - Zigbee
  - 6lowpan/IPv6
- Overall theme: leverage emerging standards

IEEE 802.15.4

- PHY/MAC specification
- MAC under development by working group
  - CSMA-CA
  - GTS
  - Slotted CSMA-CA
- Application interface in flux
- More reading:
  - `tos/lib/mac/tkn154`
  - `http://www.tkn.tu-berlin.de/publications/papers/TKN154.pdf`



ZigBee

- Network protocol and application stack built on IEEE 802.15.4
- Goal: standards-complaint Zigbee-pro stack built on 802.15.4 stack
  - Cluster-tree, mesh routing
  - Security
  - Application profiles: *i.e.* HVAC, Sensing

IPv6

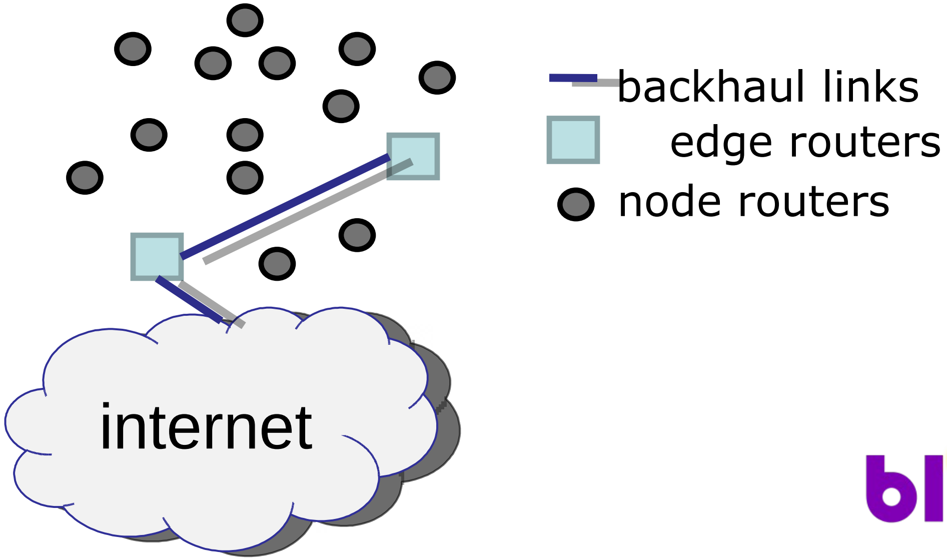
- IPv6 a good fit for sensor networks
  - What about large header size? 6loWPAN
- Ideas about many important issues
  - Management
  - Configuration
  - Security
- TEP138, draft-tavakoli-hydro-01

IPv6

- BLIP: IPv6 for TinyOS
  - Current progress: being integrated into core
- Useful basic feature set
  - Mesh routing
  - TCP/UDP
- Lots of tools, libraries for building apps
  - Shell, network reprogramming, RPC, …

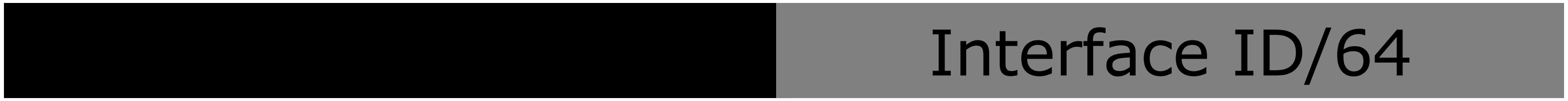
An IP Network

- “sensor network” ≈“IP subnet”
- “TOS_NODE_ID” ≈“IP address”
- “base station” ≈“edge router”
- “application gateway” no longer exists



Addressing

- 128-bit address space



- Lots of IPv6 RFCs deal with this: RFC2461, RFC4862

Address type	Example	TinyOS usage
Link-local unicast	fe80::beef	L2 Mapped
Link-local multicast	ff02::1	Radio local broadcast
Global unicast	2001::64	Routable address

Useful Interfaces

UDPSocketC

```
interface UDP {
    command error_t bind(uint16_t port);
    command error_t sendto(struct sockaddr_in6 *dest,
                                                  void *payload,
    uint16_t len);
    event void recvfrom(struct sockaddr_in6 *src, void
    *payload,
                        uint16_t len, struct ip_metadata *meta);
```

ICMPResponderC

```
interface ICMPPing {
   command error_t ping(struct in6_addr *target,
                                        uint16_t period,
uint16_t n);
   event void pingReply(struct in6_addr *source,
                                        struct icmp_stats
*stats);
   event void pingDone(uint16_t ping_rcv, uint16_t ping_n);
}
```

Address Structures

- A lot like linux: ip.h

```
struct sockaddr_in6 {
  uint16_t sin6_port;
  struct in6_addr sin6_addr;
};
```

Example App: Sense & Send

```
Configuration MyAppC{
} implementation {
        components MyAppP, new UdpSocketC();
        MyAppP.UDP -> UdpSocketC;
        ...
}
event Timer.fired() {
   call Read.read();
}
Read.readDone(error_t result, uint16_t val) {
   struct sockaddr_in6 dest;
   nx_struct report r;
   r.reading = val;
   inet_pton6(“2001::64”, &dest.sin6_addr);
   dest.sin6_port = htons(REPORT_PORT);
   call UDP.sendto(dest, &r, sizeof(r));
}
```

Conclusions

- Exciting developments expected in 2009!
- Project links:
  - 802.15.4: http://tinyos.stanford.edu:8000/15.4_WG/
  - Zigbee: http://www.open-zb.net/
  - BLIP: http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip

Hands-on Session

Răzvan, Om, et al.

Goals

- Install TinyOS
- Layout of **`tinyos-2.x`**
- Write two applications

  (A) DisseminationDemoClient
  (B) CollectionsDemoClient

Options

- LiveCD
  - XubunTOS
  - Customized Ubuntu 8.10 LiveCD

Today

- Native
  - Linux
    - .rpm packages
    - .deb packages
  - Windows: Cygwin + .rpm packages
  - MacOS X
    - stow
    - macports

Recommended

Other Options

- VMware
  - Jetos
    - based on JeOS (Ubuntu Server 8.04)
    - optimized for ssh access
    - very small: 190MB compressed
  - Lenny
    - based on Debian 5.0 “Lenny”
    - graphical interface using XFCE
    - bigger: 300MB compressed
  - XubunTOS

Components

- NesC: nesc_*.deb
- Cross compiler
  - binutils: msp430-binutils-tinyos_*.deb
  - gcc: msp430-gcc-tinyos_*.deb
  - libc: msp430-libc-tinyos_*.deb
  - gdb (optional)
- Deputy: deputy-tinyos_*.deb

Environment

```
export TOSROOT=$HOME/local/src/tinyos-2.xexport
TOSDIR=$TOSROOT/tos
export MAKERULES=$TOSROOT/support/make/Makerules

export
CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.export
PYTHONPATH=$TOSROOT/support/sdk/python
```

Architectures

- AVR
  - mica2, mica2dot
  - micaz
  - btnode
  - IRIS
- ARM
  - imote2
- MSP430
  - telosb, sky
  - shimmer
  - eyesIFX
  - tinynode
  - epic
- 8051
  - CC2430
  - CC1110/CC1111

Layout

```
+ tinyos-2.x
  + apps
  + docs
  + support
  + tools
  + tos
```

Layout

```
+ apps
  + Blink
  + Null
  + RadioCountToLeds
  + MultihopOscilloscope
  + tests
    + ...
  + ...
+ docs
+ support
+ tools
+ tos
```

Layout

```
+ apps
+ docs
  + html
  + pdf
  + txt
  + ...
+ support
+ tools
+ tos
```

Layout

```
+ apps
+ docs
+ support
  + make
    - Makerules
    + avr/
    + msp/
    + ...
  + sdk
+ tools
+ tos
```

Layout

```
+ apps
+ docs
+ support
  + make
  + sdk
    + c
    + cpp
    + java
    + python
+ tools
+ tos
```

Layout

```
+ support
  + sdk
    + c
      + blip
      + sf
    + cpp
      + sf
    + java
      - tinyos.jar
    + python
      + tinyos
      - tos.py
```

Layout

```
+ apps
+ docs
+ support
+ tools
+ tos
  + chips
  + interfaces
  + lib
  + platforms
  + sensorboards
  + systems
  + types
```

Layout

```
+ tos
  + chips
    + atm128
    + msp430
    + pxa27x
    + cc2420
    + cc1000
    + at45db
    + stm25p
    + sht11
    + ...
```

Layout

```
+ tos
  + chips
  + interfaces
    - Boot.nc
    - SplitControl.nc
    - StdControl.nc
    - ...
  + lib
  + platforms
  + sensorboards
  + systems
  + types
```

Layout

```
+ tos
  + lib
    + net
    + printf
    + timer
    + tosthreads
    + serial
      - SerialActiveMessageC.nc
      - SerialAMSenderC.nc
      - SerialAMReceiverC.nc
      - ...
    + ...
```

Layout

```
+ tos
  + lib
    + net
      + ctp
      + 4bitle
      + drip
      + Deluge
      + dip
      + blip
      + ...
```

Layout

```
+ tos
  + systems
    - AMReceiverC.nc
    - AMSenderC.nc
    - MainC.nc
    - LedsC.nc
    - TimerMilliC.nc
    - ...
```

Layout

```
+ tos
  + chips
  + interfaces
  + lib
  + platforms
  + sensorboards
  + systems
  + types
    - TinyError.h
    - messssage.h
    - ...
```

Applications

DisseminationDemo

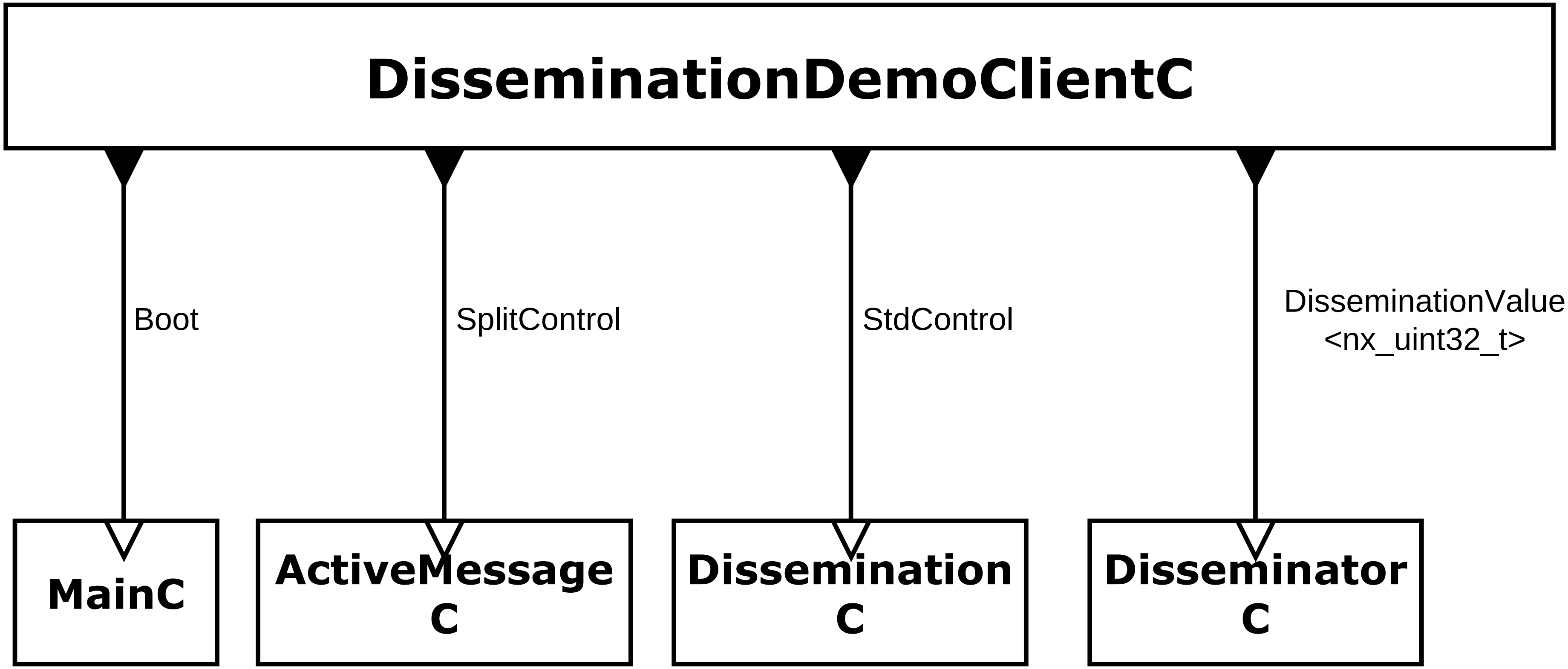
CollectionDemo

DisseminationDemo

DisseminationDemo

- DisseminationDemoClient
  - start the radio
  - start Drip
  - when a new value is received print its contents
- DisseminationDemoServer
  - start the radio
  - start Drip
  - start a periodic timer
  - on each firing or the timer increment a counter and disseminate it

DisseminationDemoClient



DisseminationDemoClient

- Interfaces
  - Boot
  - StdControl
  - SplitControl
  - DisseminationValue<t>

- Components
  - MainC
  - ActiveMessageC
  - DisseminationC
  - DisseminatorC

tos/interfaces/Boot.nc

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

tos/interfaces/StdControl.nc

```
interface StdControl
{
  command error_t start();
  command error_t stop();
}
```

tos/interfaces/SplitControl.nc

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

tos/lib/net/DisseminationValue.nc

```
interface DisseminationValue<t> {
  command const t* get();
  command void set(const t*);
  event void changed();
}
```

tos/system/MainC.nc

```
configuration MainC {
  provides interface Boot;
  uses interface Init as SoftwareInit;
}

implementation {
 ...
}
```

tos/platforms/telosa/ActiveMessageC.nc

```
configuration ActiveMessageC {
  provides {
    interface SplitControl;
    ...
  }
}

implementation {
 ...
}
```

tos/lib/net/drip/DisseminationC.nc

```
configuration DisseminationC {
  provides interface StdControl;
}

implementation {
 ...
}
```

tos/lib/net/drip/DisseminatorC.nc

```
generic configuration DisseminatorC(typedef t,
                                    uint16_t key) {
  provides interface DisseminationValue<t>;
  provides interface DisseminationUpdate<t>;
}

implementation {
 ...
}
```

Makefile

```
COMPONENT=DisseminationDemoClientAppC

CFLAGS += -I%T/lib/net
CFLAGS += -I%T/lib/net/drip
CFLAGS += -I%T/lib/printf

include $(MAKERULES)
```

Commands

```
$ make telosb

$ make telosb install,42

$ tos-dump.py serial@/dev/ttyUSB0:115200
```

Summary

```
tos/interfaces/Boot.nc
tos/interfaces/StdControl.nc
tos/interfaces/SplitControl.nc

tos/system/MainC.nc
tos/platforms/telosa/ActiveMessageC.nc
tos/lib/net/drip/DisseminationC.nc
tos/lib/net/drip/DisseminatorC.nc
```

DisseminationDemoClientAppC.nc

```
configuration DisseminationDemoClientAppC { }

implementation
{
  components MainC;
  components DisseminationC;
  components new DisseminatorC(nx_uint32_t, 2009);
  components DisseminationDemoClientC;
  components ActiveMessageC;

  DisseminationDemoClientC.Boot -> MainC;
  DisseminationDemoClientC.DisseminationStdControl -> DisseminationC;
  DisseminationDemoClientC.DisseminationValue -> DisseminatorC;
  DisseminationDemoClientC.RadioSplitControl -> ActiveMessageC;
}
```

DisseminationDemoClientC.nc

```
module DisseminationDemoClientC
{
  uses {
    interface Boot;
    interface DisseminationValue<nx_uint32_t>;
    interface StdControl as DisseminationStdControl;
    interface SplitControl as RadioSplitControl;
  }
}

implementation
{
  nx_uint32_t counter;

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

  ...

}
```

DisseminationDemoClientC.nc

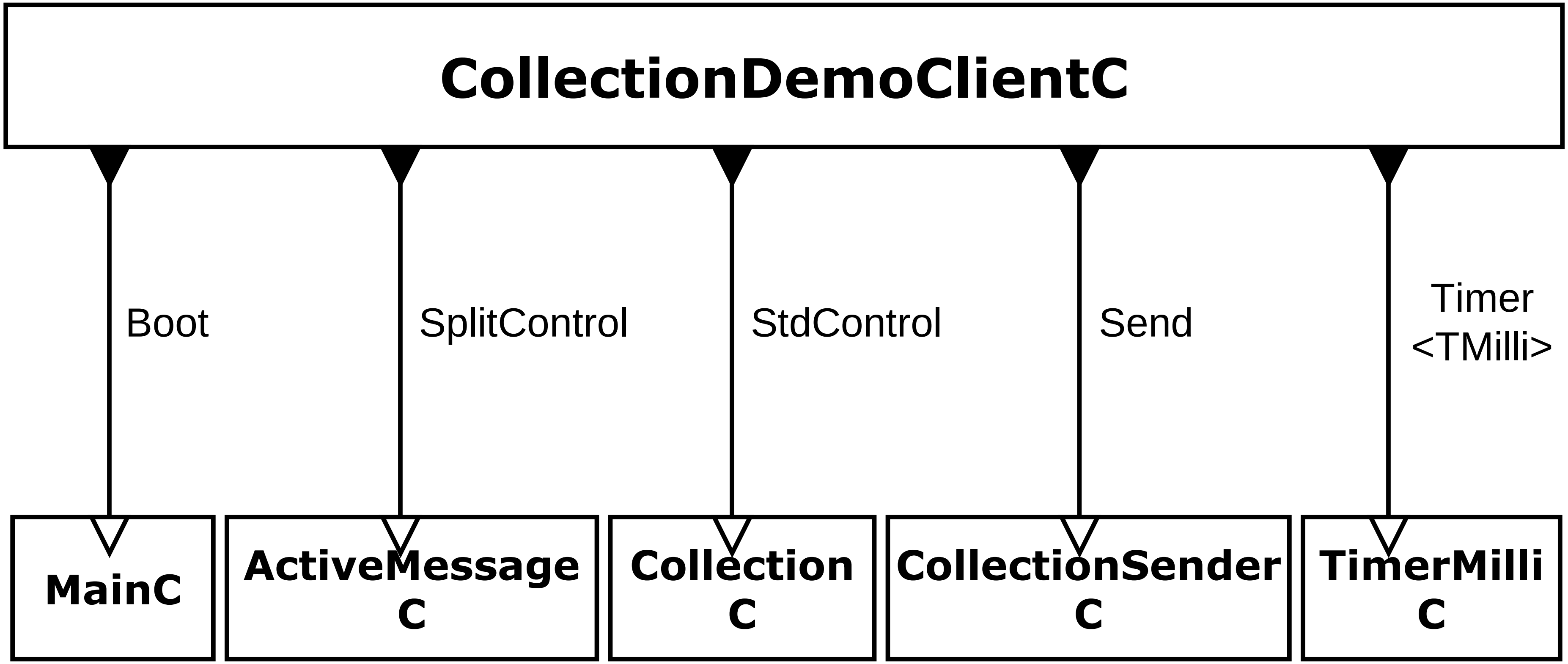
```
module DisseminationDemoClientC
{
   ...
}

implementation
{

  ...

  event void RadioSplitControl.startDone(error_t error)
  {
    call DisseminationStdControl.start();
  }

  event void DisseminationValue.changed()
  {
    printf("R: %lu\n", *(call DisseminationValue.get()));
    printfflush();
  }

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

CollectionDemo

CollectionDemo

- CollectionDemoClient
  - start the radio
  - start CTP
  - start a periodic timer
  - on each firing or the timer increment a counter and sent it over CTP
- CollectionDemoServer
  - start the radio
  - start CTP
  - when a new value is received print its contents

CollectionDemoClient



CollectionDemoClient

- Interfaces
  - Boot
  - StdControl
  - SplitControl
  - Send
  - Timer<TMilli>

- Components
  - MainC
  - ActiveMessageC
  - CollectionC
  - CollectionSenderC
  - TimerMilliC

CollectionDemoClient

- Interfaces
  - Boot
  - StdControl
  - SplitControl
  - **Send**
  - **Timer<TMilli>**

- Components
  - MainC
  - ActiveMessageC
  - **CollectionC**
  - **CollectionSenderC**
  - **TimerMilliC**

tos/interfaces/Send.nc

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

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

tos/lib/net/ctp/CollectionC.nc

```
configuration CollectionC {
  provides {
    interface StdControl;
    ...
  }
}

implementation {
 ...
}
```

tos/lib/net/ctp/CollectionSenderC.nc

```
generic configuration
CollectionSenderC(collection_id_t collectid) {
  provides {
    interface Send;
    interface Packet;
  }
}

implementation {
 ...
}
```

tos/system/TimerMilliC.nc

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

implementation {
 ...
}
```

Makefile

```
COMPONENT=CollectionDemoClientAppC

CFLAGS += -I%T/lib/net
CFLAGS += -I%T/lib/net/ctp
CFLAGS += -I%T/lib/net/4bitle
CFLAGS += -I%T/lib/printf

include $(MAKERULES)
```

Summary

```
tos/interfaces/Boot.nc
tos/interfaces/StdControl.nc
tos/interfaces/SplitControl.nc
tos/interfaces/Send.nc
tos/lib/timer/Timer.nc

tos/system/MainC.nc
tos/system/TimerMilliC.nc
tos/platforms/telosa/ActiveMessageC.nc
tos/lib/net/ctp/CollectionC.nc
tos/lib/net/ctp/CollectionSenderC.nc
```

CollectionDemoClientAppC.nc

```
configuration CollectionDemoClientAppC { }

implementation
{
  components MainC;
  components ActiveMessageC;
  components CollectionC;
  components new CollectionSenderC(16);
  components new TimerMilliC() as Timer;
  components CollectionDemoClientC;

  CollectionDemoClientC.Boot -> MainC;
  CollectionDemoClientC.RadioSplitControl -> ActiveMessageC;
  CollectionDemoClientC.CollectionStdControl -> CollectionC;
  CollectionDemoClientC.Send -> CollectionSenderC;
  CollectionDemoClientC.Timer -> Timer;
}
```

CollectionDemoClientC.nc

```
module CollectionDemoClientC
{
  uses {
    interface Boot;
    interface SplitControl as RadioSplitControl;
    interface StdControl as CollectionStdControl;
    interface Send;
    interface Timer<TMilli>;
  }
}

implementation
{
  message_t smsg;

  typedef nx_struct {
    nx_uint8_t string[8];
    nx_uint16_t counter;
  } name_t;
  name_t *name;

  ...
}
```

CollectionDemoClientC.nc

```
module CollectionDemoClientC
{
  ...
}

implementation
{

  ...

  event void Boot.booted()
  {
    name = call Send.getPayload(&smsg, sizeof(name_t));
    strcpy((char*)name->string, "name");
    name->counter = 0;
    call RadioSplitControl.start();
  }

  ...

}
```

CollectionDemoClientC.nc

```
module CollectionDemoClientC
{
  ...
}

implementation
{

  ...

  event void RadioSplitControl.startDone(error_t error)
  {
    call CollectionStdControl.start();
    call Timer.startPeriodic(1024);
  }


  ...

}
```

CollectionDemoClientC.nc

```
module CollectionDemoClientC
{
  ...
}

implementation
{

  ...

  event void Timer.fired()
  {
    error_t error;
    name->counter++;
    error = call Send.send(&smsg, sizeof(name_t));
    printf("S: %d %d\n", name->counter, error);
    printfflush();
  }

  event void Send.sendDone(message_t* msg, error_t error) { }
  event void RadioSplitControl.stopDone(error_t error) { }
}
```

Code available at

http://docs.tinyos.net/index.php/Ipsn2009-tutorial

The End.