Energy-aware Reconfiguration of Sensor Nodes

Andreas Weissel
Simon Kellner

Department of Computer Sciences 4
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

{weissel,kellner}@cs.fau.de
Sensor Nodes

- Run-time of several years

- Rely on battery power
  - recharge or exchange of power source often not possible

- BTnode (ETH Zürich)
  - Atmel ATmega128 (8 MHz)
  - 128 KB flash memory
  - 4 KB RAM
  - low-power radio ChipCon CC1000, Bluetooth
  - 2 AA batteries (each 1500 mAh)
Outline

- Approaches to Energy Management

- Energy-aware Sensor Nodes
  - Run-time Energy Accounting
  - Battery Lifetime Estimation

- Energy Management Through Reconfiguration
  - Application Upload
  - Multiple Pre-installed Applications

- Conclusion
Approaches to Energy Management

- Estimate & control energy before deployment
  - detailed energy models
  - simulation sufficient for estimation of run-time (e.g., PowerTossim for Mica2 motes, AEON)

- `sleepPeriod` determines energy consumption and run-time

```python
loop
  val ← readSensor()
  sendPacket(val)
  sleep(sleepPeriod)
end loop
```
Nondeterministic Applications

Energy consumption?
- worst/average-case estimation
- redundancy
- “trial-and-error” (deploy, test, collect, re-program)

Alternative: adaptive, on-line power management

```
loop
    val ← readSensor()
    if val ≥ threshold then
        sendPacket(val)
    end if
    sleep(sleepPeriod)
end loop
```
Energy-Aware Sensor Nodes

- Adaptive power management requires information on
  - energy consumption
  - remaining power supply

- Run-time energy estimation
  - hardware with deterministic behavior (no caches ...)
  - fine-grained control over operating modes
  - power consumption of components & modes well documented
## Energy Accounting: Microcontroller

### ATmega 128L

<table>
<thead>
<tr>
<th>instruction</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>lds</td>
<td>31.0 mW</td>
</tr>
<tr>
<td>nop</td>
<td>29.7 mW</td>
</tr>
<tr>
<td>xor</td>
<td>29.5 mW</td>
</tr>
<tr>
<td>fmul</td>
<td>29.0 mW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mode</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>14.28 mW</td>
</tr>
<tr>
<td>power save</td>
<td>0.79 µW</td>
</tr>
<tr>
<td>power down</td>
<td>0.71 µW</td>
</tr>
</tbody>
</table>

- active mode: count number of cycles
- sleep modes: measure time with externally clocked timer
Energy Accounting: Microcontroller

**ATmega 128L**

- active mode: count number of cycles
- sleep modes: measure time with externally clocked timer

### Implementation (TinyOS)
- modified scheduler
- code size: 470–988 bytes
- run time overhead < 5 %

<table>
<thead>
<tr>
<th>instruction</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>lds</td>
<td>31.0 mW</td>
</tr>
<tr>
<td>nop</td>
<td>29.7 mW</td>
</tr>
<tr>
<td>xor</td>
<td>29.5 mW</td>
</tr>
<tr>
<td>fmul</td>
<td>29.0 mW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mode</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>14.28 mW</td>
</tr>
<tr>
<td>power save</td>
<td>0.79 µW</td>
</tr>
<tr>
<td>power down</td>
<td>0.71 µW</td>
</tr>
</tbody>
</table>
Energy Accounting: Communication

- ChipCon radio
  
<table>
<thead>
<tr>
<th>mode</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive</td>
<td>28.8 mW</td>
</tr>
<tr>
<td>send</td>
<td>26.4–80.7 mW</td>
</tr>
</tbody>
</table>

  - measure time the radio is turned on (receiving/sending)
  - or: count number of packets sent

- Bluetooth
  
  - less suited for energy accounting as Bluetooth chip hides low-level functionality
  - several low-power idle modes (hold, sniff, park)
Estimating Battery Lifetime

- A/D converter to read battery voltage
Estimating Battery Lifetime

- First approach: generic battery model
  - but: hard to derive

- Second approach: regression fitting
  
  \[ f(x) = a_6 e^{a_5 x} + a_4 e^{a_3 x} + a_2 x^2 + a_1 x + a_0 \]

  - but: erratic readings of A/D converter on BTnode

- Third approach:
  store tables describing discharge characteristics on node
  - several discharge curves for different temperature ranges
Outline

- Approaches to Energy Management
- Energy-aware Sensor Nodes
  - Run-time Energy Accounting
  - Battery Lifetime Estimation
- Energy Management Through Reconfiguration
  - Application Upload
  - Multiple Pre-installed Applications
- Conclusion
Reconfiguration: Application Upload

- Update nodes via communication link

- Replace (parts of) application
  - exchange complete binary ($X_{np}$)
  - or single modules/components (Contiki, SOS, Maté)
  - overhead due to interaction between dynamically loaded modules
    (6% for SOS, 11% for Maté)

- Example
  - install new application (16.5 kB)
  - overhead of transmission:
    Bluetooth: 18.9 s, 2 J, ChipCon: 13.8 s, 1.6 J
  - overhead of programming: 1.3 s, 100 mJ
Multiple Pre-installed Applications

- Interrupt table and initialization code in first page of program memory (flash)
- Implementation
  - overwrite this page and reset BTnode
  - overhead: 20 ms, 1–2 mJ
  - code size: 1274 byte + flash page (256 bytes)
Dispatcher

- Avoid writing to flash memory
- Implementation
  - interrupt dispatcher (id of current application stored in RAM)
  - code size: 334 byte (including flash page for dispatcher)
  - run-time overhead at each interrupt due to additional level of indirection (56 cycles)
Conclusion

- Transfer concepts of adaptive power management to sensor nodes
  - extremely limited resources
  - avoid additional energy overhead

- Make sensor nodes energy-aware
  - on-line energy accounting
  - battery lifetime estimation

- Control energy consumption during runtime
  - reconfiguration through application upload
  - multiple pre-installed applications