A System Architecture for Networked Sensors

Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kris Pister

http://tinyos.millennium.berkeley.edu
U.C. Berkeley
11/13/2000

Computing in a cubic millimeter:
- Advances in low power wireless communication technology and microelectromechanical sensors (MEMS) transducers make this possible
- How do you combine sensing, communication and computation into a complete architecture?
- What are the requirements of the software?
- How do you evaluate a given design?

Ad hoc sensing
- Autonomous nodes self assembling into a network of sensors
- Sensor information propagated to central collection point
- Intermediate nodes assist distant nodes to reach the base station
- Connectivity and error rates used to infer distance

Organization
- The Vision
- Hardware of today
- Software Requirements
- TinyOS system architecture
- System evaluation

Today’s Hardware
- Assembled from off-the-shelf components
- 4 M\(^2\), 8 bit MCU (ATMEAL)
- 4 2 bytes RAM, 8 K ROM
- 900 M Hz Radio (RF Monolithics)
- 10-100 ft range
- Temperature Sensor & Light Sensor
- LEDS in us
- Serial Port

No dedicated I/O controllers
- Bit by bit interaction with Radio
- Software must process bit every 100\(\mu\)s
- No buffering
  - missed deadline \(\rightarrow\) lost data
  - Must time share granularity of \(\&\) sampling rate

http://tinyos.millennium.berkeley.edu
Key Software Requirements

- Capable of fine-grained concurrency
- Small physical size
- Efficient resource utilization
- Highly modular

TinyOS system architecture

State Machine Programming Model

- System composed of state machines
- Command and event handlers transition a module from one state to another
- Quick, low overhead, non-blocking state transitions
- Many independent modules allowed to efficiently share a single execution context

Simple State Machine Logic

```
Start

Yes

Send Byte Event
bit_cnt = 0

No

bit_cnt++

bit_cnt==8

Done

Don't you have to do computational work eventually?

"Tasks" used to perform computational work
```

TinyOS component model

- Component has:
  - Frame (storage)
  - Tasks (computation)
  - Command and Event Interface
  - Constrained Storage Model allows compile-time memory allocation
  - Provides efficient modularity
  - Explicit interfaces help with robustness

TinyOS – The Software

- Scheduler and graph of components
  - Constrained two-level scheduling model: tasks + events
  - Provides a component based model abstracting hardware specifics from application programmer
  - Capable of maintaining fine grained concurrency
  - Can interchange system components to get application specific functionality
The Application

- The single node application is just another state machine

```
Message_Handler(incoming_message){
  if(sender_is_better_parent()){
    my_parent = sender();
  } else if(I_am_parent_of_sender()){
    forward_message(my_parent, incoming_message);
  }
}
```

- Clock_Event_Handler()

```
check_expire(my_parent);
if(my_parent != null){
  send_data(my_parent);
}
```

Analysis

- Let's take a look at Space, Power, and Time

Space Breakdown...

- Code size for ad hoc networking application

```
<table>
<thead>
<tr>
<th>Component</th>
<th>Code Size (bytes)</th>
<th>Data Size (bytes)</th>
<th>Total Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message Dispatch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-Runtime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Sensor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet Layer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio Interface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Messaging Layer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio Byte Encoder</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Power Breakdown...

- Active: 5 mA, 2 mA, 5 μA
- Idle: 4 mA, 0, 0
- Sleep: 200 μA, 0, 0

- But what does this mean?
  - Lithium Battery runs for 35 hours at peak load and years at minimum load!
  - That's three orders of magnitude difference!
  - A one byte transmission uses the same energy as approx 11000 cycles of computation.

Time Breakdown...

- Components: Packet reception

```
<table>
<thead>
<tr>
<th>Component</th>
<th>Packet Reception Utilization</th>
<th>Energy (μJ/Bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>0.05%</td>
<td>0.20%</td>
</tr>
<tr>
<td>Packet</td>
<td>1.12%</td>
<td>0.51%</td>
</tr>
<tr>
<td>Radio handler</td>
<td>26.87%</td>
<td>12.16%</td>
</tr>
<tr>
<td>Radio decode thread</td>
<td>5.48%</td>
<td>2.48%</td>
</tr>
<tr>
<td>Baud</td>
<td>66.46%</td>
<td>30.08%</td>
</tr>
<tr>
<td>Radio Reception</td>
<td>1.26%</td>
<td>451.17%</td>
</tr>
<tr>
<td>Idle</td>
<td>54.75%</td>
<td>1350</td>
</tr>
<tr>
<td>Total</td>
<td>100.00%</td>
<td>2028.66</td>
</tr>
</tbody>
</table>
```

- 30 μJ transmit (6 byte copies)
- 10 μJ end to end (1.25 byte copies)
How well did we meet the requirements?

- Capable of fine grain concurrency
- Small physical size
- Efficient Resource Utilization
- Highly Modular

Conclusions

- People working on shrinking sensors and communication need to focus on what brings them together
- TinyOS is a highly modular software environment tailored to the requirements of Network Sensors stressing efficiency, modularity and concurrency
- We now have a whole new set of tradeoffs need to investigate

Hardware Kits

- Two Board Sandwich
  - Main CPU board with Radio Communication
  - Secondary Sensor Board
- Allows for expansion and customization
- Current sensors include: Acceleration, Magnetic Field, Temperature, Pressure, Humidity, Light, and RF Signal Strength
- Can control RF transmission strength & Sense Reception Strength

How to get more information:

http://tinyos.millennium.berkeley.edu

Real time operating systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Code Size</th>
<th>Target CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>QNX</td>
<td>286K</td>
<td>Pentium -&gt; Strong ARM</td>
</tr>
<tr>
<td>pOSEK</td>
<td>100K</td>
<td>Pentium II -&gt; ARM Thumb</td>
</tr>
<tr>
<td>VxWorks</td>
<td>100K</td>
<td>Pentium II -&gt; SH4</td>
</tr>
<tr>
<td>QNX RealTime</td>
<td>&gt;100K</td>
<td>Pentium II -&gt; NEC</td>
</tr>
<tr>
<td>OS-9</td>
<td>10K</td>
<td>Pentium -&gt; SH4</td>
</tr>
<tr>
<td>Chorus OS</td>
<td>10K</td>
<td>Pentium -&gt; Strong ARM</td>
</tr>
<tr>
<td>ARIEL</td>
<td>19K</td>
<td>SH2, ARM Thumb</td>
</tr>
<tr>
<td>Creem</td>
<td>560 bytes</td>
<td>ATMEL 8051</td>
</tr>
</tbody>
</table>

- QNX Context switch = 2400 µs on x86
- pOSEK contact switch > 4 µs
- Creem -> no preemption

TOS Component

//AM comp://
TOS_MODULE AM;
ACCEPTS{
  char AM_SEND_MSG(char addr, char type, char* data);
  void AM_POWER(char mode);
  char AM_INIT();
};
SIGNALS{
  char AM_MSG_REC(char type, char* data);
  char AM_MSG_SEND_DONE(char success);
};
HANDLES{
  char AM_TX_PACKET_DONE(char success);
  char AM_RX_PACKET_DONE(char packet);
};
USES{
  char AM_SUB_TX_PACKET(char* data);
  void AM_SUB_POWER(char mode);
  char AM_SUB_INIT();
};
Ad hoc networking

- Each node needs to determine its parent and its depth in the tree
- Each node broadcasts out <identity, depth, data> when parent is known
- At start, Base Station knows it is at depth 0
  - It sends out <Base ID, 0, **>
- Individuals listen for minimum depth parent

Easy Migration of the Hardware Software Boundary

- TinyOS component models hardware abstractions in software
- Component model allows migration of software components into hardware
- Example:
  - Bit level radio processing component could be implemented as specialized FIFO with complex pattern matching
  - Could reduce CPU utilization while sending by more than 50%

Sample tradeoffs

<table>
<thead>
<tr>
<th>Optimization Type</th>
<th>Average Power Consumption (mW)</th>
<th>Battery Lifetime for sensor reporting every minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Low</td>
<td>Full Time Listen: 100% 3 Days</td>
</tr>
<tr>
<td>Micro</td>
<td>Medium</td>
<td>Full Time Low_Power Listen: 100% 6.54 Days</td>
</tr>
<tr>
<td>Macros</td>
<td>Medium</td>
<td>Periodic Multi-Hop Listening: 10% 65 Days</td>
</tr>
<tr>
<td>Both</td>
<td>Medium</td>
<td>No Listen (no Multi-hop): 0.01% Years</td>
</tr>
</tbody>
</table>

Radio Receive Power Optimizations

[Graph showing power consumption and optimization types]