# The Emergence of Networking Abstractions and Techniques in TinyOS

**Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer and David Culler**

Seminar

Computational Systems

Michael Holzmann

# The Emergence of Networking Abstractions and Techniques in TinyOS

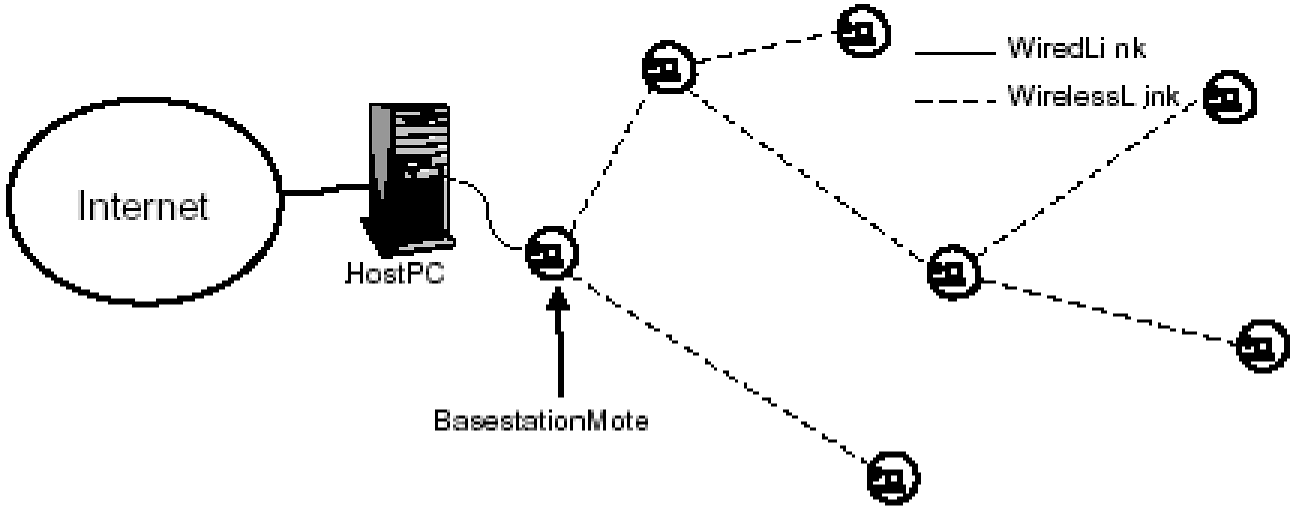
# *Wireless* Sensor Networks

# Overview

- What are Sensor Networks
- Typical Applications and Application Requirements
- TinyOS – an OS for wireless sensor networks
- Emergence of Networking Abstractions in TinyOS
  - Single-Hop Communication
  - Multi-Hop Communication
  - Networking Services
- Common Design Techniques
- Conclusion

# Sensor Networks

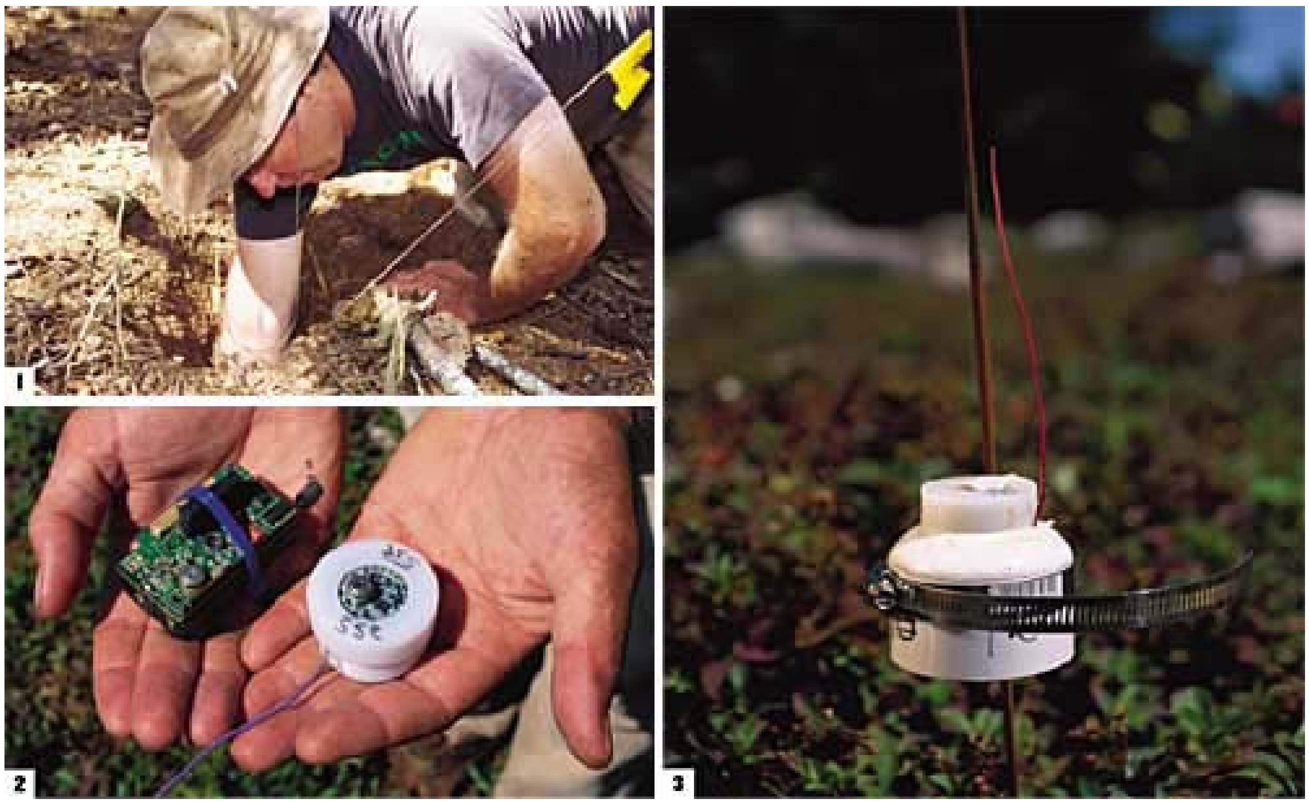
- Networked systems of small embedded computers



# Application Examples

- Habit/Habitat Monitoring



# Shooter Localization

- Localize origin of a bullet in urban setting

# Pursuer / Evader

- Vehicle Tracking

# Application requirements

- Habitat monitoring
  - Low energy consumption
- Shooter localization
  - High sampling rate
  - Time synchronization of nodes
- Pursuer / Evader
  - Localization of mobile nodes
  - More advanced routing

# General EmNet Requirements

- Networking issues are at the core of the design of EmNets:
- Communication dominates energy budget
  - Multi-hop communication
- Ad-hoc networks
  - Mobility
  - Deploy large number of nodes without configuration
  - -> no static routing infrastructure possible
- New approaches to network design are required

# TinyOS

- Develped at UC Berkeley
- Operating System for Sensor Networks
  - Limited ressources
  - Concurrency intensive operation
- Goal
  - Allow OS to adapt to hardware diversity
  - Still allow applications to reuse common services & abstractions

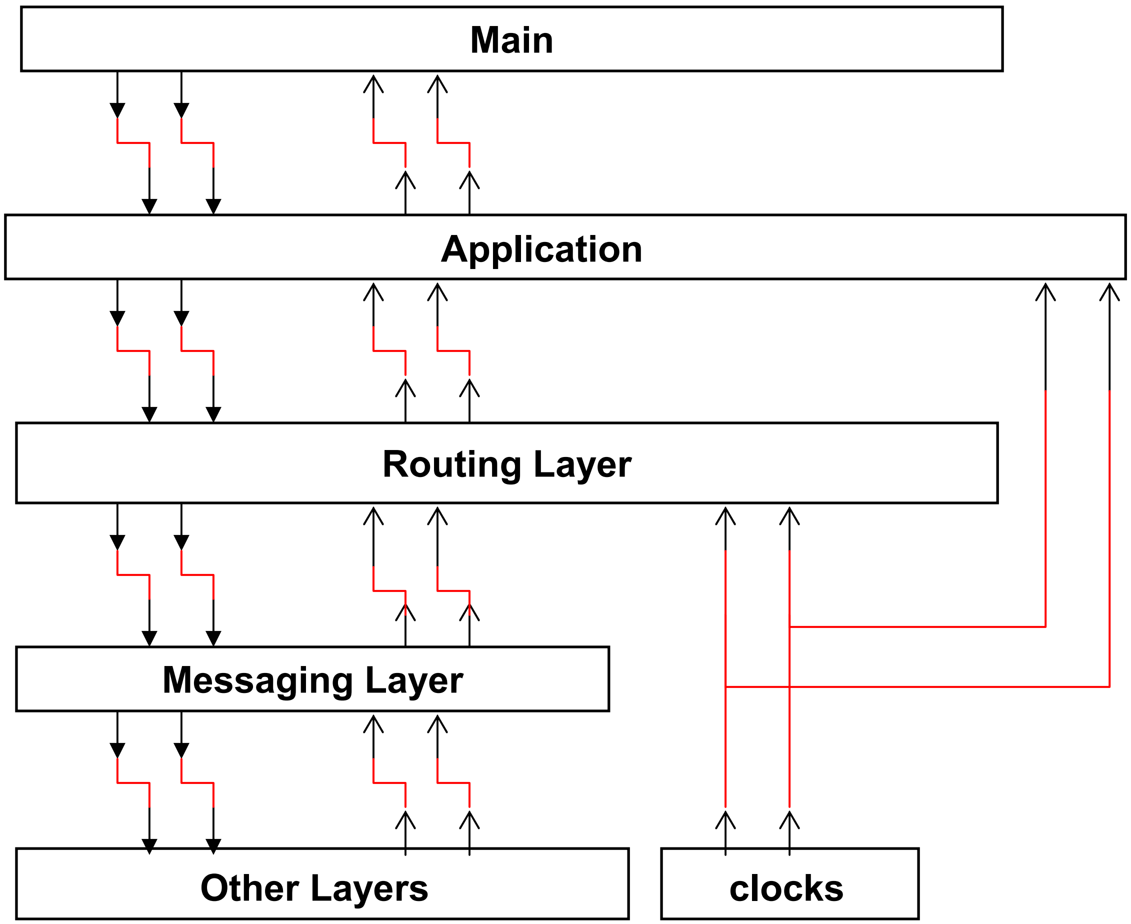
# TinyOS operation

- Has to handle high concurrency
  - Process multiple information flows as opposed to heavy computing
  - Sensor reading, communication, routing, ...
- Interaction with the physical world
  - Real-time requirements
  - E.g. radio communication, sensing data, ...
- Event driven concurrency model
  - Tasks (== deferred execution)
  - Hardware events (interrupts)

# TinyOS modularity

- Based on component model:
- Named components
  - Provide Application Code
  - Implement Interfaces
- Interface declares set of functions and event
  - commands - downcall vs events - upcall
  - Split phase operation (command & event)
  - No blocking (reactivity!)
  - Blocking sequence as state machine
  - Many concurrent operations on single stack (mem!)

# Flow of Calls in the Component Graph



# Emergence of Abstractions in TinyOS

- Examine emerged abstractions and common techniques they exploit – substantially different from GPS?
- Focus on networking abstractions
- Classification into four categories
  - General. OS provides mechanism & policy
  - Specialized. OS provides only meachnism
  - In Flux. Abstraction is part of the application
  - Absent.
- Abstractions moved among those classes as refined.

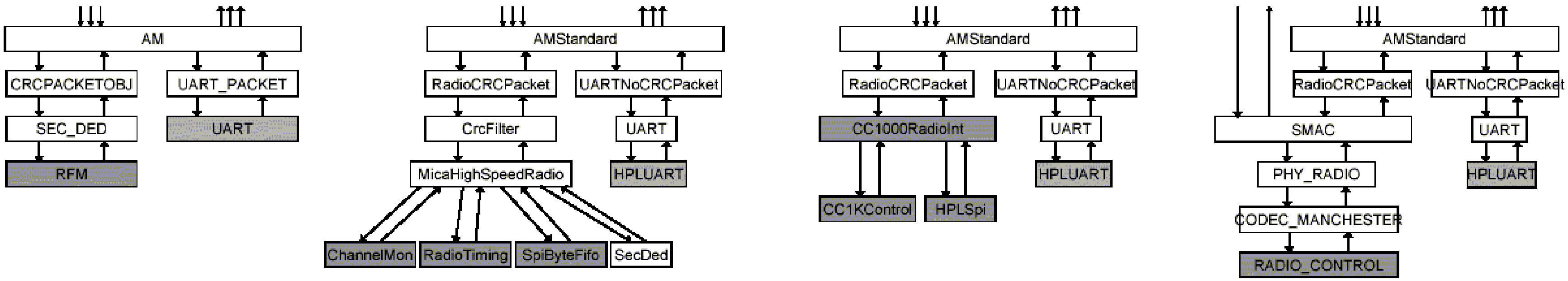
# Single Hop Communication

- Active Messages

```
interface SendMsg { // single-hop networking
  command result_t send(uint16_t addr, uint8_t len,
       TOS_MsgPtr msg);
  event result_t sendDone(TOS_MsgPtr msg, result_t
       success);
}
```

  - Identifier specifies action to be executed on reception
- AM abstraction did not change but the implementation did:
  - Due to changing HW platforms
  - Radio requires high-frequency low-jitter sampling
  - Real time requirements restrict use of tasks
  - Simplified by raising the SW/HW boundary

# Single Hop Network Stacks



# Radio hardware abstractions

- Bit level interface
  - Bit sequence written to transmit pin
  - Receive pin sampled at precise times (timer int)
  - High interrupt rate: handler cannot decode/encode
  - Decoder task scheduled every byte time:
  - No Tasks can run longer than a byte time
- Byte-level hardware
  - Reduced interrupt rate
  - Decode/encoding can be done within the handler
  - Increased task length (packet time)

# Cost of high-level hardware abstractions

- Moving hw/sw boundary changes division of work between tasks and hw events
- Stack performance improves but disallows capabilities enabled by low-level hardware access
- Raising the boundary is not without cost: Useful features become more complex to provide
  - Power management: Low power listening when idle, turn off radio between samples.
  - Synchronous link-layer acknowledgement: sender / receiver swap roles vs ACK packet

# Multi-Hop Communication

- Ad-hoc multi-hop routing
  - Tree-based collection
  - Intra-network routing
  - Dissemination
- Tree-Based Routing
  - Forward packet via parents to root of the tree
  - Key issue: how to discover and maintain routing tree
  - Parent selection algorithms try to:
    - Minimize end-to-end packet loss
    - Total expected transmissions (including retransmissions)
  - Nodes compute quality estimates on incoming links

# Multi-Hop Communication II

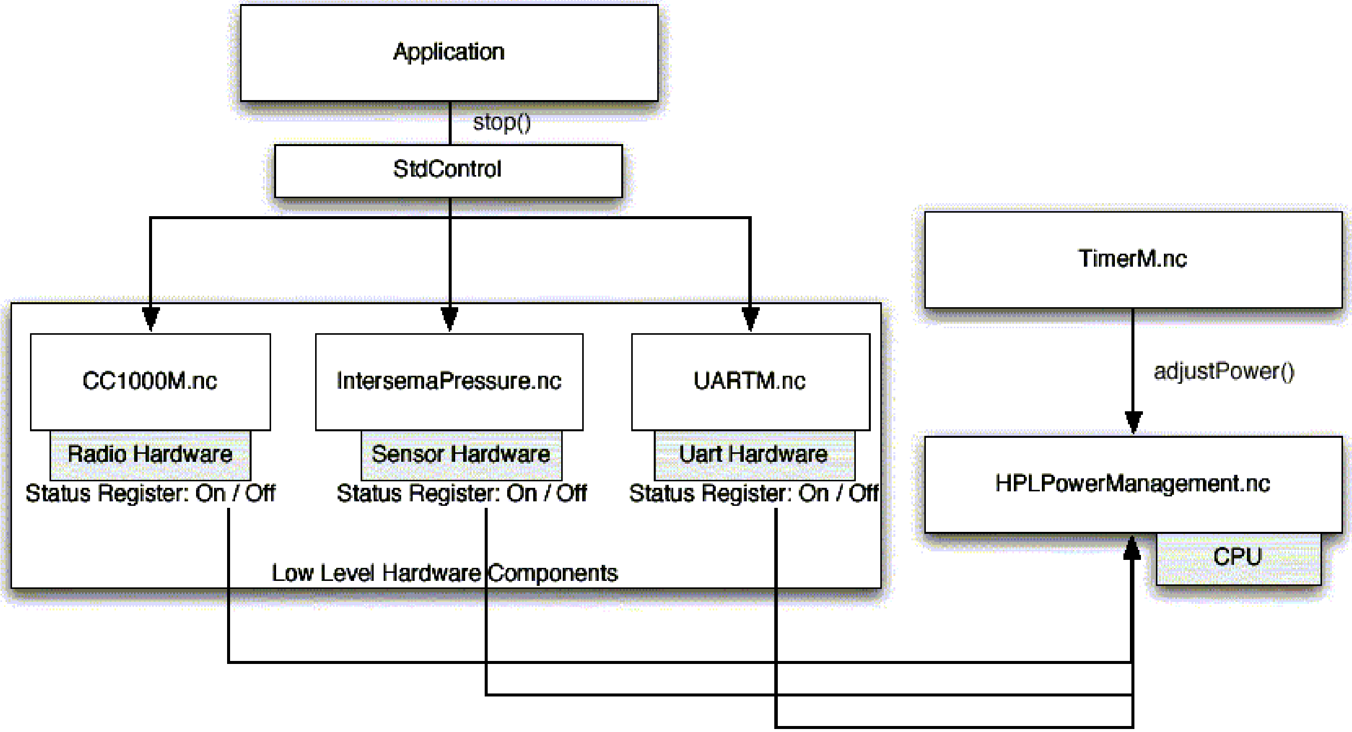
- Intra-Network Routing
  - Is uncommon in TinyOS applications
  - Route discovery and maintainance similar to IP
  - Usually a single route is maintained
- Broadcast Protocols
  - Reliably disseminate data to every node
  - Implementations: Flooding vs Epidemic

# Multi-Hop Common Developments

- Neighborhood discovery and link quality estimation
  - Node adresses, link quality, routing metadata
  - Construct routes, adapt to connectivity changes
- All Implementations built on top of AM abstraction
  - Common Interfaces / Augmentations of AM:
  - Send / Intercept IF
    - Signals packet reception to application
    - Monitored forwarding
    - Broadcast with processing at each hop
  - Pass non locally adressed packets up
    - Link estimation, neighbor table management

# Network Services

- Abstractions to support efficient, low-power networking
- Power Management, Ex. TinyDB







SOFTWARE RESEARCH LAB
Universit□t Salzburg

# Network Services II

- Time Synchronization
  - Ex.: Sensor fusion, Slot Coordination, Communication Scheduling
  - TinyOS provides get/set systemtime and transmit hook
  - Development of general purpose time synchronization was unsuccessful
  - Time Synchronization also appears to emerge as a specialized abstraction

# Abstractions and Common Techniques

- General Abstractions
  - AM abstraction
  - Tree Based Routing (Send/Intercept Interface)
- Specialized Abstractions
  - Power Management
  - Time Synchronization
- In-Flux Abstractions
  - Epidemic Propagation
  - Radio MAC
- Absent Abstractions
  - Distributed Cluster Formation
  - Receive Queues

# Common Techniques

- Communication Scheduling and Snooping
  - Two confliction techniques
  - Applications tend to strike a balance
- Cross-Layer Control
  - Application controls
    - Network services e.g. Time synchronization
    - Power Management
- Static Resource Allocation
  - Buffer allocation

# Conclusion

- Development of Abstractions/Techniques driven by
  - Power Management
  - Limited Ressources
  - Real-Time Constraints
- Most abstractions are still specialized
- Spezialized Implementations offering greater efficiency

# Comments on the Paper

- Commendation
  - Analysis / Discussion Section completes each Chapter
  - Pointing out relevant facts
  - Really good presentation
- Criticism
  - Use of terms / abbreviations without prior definition