The Emergence of Networking Abstractions and Techniques in TinyOS

Sam Madden
MIT CSAIL
madden@csail.mit.edu

With Phil Levis, David Gay, Joe Polastre, Rob Szewczyk, Alec Woo, Eric Brewer, and David Culler

UC Berkeley + Intel Research Berkeley
The Rise of Sensor Networks

- Sensornets:
  - tiny, cheap ⇒ Many, limited resource devices
  - embedded ⇒ Non-interactive, self-maintaining
  - power-constrained ⇒ Lifetime is a critical constraint
  - radio-equipped ⇒ Ad-hoc networking issues

- Promise to revolutionize industrial, scientific monitoring

- Emerging experimental platform: TinyOS + Motes
  - In use by hundreds of research groups and companies

Is sensornet system design (exemplified by TinyOS) substantially different than system design in conventional environments?
A Brief History of TinyOS

- TinyOS: initial versions developed at Berkeley in 2000
  - Perl scripts, cruft
- Moved to SourceForge in Summer 2001
- Intel-Berkeley heavily involved in development
  - Real programming language (nesC)
  - Many tools (simulators, gcc support, etc.)
- Now a large, community supported project
  - Berkeley, Intel, UCLA, Vanderbilt largest contributors
Methodology

• Using CVS, we study TinyOS evolution
  - Records covering 3 years, 10,000+ commits
• Focus on networking:
  - Software abstractions
    • General (e.g., active messages)
    • Application specialized (e.g., power management)
    • In-flux (e.g., epidemic dissemination protocols)
  - Unusual system design techniques, e.g.:
    • Cross-layer control
    • Static allocation discipline
• Highlight successes and failures
• Not an analysis of programming model
Outline

• TinyOS and Motes
• Single Hop Networking
• Multihop Networking
• Network Services
• Lessons and Conclusions
Outline

- TinyOS and Motes
- Single Hop Networking
- Multihop Networking
- Network Services
- Lessons and Conclusions
The Mote Platform

- **3 Generations:** Rene, Mica, Mica2(Dot)
  - **Rene:**
    - 512 Bytes RAM
    - 8K Code
    - 4Mhz
    - 10 kbps Radio
  - **Mica:**
    - 4K RAM
    - 128K Code
    - 4 Mhz
    - 40 kbps Radio
  - **Mica2(Dot):**
    - 4K RAM
    - 128K Code
    - 4/7 Mhz
    - 38.6 kbps Radio

- **Non-Berkeley Platforms:**
  - Intel iMote
  - BTNNode
  - **Intel iMote:**
    - 64K RAM
    - 512K Code
    - 12 Mhz
    - 38.6 kbps Radio
  - **BTNNode:**
    - 4K RAM
    - 128K Code
    - 8 Mhz
    - 460 kbps Radio
TinyOS

- Programming model and language (nesC)
- Set of software abstractions
  - Single and multi-hop communication
  - Power management
  - Time Synchronization
  - Flash file system, timers, clocks, etc.
- Simple concurrency model
  - Hardware events (interrupts): fire asynchronously
    - E.g., timers, peripheral activity, reset
  - Tasks: “posted” to a queue (by events), execute serially
- No “kernel”; single application at a time
  - Each application includes its own set of OS services
• **Component-based modularity**
  - Components provide and require interfaces
  - Configurations wire components + configurations
  - Provides for easy composition, interposition

• **Event-driven**
  - Single (interruptable) thread of execution
    - Dictated by serial ordering of tasks
  - Tasks must be non-blocking, short-lived
    - Instead of blocking, use timer events or other interrupts
  - **Upside**: Mostly synchronization-free, only one stack
  - **Downside**: Complicates programs
Applications and Requirements

• Habitat Monitoring
  - E.g., TinyDB
  - Many-to-one routing
  - Collaborative, low sample rates, loose time sync, power management

• Localization
  - E.g., Vanderbilt shooter localization
  - Precise time sync, high sample rates

• Tracking
  - E.g., NEST Pursuer-Evader Games demo
  - Localization, any-to-any routing/collaboration
Outline

- TinyOS and Motes
- **Single Hop Networking**
- Multihop Networking
- Network Services
- Lessons and Conclusions
Single Hop Networking

- **Fundamental link-layer primitive**
  - Broadcast a message from A to nearby motes
  - Trivial: send from A to its neighbor B
  - General service since TinyOS 0.1
- **Active Messages (AM)**
  - Message handler dispatch based on AM type
- **Issues/Tensions:**
  - Bit/byte level timing and software decomposition
  - Hardware / software boundary
- **Relationship to time stamping and acknowledgments**
  - MAC layer (CSMA/TSMA/Hybrid)

```c
Interface SendMsg {
    command result_t send(uint16_t addr, uint16_t len, TOS_MsgPtr msg);
    event result_t sendDone(TOS_MsgPtr msg, result_t success);
}

Interface ReceiveMsg {
    event TOS_MsgPtr receive(TOS_MsgPtr msg);
}
```
The Rene Radio Stack

- RFM TR1000 Radio
- Hardware interface: read/write bit
- Software manages:
  - Timer interrupt to read/write bits
  - SEC/DED and CRC coding, DC balancing
- Interrupt rate limits to 10kbps
  - Encode/decode in tasks to limit per-interrupt time
  - 1-byte buffer limits task runtime to 1 byte time (~1.8ms)
- Low-power listening
  - Sample radio periodically, wake on transmission
- Synchronous acknowledgments
  - Sender and receiver switch roles without reacquiring channel
- Enabled by quick start up and switching times
The Mica2 Stack

- **Chipcon CC1000 Radio**
- 1-byte buffer with:
  - HW encoding
  - Interrupt per byte
  - CRC computation/checking in event handlers
  - Operation up to 38.6 kbps
  - 1 task per message
- **Synchronous acknowledgements are impractical**
  - Long send/receive switch time
    - another sender could acquire channel
- **Low-power listen less effective than on RFM**
  - On/off times much longer; can’t sample channel as quickly
Trends & Observations

- **SW/HW boundary moving towards HW**
  - 802.15.4 provides packet-level interface
    - Encryption, authentication, acknowledgments, CRC
  - Decreases CPU load, software complexity
  - Decreases flexibility
    - E.g., link-layer acks infeasible on Mica2

- **Fine line between useful and over-specified**
  - E.g., bluetooth inappropriate for sensornets [Leopold et al, Sensys 2003]
Outline

• TinyOS and Motes
• Single Hop Networking
• Multihop Networking
• Network Services
• Lessons and Conclusions
3 Types of Multihop Networking

• **Many-to-one**
  - “Tree-based routing”
  - Recently: general implementation

• **One-to-many**
  - Broadcast flood
  - Epidemic/gossip
  - Hybrid
  - Largely application-specific

• **Many-to-many**
  - Geographic routing
  - Landmark-based full routes

```c
interface Send {
    command result_t send(TOS_MsgPtr msg,
                          uint16_t length);

    command void* getBuffer(TOS_MsgPtr msg,
                            uint16_t* length);

    event result_t sendDone(TOS_MsgPtr msg,
                            result_t success);
}

interface Intercept {
    event result_t intercept(TOS_MsgPtr msg,
                              void* payload,
                              uint16_t payloadLen);
}
```
**Many-to-1: AMROUTE vs. MultihopRouter**

- **AMRoute: Proto-routing**
  - Pick first neighbor who transmits beacon as parent

- **MultihopRouter**
  - Estimate link-quality to neighbors
    - Using neighbor beacons
  - Pick path of fewest hops
    - Or fewest transmissions

### Problems:
Bad Parent Selection
Asymmetric Links
Adaptation vs. Stability

<table>
<thead>
<tr>
<th>Node D</th>
<th>Neigh</th>
<th>Qual</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>.75</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>.66</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>.45</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>.82</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node C</th>
<th>Neigh</th>
<th>Qual</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>.44</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>.53</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>.35</td>
<td></td>
</tr>
</tbody>
</table>
Broadcast Floods and Epidemics

- **Common app need: reliable dissemination**
  - E.g., TinyDB queries, PEG parameters
- **Floods**
  - Used extensively
  - Effective way to reach most nodes
  - Randomize retransmits to avoid collisions
- **Epidemics**
  - Nodes “infect” neighbors with data, programs
  - Reach all nodes eventually
  - Requires careful tuning of transmit rate
- **Hybrid**
  - Flood + epidemic patchup
  - E.g., tinydb, network reprogramming algorithms
Trends & Observations

• Standard multihop interface has emerged
  - Including promiscuous “intercept” interface

• Common abstractions
  - Cross-layer neighbor table
    • Link state (e.g., qualities)
    • Network state (e.g., parent, depth, location)
  - Link quality estimation
    • Appears in MultihopRoute, DSDV, TinyDiffusion
  - Forwarding queue; app-configurable length

• Surprising
  - No receive queues
  - Segmentation/framing generally done by applications
  - Most apps are many-to-one
Outline

- TinyOS and Motes
- Single Hop Networking
- Multihop Networking
- Network Services
  - Time Sync, Power Management
- Lessons and Conclusions
Power Management & Scheduling

- **HPLPowerManagement** monitors processor state
  - Powers down when not in use
  - Brittle, platform specific technique
- Application uses `stop` interface to power down components
  - Uses timers to power back up
- **Common forms of power management:**
  - Low-power listening,
  - Scheduled operation (require synchronization)
- Power management is app-specific, with simple OS mechanism
  - Application knows when it should be on or off
- Versus traditional mobile environments
  - Where needs multiple apps, interactivity requirements conflict
Time Synchronization

• Many implementations
  - Vanderbilt, UCLA, Berkeley
  - Most rely on low-level events from radio
    • Cross-layer optimization

• Building a general purpose time sync is hard
  - Not for the reasons the research community is concerned with
  - Instead, due to interactions with application/OS timers
    • Similar to NTP observations

• Application controlled time-sync much easier
  - Application knows when changes are safe
  - E.g., TinyDB adjusts length of sleep intervals
Trends & Observations

• Application control of OS mechanism
  - Single app makes this more feasible
  - Low-interactivity enables aggressive policies
    • Tailored to each application

• Power management surprisingly rare in apps
  - Many apps are “demos”, not “deployments”
Outline

• TinyOS and Motes
• Single Hop Networking
• Multihop Networking
• Network Services
• Lessons and Conclusions
Widespread Abstractions

• AM, single, multihop interfaces are “standards”
  - Several link/many-to-one network implementations
• Time sync, power management: app specific policy
  - Standardized mechanisms
• Many abstractions still in-flux
  - E.g., many-to-many routing, epidemic protocols
• Some abstractions have never emerged
  - Where is distributed cluster formation?
Interesting Development Techniques

- Cross layer control
- Scheduling vs. snooping
- Static Resource Allocation
Conclusions

- So what’s really different?
  - Limited memory constrains software design
    - E.g, RAM limitations imply a static discipline
  - Timing sensitive net services imply cross-layer control
    - E.g, time-sync, power-scheduling, localization
  - Single, non-interactive app
    - Services are different from their laptop counterparts
  - In-network processing vs. end-to-end connectivity
    - Traditional networking focuses on the latter

- Conclusion: TinyOS isn’t solely a product of a crippled hardware platform!