CS450
Networked Embedded Sensing Systems
Week 4: Node HW and SW

Andreas Terzis
http://hinrg.cs.jhu.edu/CS450/
MCU Basics: Many flavors of Microcontrollers

- From embedded x86 processors 16 and 32-bit processors all the way down to tiny 4-bit processors
- Some of the popular 8-bit families
  - AVR, 8051, Z80, 6502, PIC, Motorola HC11
- 16-bit families
  - Hitachi, Dragon
- Many embedded Java controllers are also emerging
- Reading Sensors
  - A/D Controllers
  - Registers
Power Consumption

• Need long lifetime with battery operation
  • No infrastructure, high deployment & replenishment costs
• Challenges
  • Energy to wirelessly transport bits is \( \sim \) constant
    • Shannon, Maxwell
  • Fundamental limit on ADC speed*resolution/power
  • No Moore’s law for battery technology
    • \( \sim 5\% \)/year
• How is power consumed
  • CPU
  • Radio
• Power Source
• Mechanisms to conserve power
Energy Consumption in Wireless Sensor Nodes

- Processing
  - excluding low-level processing for radio, sensors, actuators
- Radio
- Sensors
- Actuators
- Power supply
Power Consumption in CMOS Digital logic

\[ P = A \cdot C \cdot V^2 \cdot f + A \cdot I_{sw} \cdot V \cdot f + I_{\text{leak}} \cdot V \]

where

- \( A \) = activity factor (probability of 0 \( \rightarrow \) 1 transition)
- \( C \) = total chip capacitance
- \( V \) = total voltage swing, usually near the power supply voltage
- \( f \) = clock frequency
- \( I_{sw} \) = short circuit current when logic level changes
- \( I_{\text{leak}} \) = leakage current in diodes and transistors
CPU: Approaches to Energy Efficiency

\[ P = \alpha C V^2 f \]

“Continuous”
Only Throughput is Important

Reduce \( V \)
Increase h/w and algorithmic concurrency

“Event-Driven”
Latency is Important
(Burst throughput)

Make \( f \) low or 0
Shutdown when inactive

Reduce \( \alpha C \)
Energy efficient s/w
System partitioning
Efficient Circuits & Layouts
Shutdown for Energy Saving

- Subsystems may have small duty factors
  - Wireless interface is often idle

- Huge difference between “on” & “off” power
  - Some Low-Power CPUs:
    - StrongARM: 400mW (active)/ 50 mW (idle) / 0.16 mW (sleep)

- Matches WSN Applications

\[
\text{ideal improvement} = 1 + \frac{T_{\text{block}}}{T_{\text{active}}}
\]
Energy in Radio

- Wireless communication subsystem consists of three components with substantially different characteristics
- Their relative importance depends on the transmission power of the radio
Examples

- The RF energy increases with transmission range
- The electronics energy for transmit and receive are typically comparable
- Telos $E_{tx} = 129\text{nJ/bit}$, $E_{rx} = 144 \text{ nJ/bit}$
### Computation & Communication

- **Energy/bit ÷ Energy/op large even for short ranges!**

<table>
<thead>
<tr>
<th></th>
<th>Mote-class Node</th>
<th>WINS-class Node</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transmit</strong></td>
<td>720 nJ/bit</td>
<td>6600 nJ/bit</td>
</tr>
<tr>
<td><strong>Receive</strong></td>
<td>110 nJ/bit</td>
<td>3300 nJ/bit</td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>4 nJ/op</td>
<td>1.6 nJ/op</td>
</tr>
<tr>
<td><strong>~ 200 ops/bit</strong></td>
<td></td>
<td><strong>~ 6000 ops/bit</strong></td>
</tr>
</tbody>
</table>

**Energy breakdown for acoustic**

- **Encode**: ...
- **Decode**: ...
- **Receive**: ...
- **Transmit**: ...

**Energy breakdown for image**

- **Encode**: ...
- **Decode**: ...
- **Transmit**: ...
- **Receive**: ...
Other power management features

- Wake on wireless: Bluetooth based remote wakeup
  - BT module awake, rest of the system is shutdown
  - Incoming BT packet causes wakeup
  - On-demand power management (event-driven apps)
  - BT module in “wake on wireless” mode draws ~ 3mA
- Motion detection for wake up
  - Passive small-bead mercury switch connected to GPIO
  - Movement causes switch to close and wakeup system
  - Can also be used to trigger wireless scanning for APs
Source of Power: Batteries

- When in operation the **electrochemical cell** essentially **discharges** its chemical energy in favor of electric energy. If the cell is connected via an external circuit from the cathode to the anode, electrons flow from the oxidized anode and are received by the cathode, which is subsequently reduced. The electric circuit is completed by cations and anions, within the electrolyte, which flow to the cathode and anode, respectively.

![Diagram of Cell in Operation (Discharge)](image-url)
Battery Characteristics

- Important characteristics:
  - Energy density (Wh/liter) and specific energy (Wh/kg)
  - Open-circuit voltage, operating voltage
  - Cut-off voltage (at which considered discharged)
  - Shelf life (leakage)
  - Cycle life (rechargeable)
- The above are decided by “system chemistry”
  - Advances in materials and packaging have resulted in significant changes in older systems
- New systems
  - Primary and secondary (rechargeable) Li
  - Secondary zinc-air, Ni-metal hydride
Modeling the Battery Behavior

- Theoretical capacity of battery is decided by the amount of the active material in the cell
- Batteries often modeled as buckets of constant energy
- In reality, delivered or nominal capacity depends on how the battery is discharged
  - Discharge rate (load current)
  - Discharge profile and duty cycle
  - Operating voltage and power level drained
Battery Capacity

- Current in “C” rating: load current normalized to battery’s capacity
  - e.g. a discharge current of 1C for a capacity of 500 mA-hrs is 500 mA

From [Powers95]
Battery Capacity vs. Discharge Current: Peukert’s Formula

- Energy capacity: $C = k / I^\alpha$
  - $k$ = constant dependent on chemistry & design
  - $\alpha = 0$ for ideal battery (constant capacity), up to 0.7 for most loads in real batteries
    - Also depends on chemistry and design
- Good first order approximation
  - Does not capture effects of discharge profile
- Battery life at constant voltage and current:

  $$L = \frac{C}{P} = \frac{C}{V \cdot I} = \frac{K}{V \cdot I^\alpha} = \frac{K}{V} \cdot I^{-(1+\alpha)}$$
Many ways to Optimize Power Consumption

- Power aware computing
  - Ultra-low power microcontrollers
  - Dynamic power management HW
    - Dynamic voltage scaling (e.g Intel’s PXA, Transmeta’s Crusoe)
    - Components that switch off after some idle time
- Energy aware software
  - Power aware OS: dim displays, sleep on idle times, power aware scheduling
- Power management of radios
  - Sometimes listen overhead larger than transmit overhead
- Energy aware packet forwarding
  - Radio automatically forwards packets at a lower level, while the rest of the node is asleep
- Energy aware wireless communication
  - Exploit performance energy tradeoffs of the communication subsystem, better neighbor coordination, choice of modulation schemes
# Mote Evolution

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>AT90LS8535</td>
<td>ATmega163</td>
<td>ATmega128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program memory (KB)</td>
<td>8</td>
<td>16</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM (KB)</td>
<td>0.5</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active Power (mW)</td>
<td>15</td>
<td>15</td>
<td>8</td>
<td>33</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sleep Power (μW)</td>
<td>45</td>
<td>45</td>
<td>75</td>
<td>75</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wakeup Time (μs)</td>
<td>1000</td>
<td>36</td>
<td>180</td>
<td>180</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonvolatile storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chip</td>
<td>24LC256</td>
<td></td>
<td></td>
<td>AT45DB041B</td>
<td></td>
<td>ST M24M01S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection type</td>
<td>I²C</td>
<td></td>
<td></td>
<td>SPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size (KB)</td>
<td>32</td>
<td></td>
<td></td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio</td>
<td>TR1000</td>
<td>TR1000</td>
<td>CC1000</td>
<td>CC2420</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data rate (kbps)</td>
<td>10</td>
<td>40</td>
<td>38.4</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modulation type</td>
<td>OOK</td>
<td>ASK</td>
<td>FSK</td>
<td>O-QPSK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Power (mW)</td>
<td>9</td>
<td>12</td>
<td>29</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmit Power at 0dBm (mW)</td>
<td>36</td>
<td>36</td>
<td>42</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Consumption</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td>1.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum Operation (V)</td>
<td>24</td>
<td>27</td>
<td>44</td>
<td>89</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Active Power (mW)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming and Sensor Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expansion</td>
<td>none</td>
<td>51-pin</td>
<td>51-pin</td>
<td>none</td>
<td>51-pin</td>
<td>19-pin</td>
<td>51-pin</td>
<td>10-pin</td>
</tr>
<tr>
<td>Communication</td>
<td>IEEE 1284 (programming) and RS232 (requires additional hardware)</td>
<td>USB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrated Sensors</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Telos Platform

• Radio:
  • IEEE 802.15.4
    • CC2420 radio
    • 250kbps
    • 2.4GHz ISM band
• Processor:
  • TI MSP430 (16bit) @8MHz
    • 1.6µA sleep
    • 460µA active
    • 1.8V operation
• Robustness
  • Integrated antenna
  • Integrated sensors
  • Soldered connections
Low Power Operation

- TI MSP430 -- Advantages over previous motes
  - 16-bit core
  - 12-bit ADC
    - 16 conversion store registers
  - Sequence and repeat sequence programmable
  - < 50nA port leakage (vs. 1mA for Atmels)
  - Double buffered data buses
  - Interrupt priorities
  - Calibrated DCO
- Buffers and Transistors
  - Switch on/off each sensor and component subsystem
Minimize Power Consumption

- Compare to MicaZ: a Mica2 mote with AVR mcu and 802.15.4 radio
- Sleep
  - Majority of the time
  - Telos: 2.4mA
  - MicaZ: 30mA
- Wakeup
  - As quickly as possible to process and return to sleep
  - Telos: 290ns typical, 6ms max
  - MicaZ: 60ms max internal oscillator, 4ms external
- Active
  - Get your work done and get back to sleep
  - Telos: 4-8MHz 16-bit
  - MicaZ: 8MHz 8-bit
CC2420 Radio
IEEE 802.15.4 Compliant

- CC2420
  - Fast data rate, robust signal
    - 250kbps : 2Mchip/s : DSSS
    - 2.4GHz : Offset QPSK : 5MHz
    - 16 channels in 802.15.4
    - -94dBm sensitivity
  - Low Voltage Operation
    - 1.8V minimum supply
  - Software Assistance for Low Power Microcontrollers
    - 128byte TX/RX buffers for full packet support
    - Automatic address decoding and automatic acknowledgements
  - Hardware encryption/authentication
  - Link quality indicator (assist software link estimation)
    - samples error rate of first 8 chips of packet (8 chips/bit)
Outline

• Examples of operating systems for networked embedded devices
  • TinyOS (covered by Mike)
  • Mantis
  • Contiki
  • SOS
  • Emstar
MANTIS: System Support For Multimodal NeTworks of In-situ Sensors

Hector Abrach, Shah Bhatti, Jim Carlson, Hui Dai, Jeff Rose, Anmol Sheth, Brian Shucker, Jing Deng, Richard Han

University of Colorado at Boulder
MANTIS Goals

- Offer a general-purpose software/hardware platform that
  - Simplifies sensor networking for newcomers
  - Preserves flexibility for advanced research
  - Adapts to resource constraints
- Targets for streamlining
  - Software API and OS
  - Hardware Accessibility
C & Threads: the MANTIS Sensor OS

- Simple C API
  - Easy to program
  - For advanced users:
    - Reuse code
    - Cross-platform Portability
- Simple multithreading
  - Familiar UNIX-like semantics
  - For advanced users:
    - flexible time-sliced scheduling for complex app’s

```c
void adc_send()
{
    uint8_t value;
    while(1)
    {
        value = adc_read_8bit(ADC_CH_2);
        mos_send_to(NODE2, PORT1, &value, 1, FLOODING);
    }
}
```
Multithreaded MANTIS OS (MOS)
Lightweight MOS Kernel

- Multi-threaded
  - Priority/round-robin scheduling
  - Fast context switching
- Counting and binary semaphores
- 144 byte static RAM footprint
  - 10-byte thread table entries
  - Thread context saved on stack
  - Stack space allocated dynamically
Cross-Platform MANTIS OS

Sensor Node

- Network Stack
- Command Server
- Kernel/Scheduler
- Device Drivers
- Sensor Hardware

X86 PC

- Network Stack
- Command Server
- POSIX Shim Layer
- UNIX
- X86 Hardware

Fall 2008 “AMOS”

“XMOS”

CS450
Cross-Platform User-level Network Stack

- Non-strict layered design
- Easy to modify/experiment
- Chained headers eliminate inter-layer copies
  - Zero-copy on send, forward
  - Zero- or single-copy on delivery to application
Integrating Applications Into the Sensor Network Is Easy

Application, e.g.
- Visualization app
- Bridging Gateway

Diagram:
- Socket API
- Database API
- X Windows GUI
- MANTIS System API
- POSIX Shim Layer
- UNIX
- X86 Hardware
Dynamic Reprogramming

- Reprogram entire node while deployed
  - MOS boot loader can re-flash entire OS
  - Load stored code image from EEPROM
- Source-independent reprogramming
  - Standard API to store code image to EEPROM
  - Reprogramming possible over arbitrary connection (multi-hop), or from application
  - Simple, flexible network management
Remote Shell/Command Server

- Remote "login" to nodes
- Debugging functions
  - Peek/poke
  - Kernel status info
- Configuration
  - Spawn threads
  - Call functions
  - Reprogram node
Hardware Overview

• Same core as MICA 2
  • CC1000 multi-channel radio
  • ATMEGA 128 microcontroller
  • Possible to install MOS on MICA 2 (minor port)
• Single board design
  • Standard 3-wire sensor interface on-board
  • On-board serial doubles as programming interface
• 20-pin expansion connector
• GPS port
• Multiple power options
# Power Consumption

<table>
<thead>
<tr>
<th>Test</th>
<th>Input Voltage (V)</th>
<th>Load Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Just Operating System Running</td>
<td>3.0</td>
<td>15</td>
</tr>
<tr>
<td>Scheduled Single Thread While(1)</td>
<td>3.0</td>
<td>16</td>
</tr>
<tr>
<td>Scheduler with blinking LED ON</td>
<td>3.0</td>
<td>24</td>
</tr>
<tr>
<td>Scheduler with blinking LED OFF</td>
<td>3.0</td>
<td>16</td>
</tr>
<tr>
<td>Reading Sensor Data on while(1)</td>
<td>3.0</td>
<td>16</td>
</tr>
<tr>
<td>Sensing and Sending over Serial at 19.2kBaud</td>
<td>3.0</td>
<td>16</td>
</tr>
<tr>
<td>Sensing and Sending over radio, Transmitting at max power</td>
<td>3.0</td>
<td>75</td>
</tr>
<tr>
<td>Sensing and Sending over radio, Transmitting at min power</td>
<td>3.0</td>
<td>42</td>
</tr>
<tr>
<td>Sensing and Sending over radio, Receiving at max power</td>
<td>3.0</td>
<td>42</td>
</tr>
<tr>
<td>Sensing and Sending over radio, Receiving at min power</td>
<td>3.0</td>
<td>29</td>
</tr>
<tr>
<td>Absolute Max power with all LEDs on and radio transmit at max power</td>
<td>3.0</td>
<td>95</td>
</tr>
<tr>
<td>Absolute Max power with all LEDs on and radio receive at max power</td>
<td>3.0</td>
<td>62</td>
</tr>
<tr>
<td>Single LED power consuption</td>
<td>3.0</td>
<td>9</td>
</tr>
<tr>
<td>Everything in sleep mode</td>
<td>3.0</td>
<td>under 1</td>
</tr>
<tr>
<td>Ellite Nymph - Operating System and GPS</td>
<td>3.0</td>
<td>101</td>
</tr>
<tr>
<td>Ellite Nymph - Everything Running at maximum power including LEDs</td>
<td>3.0</td>
<td>174</td>
</tr>
</tbody>
</table>
SOS - Dynamic operating system for sensor networks

Simon Han, Ram Kumar, Roy Shea, Eddie Kohler and Mani Srivastava
Motivation: Re-tasking sensor networks

Re-tasking a deployed network

Data Gathering

Bird Localization

Fire Emergency

Requires in-situ re-programming
Re-programming Challenges

• Severe resource constraints on nodes
  • 4 KB RAM, 128 KB FLASH, 2 AA batteries

• Avoiding crashes
  • Unattended operation - Crashed node is useless
  • No architecture support for protection e.g. MMU

• Balancing flexible and concise updates
  • Update applications, services and drivers
  • Energy efficient distribution and storage
Towards general purpose sensor OS

- TinyOS and Maté
  - Application and OS are tightly linked
- Design Goal: An application independent sensor OS
  - Independently written & deployed apps run on one network
  - Towards traditional kernel space/user space programming model
- Re-programming via binary modules
  - Risk: Lose safety provided by static analysis or dynamic interpreter
- Design Challenge
  - Provide general purpose OS semantics on resource constrained embedded sensor nodes
Architecture Overview

Static SOS Kernel

- Dynamic Memory
- Message Scheduler
- Dynamic Linker
- Sensor Manager
- Messaging I/O
- System Timer
- Radio*
- I2C
- ADC*

Dynamically Loaded modules

Kernel Components

SOS Services

Device Drivers

* - Drivers adapted from TinyOS for Mica2
SOS Overview

- Programmed entirely in C
- Co-operatively scheduled system
- Event-driven programming model
- System provides no memory protection
Designing Safety Features

- Dynamically evolving system
  - Unspecified behavior resulting from transient states

- Goals
  - Ensure system integrity
  - Graceful recovery from failures

- Design
  - Minimal set of run-time checks
  - Designed for low resource utilization
  - Does not cover all failure modes
Installing Dynamic Modules

• Modules implement specific function or task
• Position independent binary
• Loader stores module at arbitrary program memory location
• Minimal state maintenance
  • 8 bytes per module
  • Stores module identity and version

FLASH Layout

<table>
<thead>
<tr>
<th>SOS Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Empty Space&gt;</td>
</tr>
<tr>
<td>Module 1</td>
</tr>
<tr>
<td>&lt;Empty Space&gt;</td>
</tr>
<tr>
<td>Bootloader</td>
</tr>
</tbody>
</table>
Inter-module Communication

- Dynamic Linking
  - Synchronous communication
  - Blocking function calls that return promptly

- Message Passing
  - Asynchronous communication
  - Long running operations
Dynamic Linking Overview

• Goals
  • Low latency inter-module communication comparable to direct function calls
  • Functional interface is convenient to program

• Challenges
  • Safety features to address missing and updated modules

• Constraints
  • Minimize RAM usage
Dynamic Linking Design

- **Publish** functions for the other parts of the system to use
- **Subscribe** to functions supplied by other modules
- Indirection provides support for safety features
- Dynamic function call overhead
  - 21 cycles compared to 4 cycles for direct function call

![Diagram](image)
Message Passing System

Data Collector Application → Inter-module communication → Tree Routing Module

Kernel - module communication

System Timer

System Scheduler

MESSAGE
<Dest. Addr>
<Dest. Mod. Id>
<Message Type>
<Payload> …

- Scheduler looks up handler of destination module
- Handler performs long operations on message payload
Module-Kernel Communication

- Kernel services available as system calls
- Jump table redirects system calls to handlers
- Update kernel independent of modules
- System Call Overhead - 12 clock cycles
Evaluation

• Design Goal
  • Provide general purpose OS semantics
  • Low resource utilization
• Hypothesis
  • Performance no worse TinyOS
  • Update cost closer to Maté
• Experiment Setup
  • Surge data collection and tree routing on 3 hop network
  • Low duty cycle application
  • Mica2 motes: AVR 8-bit microcontroller
Application Performance Comparison

- Application performance is nearly identical for TinyOS, SOS and Mate

Data Transfer Delay

Packet Delivery Ratio
Performance Overhead

Active Time (%)
Average Power(mW)

- CPU Active Time - Metric to measure OS overhead
  - Measured by profiling Surge for 1 min. on real nodes
  - Averaged over 20 experiments for each system
- SOS has 1% overhead relative to TinyOS
  - Surge has minimal application level processing ("worst" case OS overhead)
- Insignificant variation of average power consumption
  - Surge application has a very low CPU utilization
  - System level energy: $E(\text{CPU}) < < E(\text{Radio})$
  - Duty Cycling - Idle energy dominates over active energy

<table>
<thead>
<tr>
<th></th>
<th>TinyOS</th>
<th>SOS</th>
<th>Maté</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Time (%)</td>
<td>4.58%</td>
<td>4.64%</td>
<td>5.13%</td>
</tr>
<tr>
<td>Average Power(mW)</td>
<td>29.92</td>
<td>29.94</td>
<td>30.02</td>
</tr>
</tbody>
</table>
Update Costs

<table>
<thead>
<tr>
<th>Method</th>
<th>Energy</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entire binary upgrade (TinyOS)</td>
<td>784.14 mJ</td>
<td>High</td>
</tr>
<tr>
<td>Modular binary upgrade (SOS)</td>
<td>12.25 mJ</td>
<td>Moderate</td>
</tr>
<tr>
<td>Virtual Machine scripts (Maté)</td>
<td>0.34 mJ</td>
<td>Low</td>
</tr>
</tbody>
</table>

- Re-programming cost involves
  - Communication Energy - Transfer the new code
  - Storage Energy - Write the code to RAM/FLASH etc.

- Impact on system level energy
  - Depends significantly upon frequency of updates
  - Difference in update cost amortized over the interval between updates
  - Idle energy in the interval between updates dominates
  - Idle energy consumption does not depend on the OS