CS 525M – Mobile and Ubiquitous Computing Seminar

A Network-Centric Approach to Embedded Software for Tiny Devices

Culler, Hill, Buonadonna, Szewczyk, Woo

Presented by Mike Scaviola

Introduction

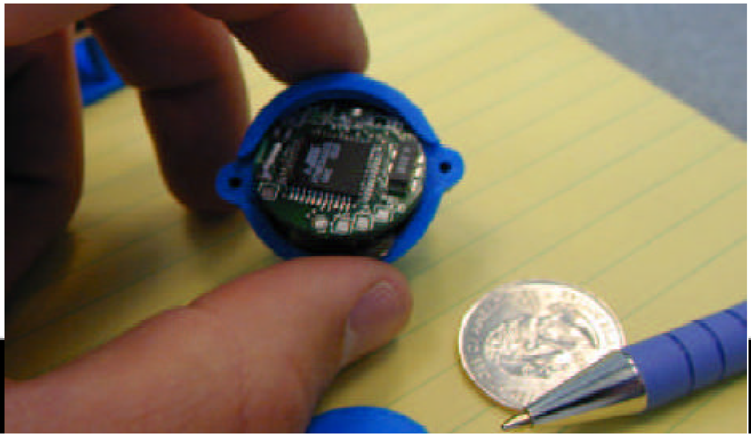
- Traditional embedded systems engineered to a particular task
  - Developed incrementally over generations
  - Controller is just a command processing loop
  - Sized and powered specially for the application
- Examples: Disk drive controller, engine ignition controller

Introduction

- Sensors are a different kind of embedded device
  - Distributed, dynamic, not designed to a specific control path
  - Can communicate to coordinate at a higher level
    - Multihop routing, location sensing
  - Many different tasks sensors can perform
  - Realtime action and long-scale processing

TinyOS

- They developed small RF wireless sensor devices and a tiny operating system
  - 4MHz Atmel AVR 8535 microcontroller
  - Single channel low-power radio
  - 8KB program, .5KB SRAM
- TinyOS: Simple, component-based
  - Framework for managing concurrency in a very limited environment (storage, energy)



TinyOS Concepts

- TinyOS consists of a scheduler and graph of components
  - Each component has an interface and internal implementation
    - Interface has synchronous *commands* and asynchronous *events*
    - Storage *frames*
    - Concurrent *tasks*

Example application

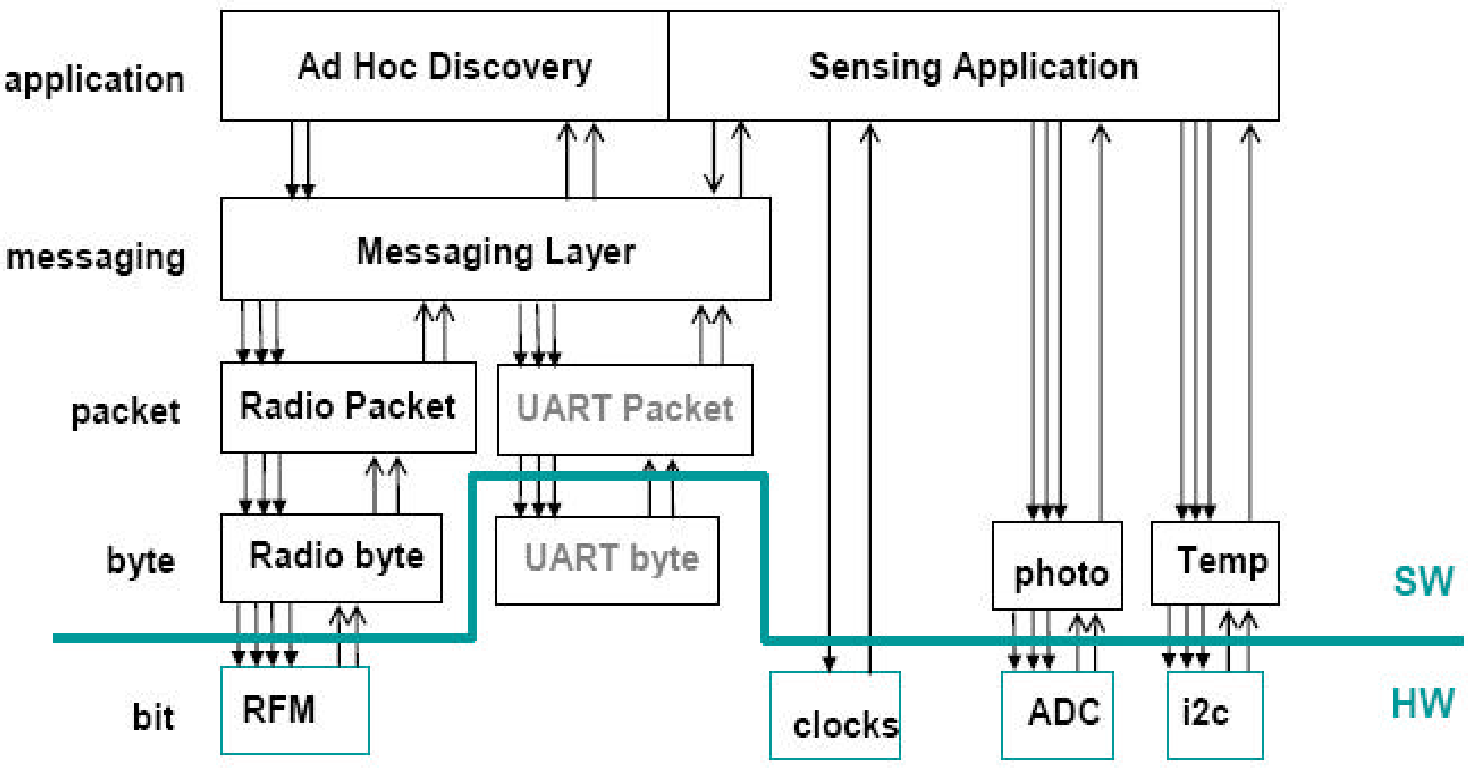


Figure 2. Typical networking application component graph.

Concurrency Model

- Events preempt tasks, tasks don't preempt other tasks
  - Tasks call commands
    - Commands can be accepted or refused (storage constraints, etc)
- Events triggered by hardware interrupts
- TinyOS is non-blocking
  - Components are reentrant state machines, can resume operation after being interrupted.

Application-Level Communications

- Tiny Active Messages
  - Active Message communication model, only smaller
    - Event-driven, has lean communication stack
  - 4 components to initiate AM:
    - Specify data arguments
    - Name handler
    - Request Transmission
    - Detect completion

Application-Level Communications

- Managing Packet Buffers
  - Typically handled by an OS's kernel
  - 3 issues to address:
    - Encapsulating data with header/trailer
      - Holes
    - Determining when buffer can be reused
      - pWn3d! ('0wn3d') by network
    - Providing an input buffer before message has been inspected

Application-Level Communications

- Network discovery and ad hoc routing
  - Uses the Active Messages
  - Node periodically transmits ID and distance to its neighborhood
  - Message handler checks if node is closest, records source, increments distance, retransmits message.
  - Builds a breadth-first spanning tree rooted at the source (typically a gateway node)
  - Packets get routed up the tree to parents (neighbors just discard the packet)

Lower-level Communication Challenges

- Crossing layers without buffering
  - ‘Data pumps’
  - Partition data into subunits, then operate on them at each level, unit-by-unit
  - Components use the frame/command/event framework to make this a reentrant state machine

Lower-level Communication Challenges

- Listening at low power
  - Too much energy spent listening for nothing
  - Periodic and low-power listening!
    - Create time periods when nodes cannot transmit. Then nodes only need to listen part of the time
    - Turn radio on for 30µs of every 300µs
    - How to find out if a node is transmitting?
      - Nodes send preamble of at least 300µs
    - Data length is 56,100µs, so 1% increase in xmit costs

Lower-level Communication Challenges

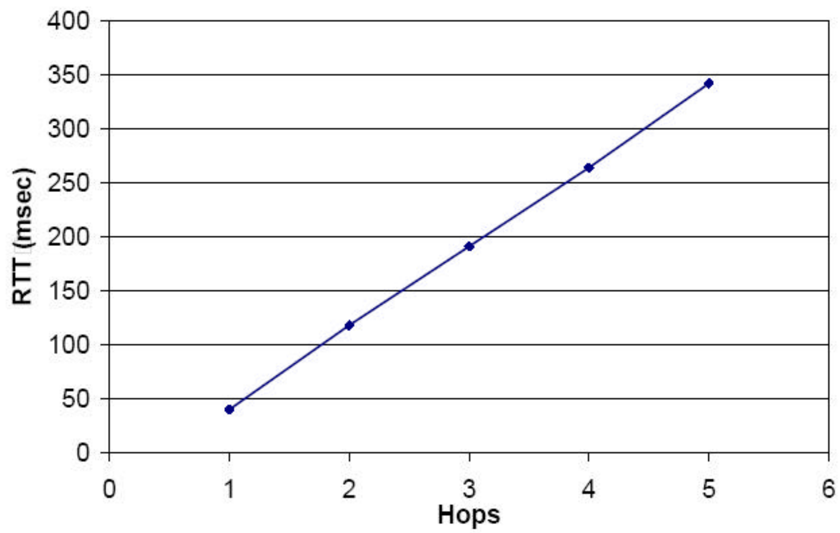
- Physical layer interface
  - Microcontroller is directly connected to the radio
    - Realtime requirements – each bit handled by microcontroller!
  - Uses a bit-level data pump
  - Complex encoding done on each byte takes longer than the transmission time of a *bit*
    - Need to encode next byte while transmitting current byte
  - Reception is tricky. 18-bit sliding window

Lower-level Communication Challenges

- Media Access and Transmission Rate Control
  - Radio doesn't support anything
  - Use CSMA scheme – only TX when idle
    - Random backoff if channel busy
  - Detection of a busy channel might mean that communication patterns of nodes are synchronized. The TX failure can be used as feedback to shift sensor sampling phase and desynchronize.

Evaluation

- •Tiny Active Message component is 322 bytes!
- •10kbps raw bit rate (4b6 encoding)
  - – 833 bytes/sec throughput!
- •Device-device RTT of 78ms



Idle State	5 μAmps
Peak	5 mAmps
Energy per bit	1 μJoule

Table 3. Power and energy consumption measurements.

Conclusion

- Event-driven model interleaves processor between multiple data flows and stack layers
- Tasks provide logical concurrency within the stack
- The approach avoids complexities that the hardware could not otherwise handle (threading, multiple stacks, complex synchronization)
- Allows for high level applications on very limited hardware

Questions?

Who ‘0wnz’ the buffers?