Software for the 100 billion computers of the next decade
After 8 Years

Ph.D.s per network >> 1
We need exceptionally simple primitives.
Three Primitives

- Power locks: automatic power management
- Trickle: eventual consistency
- Grant-to-send: protocol isolation
Three Primitives

- Power locks: automatic power management
- Trickle: eventual consistency
- Grant-to-send: protocol isolation
Common Belief

- Power management requires application knowledge and control
- OS provides explicit power management
  - TinyOS 1.x: StdControl/SplitControl
  - MantiOS: on/off
  - SOS: device-specific functions
- No limits, but also no help
Define three classes of driver power management: virtual, dedicated, shared.

Integrate power management and concurrency control at lowest levels of OS.

Reduces sample application from 400 to 50 lines, 99% efficiency of hand-tuned.
Basic Application

**Every 5 minutes:**
- Turn on SPI bus
- Turn on flash chip
- Turn on voltage reference
- Turn on I2C bus
- Log prior readings
- Start humidity sample
- Wait 5ms for log
- Turn off flash chip
- Turn off SPI bus
- Wait 12ms for vref
- Turn on ADC
- Start total solar sample
- Wait 2ms for total solar
- Start photo active sample
- Wait 2ms for photo active
- Turn off ADC
- Turn off vref
- Wait 34ms for humidity
- Start temperature sample
- Wait 220ms for temperature
- Turn off I2C bus

**Every 12 hours:**
- Turn on SPI bus
- Turn on radio
- Turn on flash chip
- while (new readings):
  - turn on SPI bus
  - send prior reading
  - get next reading
  - wait 5ms for log
  - turn off SPI bus
  - wait for send

**Every 5 minutes:**
- Write prior samples
- Sample photo active
- Sample total solar
- Sample temperature
- Sample humidity

**Every 12 hours:**
- For all new log entries:
  - Send current sample
  - Read next sample
Shared

- Many clients, explicit concurrency (lock)
  - Allow clients to manage atomicity
  - No request buffering/scheduling

- Lock has request information, manages power

- Example: SPI bus

```c
generic configuration Msp430Spi0C() {
    provides interface Resource;
    provides interface SpiByte;
    provides interface SpiPacket;
}
```
• Need a general, simple mechanism for a huge variety of subsystems, devices, and policies
Utility

- Power locks have been applied to the entire TinyOS code base (114 drivers)

ADC on atmega128

USART on msp430
• Virtualized and shared drivers depend on dynamic request streams

• Very limited RAM
  ● Don’t want one subsystem to use up the heap, prevent the radio from acquiring the SPI bus
  ● Or a routing layer from being able to send a packet
  ● Also don’t want to waste energy

• Solution: static functions
The power of counting

- Counting across a program is a very powerful primitive for memory allocation
  - How many packet senders are there?
  - How many timers are used?
  - How many tasks are there?

- Only count across used parts of the system

- Assign each element a unique counter
  - 3 senders: sender 0, sender 1, sender 2
  - 6 timers: timer 0, timer 1, timer 2, ... timer 5
How you do it

- **unique("key")**: each invocation with key returns a unique value starting at zero

- **uniqueCount("key")**: returns the number of values allocated for key

- Evaluated at compile time: constants to underlying C compiler
Example: queue

- AMSenderC is a virtualized packet sender
  - N clients create an N-deep packet queue
  - Each client has a reserved slot
  - Link layer iterates across queue

- Can add a queue on top of virtualization
  - E.g., routing layers
  - Queue imposes per-packet fairness across senders
Dependable counting

- No compile-time magic constants or waste
  - E.g., no “max file descriptors”

- No dynamic failures
  - No cascading failures due to a memory leak
  - Behavior of each instance is independent of all others

- Used all over TinyOS
  - Task queue, packet queue, lock queue, timers, log handles, sensor request queue, etc.
Workload?

- The OS can manage energy effectively
  - Predicated on giving it a helpful workload
- Parallelism through asynchronous I/O is key
  - In traditional systems, improves throughput
  - In low-power sensors, reduces awake time
- Give the OS a batch job, and it can optimize for energy
  - Counting enables dependable behavior
98-99% efficient
## Basic Application

<table>
<thead>
<tr>
<th>Every 5 minutes:</th>
<th>Every 12 hours:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn on SPI bus</td>
<td>Turn on SPI bus</td>
</tr>
<tr>
<td>Turn on flash chip</td>
<td>Turn on radio</td>
</tr>
<tr>
<td>Turn on voltage reference</td>
<td>Turn on flash chip</td>
</tr>
<tr>
<td>Log prior readings</td>
<td>While (new readings):</td>
</tr>
<tr>
<td>Start humidity sample</td>
<td>Turn on SPI bus</td>
</tr>
<tr>
<td>Wait 5ms for log</td>
<td>Turn off SPI bus</td>
</tr>
<tr>
<td>Turn off flash chip</td>
<td>Wait for send</td>
</tr>
<tr>
<td>Turn off SPI bus</td>
<td></td>
</tr>
<tr>
<td>Wait 12ms for vref</td>
<td></td>
</tr>
<tr>
<td>Turn on ADC</td>
<td></td>
</tr>
<tr>
<td>Start total solar sample</td>
<td></td>
</tr>
<tr>
<td>Wait 2ms for total solar</td>
<td></td>
</tr>
<tr>
<td>Start photo active sample</td>
<td></td>
</tr>
<tr>
<td>Wait 2ms for photo active</td>
<td></td>
</tr>
<tr>
<td>Turn off ADC</td>
<td></td>
</tr>
<tr>
<td>Turn off ADC</td>
<td></td>
</tr>
<tr>
<td>Wait 34ms for humidity</td>
<td></td>
</tr>
<tr>
<td>Start temperature sample</td>
<td></td>
</tr>
<tr>
<td>Wait 220ms for temperature</td>
<td></td>
</tr>
<tr>
<td>Turn off I2C bus</td>
<td></td>
</tr>
</tbody>
</table>

A simple primitive can enable the OS to manage power automatically.

<table>
<thead>
<tr>
<th>Every 5 minutes:</th>
<th>Every 12 hours:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write prior samples</td>
<td>For all new log entries:</td>
</tr>
<tr>
<td>Sample photo active</td>
<td>Send current sample</td>
</tr>
<tr>
<td>Sample total solar</td>
<td>Read next sample</td>
</tr>
<tr>
<td>Sample temperature</td>
<td></td>
</tr>
<tr>
<td>Sample humidity</td>
<td></td>
</tr>
</tbody>
</table>
Three Primitives

• Power locks: automatic power management
• Trickle: eventual consistency
• Grant-to-send: protocol isolation
Trickle (NSDI 2004)

- “Every once in a while, transmit if you haven’t heard someone agree recently.”

- Eventual consistency
- Node, time, and space diversity
- log(n) scaling, 5 bytes RAM, 90 lines of code

- Simple, local rule
Once in a While

- Dynamically adjust the interval length $\tau$
  - When every agrees, use a large interval
  - On inconsistency, make it small
  - Grow it exponentially up to a max size

Inconsistency detected
## Trickle Algorithm

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval expires</td>
<td>Double interval up to maximum size.</td>
</tr>
<tr>
<td></td>
<td>Pick new random transmit time.</td>
</tr>
<tr>
<td>Reach transmit time</td>
<td>Transmit if haven’t heard others.</td>
</tr>
<tr>
<td>Hear inconsistency</td>
<td>Shrink interval to minimum size.</td>
</tr>
</tbody>
</table>
Inconsistency?

- Trickle originally intended for installing code or other management data
  - Deluge, Maté, etc.
- Inconsistency was “different version numbers”
- Trickle has much broader applicability
  - Collection tree protocol, Omprakash Gnawali et al.
Wireless (SenSys 2008)

- Wireless links are often bursty
Implications

- In 802.15.4, bursts seem to be ~500ms
- Measuring links with periodic beacons is fundamentally flawed
  - Long-term estimates do not reflect short-term behavior
  - Same observation as Cerpa’s RNP, but time scale is much much smaller
- Link estimation must use the data path
4B Estimator (HotNets 2007)

- Hybrid estimator
  - Every five data packets, generate an estimate
  - Also use periodic beacons (bootstrapping)
4B Improvements

- 29% lower cost, 11% fewer hops
- 55% lower cost, 50% fewer hops
The Problem

- 4 bit estimator can change estimates very fast
  - Protocol actually avoids broken links
- Leads to rapid topology changes (loops!)
- Tradeoff in beacon rate
  - Fast beacons: fast recovery, high cost
  - Slow beacons: slow recovery, low cost
- Insight: use Trickle
Collection Tree Protocol

- Sends control beacons using Trickle algorithm
- Inconsistency when
  - Asked to forward a data packet from a lower ETX
  - ETX jumps significantly
  - Neighbor says it has no route
- As long as the topology is consistent, send few control beacons; when it’s inconsistent, quickly repair.
Results

- Cuts beacons by >75%
- Cuts response time by >99%

8 nodes die
Eventually

- Example use: routing trees
- Control traffic that maintains topology
  - Consistent topology: send rarely
  - Detect inconsistency: send quickly
- Cuts control traffic by 75%
- Recovery time is 20 times faster
- Other uses: dissemination, neighbor discovery, TCP flow timing

A simple primitive can greatly improve performance of many protocols.
Three Primitives

- Power locks: automatic power management
- Trickle: eventual consistency
- Grant-to-send: protocol isolation
Grant to Send

- “Please give the recipient a chance to speak.”
  - Prevents protocol interactions
  - 1 byte header, 3 bytes state, 200 lines of code
- Simple, local rule
- Complex systems out of building blocks
Intra-Flow

Packet flow

A

backoff

packet

B

C

time
Solution

Packet flow

backoff  packet  grant

A

B

C

time
Grant-to-send cuts losses by 95% and doubles throughput
Collision avoidance as good as RTS/CTS

Goodput as good as CSMA
Inter-Flow

- Prevents collisions between protocols

A simple primitive can improve protocol performance, greatly aid management and simplify troubleshooting.
Summary

- Efficiency does not require complexity
- Simple rules make simple systems
  - Power locks: don’t need to worry about power
  - Trickle: can establish consistency
  - Grant-to-send: prevent unforeseen interactions
- Simple systems are easier to design, debug, troubleshoot, and understand
Gaining Visibility

Sensor Readings

Wireless

Output

?
Thanks

- Maria Kazandjieva, Mayank Jain, Jung Il Choi, Jung Woo Lee, Kannan Srinivasan, Tal Rusak, David Gay, Kevin Klues, Omprakash Gnawali, Prabal Dutta, Rodrigo Fonseca, Kyle Jamieson, Arsalan Tavakoli, David Chu, David Culler, Ion Stoica, Will Archer, John Regehr, Vlado Handziski, Kaisen Lin, Andreas Terzis, Ramesh Govindan, Branislav Kusy, and many others...
Questions