What the Second Generation Holds

Philip Levis, David Gay, David Culler, Vlado Handziski, Jan Hauer, Cory Sharp, Ben Greenstein, Jonathan Hui, Joe Polastre, Robert Szewczyk, Kevin Klues, Gilman Tolle, Lama Nachman, Adam Wolisz, and a few more...
In the Beginning
(1999)

- Sensor networks are on the horizon...
- ... but what are they going to do?
  - What problems will be important?
  - What will communication look like?
  - What will hardware platforms look like?
- Having an operating system is nice...
- ... but how do you design one with these uncertainties?
The TinyOS Goals
(ASPLOS 2000)

- Allow high concurrency
- Operate with limited resources
- Adapt to hardware evolution
- Support a wide range of applications
- Be robust
- Support a diverse set of platforms
TinyOS Today

- Ported to >15 different platforms
  - USA, Ireland, Switzerland, Russia, Korea, Australia, Germany, etc.
- Runs on hundreds of thousands of nodes
- 500,000 lines of code
  - 60,000 core (Berkeley, Intel, TU Berlin, Stanford)
  - 440,000 add-ons (larger community)
TinyOS Basics

- A program is a set of components
  - Components can be easily developed and reused
    - Adaptable to many application domains
  - Components can be easily replaced
  - Components can be hardware or software
    - Allows boundaries to change unknown to programmer
- Hardware has internal concurrency
  - Software needs to be able to have it as well
- Hardware is non-blocking
  - Software needs to be so as well
The TinyOS Goals
(ASPLOS 2000)

- Allow high concurrency
- Operate with limited resources
- Adapt to hardware evolution
- Support a wide range of applications
- Be robust
- Support a diverse set of platforms
The T2 Agenda

- Build on 6 years of experience
- “Clean slate” design
  - Keep and adapt what works
  - Remove the historical artifacts
  - Clean up the problematic edge cases
- TinyOS is larger than Berkeley: build and foster community involvement.
The Issues Ahead

- **Support a wide range of applications**
  - Building small applications is easy
  - Building large applications is possible, but hard

- **Be robust**
  - New hardware has punched holes in assumptions
  - Many unforeseen interactions

- **Support a diverse set of platforms**
  - Trend is to have 1–2 core supported platforms
  - Other groups write new platforms by copying/rewriting a lot of code
Outline

- Future directions for TinyOS
- T2 platforms
- Three technical approaches
- Engendering a community
A platform is a collection of chips

<table>
<thead>
<tr>
<th>Platform</th>
<th>MCU</th>
<th>Radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>mica2</td>
<td>ATMega128</td>
<td>CC1000</td>
</tr>
<tr>
<td>micaZ</td>
<td>ATMega128</td>
<td>CC2420</td>
</tr>
<tr>
<td>Telos</td>
<td>MSP430</td>
<td>CC2420</td>
</tr>
<tr>
<td>eyes</td>
<td>MSP430</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Chip implementations are platform independent

Platforms provide adapter code
Outline

- Future directions for TinyOS
- T2 platforms
- Three technical approaches
- Engendering a community
Three Approaches

- Telescoping abstractions
  - Present a true spectrum of abstraction layers, from the most general and portable to the most efficient and platform-specific

- Static binding and allocation
  - Anything and everything that can be determined statically, should be

- Partial Virtualization
  - Three classes of resources: virtualized
Telescoping Abstractions

- Many fine-grained layers of increasing power and specificity

ActiveMessageC

CC2420ActiveMessageC

CC2420RadioC

Active Message
CSMA
802.15.4 Packet
802.15.4 Specific
**Static Binding**

- nesC components can only interact through interfaces

```plaintext
interface SendMsg {...}

configuration MyAppC {
    uses interface SendMsg;
}

configuration CommC {
    provides interface SendMsg;
}
```

MyAppC.SendMsg -> CommC.SendMsg
Static Binding, Continued

- Run-time vs. compile time parameters

```plaintext
interface CC2420Register {
    command uint16_t read(uint8_t reg);
    command uint8_t write(uint8_t reg, uint16_t val);
    command uint8_t strobe();
}
component CC2420C {
    provides interface CC2420Register;
}

interface CC2420StrobeReg {
    command uint8_t strobe();
}
component CC2420C {
    provides interface CC2420StrobeReg as SNOP;
    provides interface CC2420StrobeReg as STXONCCCA;
    ....
}
```
Static Allocation

- T2 has the same basic concurrency model
  - Tasks, sync vs. async
- T2 changes the task semantics
  - TinyOS 1.x: post() can return FAIL, can post() multiple times (shared slots)
  - T2: post returns FAIL iff the task is already in the queue (single reserved slot per task)
Partial Virtualization

- Three classes of OS resources
  - Physical and dedicated
  - Physical and shared
  - Virtualized
- Power management
- Arbitration
Three Classes

- **Client**
- **Abstraction**
- **Arbiter**
- **Virtualizer**

- **Physical and Dedicated**
- **Physical and Shared**
- **Virtualized**
Examples of Classes

- CC2420 Radio
- Physical and dedicated
  - Timer registers for backoff
- Physical and shared
  - Bus for communicating with radio
- Virtualized
  - Sending packets
Outline

- Future directions for TinyOS
- T2 platforms
- Three technical approaches
- Engendering a community
Community Building

- Documentation
  - Tutorials to get started
  - TEPs for system details
  - Programming guide for developers

- Working Groups
  - 2.x core (chair: Philip Levis)
  - net2 (chair: Rodrigo Fonseca)
  - Alliance (chair: David Culler)
Documentation

- Good documentation has a tremendous effect on community involvement
  - Reduced barrier to entry
  - Reduced “I don’t get it” antipathy
- Good documentation forces you to specify the system precisely
  - Opens decisions up to review and discussion
  - Provides a checking mechanism (e.g., “ActiveMessageC does not seem to follow the guidelines in TEP 107”)
Working Groups

- Small groups (e.g., 10) of contributors collaborating on a set of issues
  - 2.x: hardware-independent core abstractions
  - net2: multihop protocols
  - alliance: administrative structure, IP, etc.
- Each WG has its own membership policy
- Output are TEPs
  - Documentary, Informational, Experimental, or Best Common Practice
The State of the Art Today

- Sensor networks are different
- Revisit old problems and assumptions
  - Different resource constraints
  - Different design tradeoffs
  - Sometimes new problems, sometimes not
- Support a huge range of requirements, applications, and abstractions
- Open the doors to innovation and community involvement
Questions