Wireless Sensor Networks
6th Lecture
14.11.2006

Christian Schindelhauer
schindel@informatik.uni-freiburg.de

University of Freiburg
Computer Networks and Telematics
Prof. Christian Schindelhauer
Transceiver characteristics

- **Capabilities**
  - Interface: bit, byte, packet level?
  - Supported frequency range?
    - Typically, somewhere in 433 MHz – 2.4 GHz, ISM band
  - Multiple channels?
  - Data rates?
  - Range?

- **Energy characteristics**
  - Power consumption to send/receive data?
  - Time and energy consumption to change between different states?
  - Transmission power control?
  - Power efficiency (which percentage of consumed power is radiated?)

- **Radio performance**
  - Modulation? (ASK, FSK, …?)
  - Noise figure? \( NF = \frac{SNR_I}{SNR_O} \)
    - output noise added
  - Gain? (signal amplification)
  - Receiver sensitivity? (minimum \( S \) to achieve a given \( E_b/N_0 \))
  - Blocking performance (achieved BER in presence of frequency-offset interferer)
  - Out of band emissions
  - Carrier sensing & RSSI characteristics
    - Received Signal Strength Indication
  - Frequency stability (e.g., towards temperature changes)
  - Voltage range
Transceiver states

- Transceivers can be put into different operational states, typically:
  - Transmit
  - Receive
  - Idle – ready to receive, but not doing so
    - Some functions in hardware can be switched off, reducing energy consumption a little
  - Sleep – significant parts of the transceiver are switched off
    - Not able to immediately receive something
    - Recovery time and startup energy to leave sleep state can be significant

- Research issue: Wakeup receivers – can be woken via radio when in sleep state (seeming contradiction!)
Example radio transceivers

- Almost boundless variety available
- Some examples
  - RFM TR1000 family
    - 916 or 868 MHz
    - 400 kHz bandwidth
    - Up to 115.2 kbps
    - On/off keying or ASK
    - Dynamically tuneable output power
    - Maximum power about 1.4 mW
    - Low power consumption
  - Chipcon CC1000
    - Range 300 to 1000 MHz, programmable in 250 Hz steps
    - FSK modulation
    - Provides RSSI
  - Chipcon CC 2400
    - Implements 802.15.4
    - 2.4 GHz, DSSS modem
    - 250 kbps
    - Higher power consumption than above transceivers
  - Infineon TDA 525x family
    - E.g., 5250: 868 MHz
    - ASK or FSK modulation
    - RSSI, highly efficient power amplifier
    - Intelligent power down, “self-polling” mechanism
    - Excellent blocking performance
Wakeup receivers

➤ **Major energy problem:** RECEIVING
   - Idling and being ready to receive consumes considerable amounts of power

➤ **When to switch on a receiver is not clear**
   - Contention-based MAC protocols: Receiver is always on
   - TDMA-based MAC protocols: Synchronization overhead, inflexible

➤ **Desirable: Receiver that can (only) check for incoming messages**
   - When signal detected, wake up main receiver for actual reception
   - Ideally: *Wakeup receiver* can already process simple addresses
   - Not clear whether they can be actually built, however
Optical communication

- Optical communication can consume less energy
- Example: passive readout via corner cube reflector
  - Laser is reflected back directly to source if mirrors are at right angles
  - Mirrors can be “tilted” to stop reflecting
  - Allows data to be sent back to laser source
Ultra-wideband communication

- **Standard radio transceivers**: Modulate a signal onto a carrier wave
  - Requires relatively small amount of bandwidth

- **Alternative approach**: Use a large bandwidth, do not modulate, simply emit a “burst” of power
  - Forms almost rectangular pulses
  - Pulses are very short
  - Information is encoded in the presence/absence of pulses
  - Requires tight time synchronization of receiver
  - Relatively short range (typically)

- **Advantages**
  - Pretty resilient to multi-path propagation
  - Very good ranging capabilities
  - Good wall penetration
Sensors as such

- **Main categories**
  - Any energy radiated? Passive vs. active sensors
  - Sense of direction? Omnidirectional?
    - Passive, omnidirectional
      - Examples: light, thermometer, microphones, hygrometer, ...
    - Passive, narrow-beam
      - Example: Camera
    - Active sensors
      - Example: Radar

- **Important parameter: Area of coverage**
  - Which region is adequately covered by a given sensor?
Outline

➢ Sensor node architecture
➢ Energy supply and consumption
➢ Runtime environments for sensor nodes
➢ Case study: TinyOS
Energy supply of mobile/sensor nodes

- Goal: provide as much energy as possible at smallest cost/volume/weight/recharge time/longevity
  - In WSN, recharging may or may not be an option

- Options
  - Primary batteries – not rechargeable
  - Secondary batteries – rechargeable, only makes sense in combination with some form of energy harvesting

- Requirements include
  - Low self-discharge
  - Long shelf live
  - Capacity under load
  - Efficient recharging at low current
  - Good relaxation properties (seeming self-recharging)
  - Voltage stability (to avoid DC-DC conversion)
## Battery examples

**Energy per volume (Joule per cubic centimeter):**

<table>
<thead>
<tr>
<th></th>
<th>Primary batteries</th>
<th>Secondary batteries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chemistry</strong></td>
<td>Zinc-air</td>
<td>Lithium</td>
</tr>
<tr>
<td><strong>Energy (J/cm³)</strong></td>
<td>3780</td>
<td>2880</td>
</tr>
<tr>
<td><strong>Chemistry</strong></td>
<td>Lithium</td>
<td>NiMHd</td>
</tr>
<tr>
<td><strong>Energy (J/cm³)</strong></td>
<td>1080</td>
<td>860</td>
</tr>
</tbody>
</table>
Energy scavenging

➢ How to recharge a battery?
  - A laptop: easy, plug into wall socket in the evening
  - A sensor node? – Try to **scavenge** energy from environment

➢ Ambient energy sources
  - Light → solar cells – between 10 µW/cm² and 15 mW/cm²
  - Temperature gradients – 80 µ W/cm² @ 1 V from 5K difference
  - Vibrations – between 0.1 and 10000 µ W/cm³
  - Pressure variation (piezo-electric) – 330 µ W/cm² from the heel of a shoe
  - Air/liquid flow
    (MEMS gas turbines)
### Energy scavenging – overview

<table>
<thead>
<tr>
<th>Energy source</th>
<th>Energy density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batteries (zinc-air)</td>
<td>1050 – 1560 mWh/cm³</td>
</tr>
<tr>
<td>Batteries (rechargable lithium)</td>
<td>300 mWh/cm³ (at 3 – 4 V)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy source</th>
<th>Power density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solar (outdoors)</td>
<td>15 mW/cm² (direct sun)</td>
</tr>
<tr>
<td></td>
<td>0.15 mW/cm² (cloudy day)</td>
</tr>
<tr>
<td>Solar (indoors)</td>
<td>0.006 mW/cm² (standard office desk)</td>
</tr>
<tr>
<td></td>
<td>0.57 mW/cm² (&lt; 60 W desk lamp)</td>
</tr>
<tr>
<td>Vibrations</td>
<td>0.01 – 0.1 mW/cm³</td>
</tr>
<tr>
<td>Acoustic noise</td>
<td>3 · 10⁻⁶ mW/cm² at 75 Db</td>
</tr>
<tr>
<td></td>
<td>9.6 · 10⁻⁴ mW/cm² at 100 Db</td>
</tr>
<tr>
<td>Passive human-powered systems</td>
<td>1.8 mW (shoe inserts)</td>
</tr>
<tr>
<td>Nuclear reaction</td>
<td>80 mW/cm³, 10⁶ mWh/cm³</td>
</tr>
</tbody>
</table>
Energy consumption

➤ A “back of the envelope” estimation

➤ Number of instructions
  – Energy per instruction: 1 nJ
  – Small battery (“smart dust”): 1 J = 1 Ws
  – Corresponds: $10^9$ instructions!

➤ Lifetime
  – Or: Require a single day operational lifetime = $24 \times 60 \times 60$ s = 86400 s
  – $1 \text{ Ws} / 86400\text{s} = 11.5 \mu\text{W}$ as max. sustained power consumption!

➤ Not feasible!
Multiple power consumption modes

- **Way out:** Do not run sensor node at full operation all the time
  - If nothing to do, switch to *power safe mode*
  - Question: When to throttle down? How to wake up again?

- **Typical modes**
  - Controller: Active, idle, sleep
  - Radio mode: Turn on/off transmitter/receiver, both

- **Multiple modes possible, “deeper” sleep modes**
  - Strongly depends on hardware
  - TI MSP 430, e.g.: four different sleep modes
  - Atmel ATMega: six different modes
Some energy consumption figures

- **Microcontroller**
  - TI MSP 430 (@ 1 MHz, 3V):
    - Fully operation 1.2 mW
    - Deepest sleep mode 0.3 μW – only woken up by external interrupts (not even timer is running any more)
  - Atmel ATMega
    - Operational mode: 15 mW active, 6 mW idle
    - Sleep mode: 75 μW
Alternative: Dynamic voltage scaling

- Switching modes complicated by uncertainty how long a sleep time is available
- Alternative: Low supply voltage & clock
  - Dynamic voltage scaling (DVS)
- Rationale:
  - Power consumption $P$ depends on
    - Clock frequency
    - Square of supply voltage
    - $P \propto f V^2$
  - Lower clock allows lower supply voltage
  - Easy to switch to higher clock
  - But: execution takes longer
Memory power consumption

- Crucial part: FLASH memory
  - Power for RAM almost negligible

- FLASH writing/erasing is expensive
  - Example: FLASH on Mica motes
  - Reading: 1.1 nAh per byte
  - Writing: 83.3 nAh per byte
Transmitter power/energy consumption for n bits

- Amplifier power: \( P_{\text{amp}} = \alpha_{\text{amp}} + \beta_{\text{amp}} P_{\text{tx}} \)
  - \( P_{\text{tx}} \) radiated power
  - \( \alpha_{\text{amp}}, \beta_{\text{amp}} \) constants depending on model
  - Highest efficiency (\( \eta = P_{\text{tx}} / P_{\text{amp}} \)) at maximum output power

- In addition: transmitter electronics needs power \( P_{\text{txElec}} \)

- Time to transmit n bits: \( n / (R \cdot R_{\text{code}}) \)
  - \( R \) nominal data rate, \( R_{\text{code}} \) coding rate

- To leave sleep mode
  - Time \( T_{\text{start}} \), average power \( P_{\text{start}} \)

\[
E_{\text{tx}} = T_{\text{start}} P_{\text{start}} + n / (R \cdot R_{\text{code}}) (P_{\text{txElec}} + \alpha_{\text{amp}} + \beta_{\text{amp}} P_{\text{tx}})
\]

- Simplification: Modulation not considered
Receiver power/energy consumption for n bits

- Receiver also has startup costs
  - Time $T_{\text{start}}$, average power $P_{\text{start}}$
- Time for n bits is the same $n / (R \cdot R_{\text{code}})$
- Receiver electronics needs $P_{\text{rxElec}}$
- Plus: energy to decode n bits $E_{\text{decBits}}$

$$E_{\text{rx}} = T_{\text{start}} P_{\text{start}} + n / (R \cdot R_{\text{code}}) P_{\text{rxElec}} + E_{\text{decBits}}(R)$$
## Some transceiver numbers

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>( \mu \text{AMPS-1} ) [559]</th>
<th>WINS [670]</th>
<th>MEDUSA-II [670]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha_{\text{amp}} )</td>
<td>Eq. (2.4)</td>
<td>174 mW</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>( \beta_{\text{amp}} )</td>
<td>Eq. (2.4)</td>
<td>5.0</td>
<td>8.9</td>
<td>7.43</td>
</tr>
<tr>
<td>( P_{\text{amp}} )</td>
<td>Amplifier pwr.</td>
<td>179 – 674 mW</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>( P_{\text{rxElec}} )</td>
<td>Reception pwr.</td>
<td>279 mW</td>
<td>368.3 mW</td>
<td>12.48 mW</td>
</tr>
<tr>
<td>( P_{\text{rxIdle}} )</td>
<td>Receive idle</td>
<td>N/A</td>
<td>344.2 mW</td>
<td>12.34 mW</td>
</tr>
<tr>
<td>( P_{\text{start}} )</td>
<td>Startup pwr.</td>
<td>58.7 mW</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>( P_{\text{txElec}} )</td>
<td>Transmit pwr.</td>
<td>151 mW</td>
<td>( \approx ) 386 mW</td>
<td>11.61 mW</td>
</tr>
<tr>
<td>( R )</td>
<td>Transmission rate</td>
<td>1 Mbps</td>
<td>100 kbps</td>
<td>OOK 30 kbps</td>
</tr>
<tr>
<td>( T_{\text{start}} )</td>
<td>Startup time</td>
<td>466 ( \mu \text{s} )</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Controlling transceivers

- Similar to controller, low duty cycle is necessary
  - Easy to do for transmitter – similar problem to controller: when is it worthwhile to switch off
  - Difficult for receiver: Not only time when to wake up not known, it also depends on remote partners
  → Dependence between MAC protocols and power consumption is strong!

- Only limited applicability of techniques analogue to DVS
  - Dynamic Modulation Scaling (DSM): Switch to modulation best suited to communication – depends on channel gain
  - Dynamic Coding Scaling – vary coding rate according to channel gain
  - Combinations
Computation vs. communication energy cost

➢ Tradeoff?
  – Directly comparing computation/communication energy cost not possible
  – But: put them into perspective!
  – Energy ratio of “sending one bit” vs. “computing one instruction”: Anything between 220 and 2900 in the literature
  – To communicate (send & receive) one kilobyte
    = computing three million instructions!

➢ Hence: try to compute instead of communicate whenever possible

➢ Key technique in WSN – *in-network processing!*
  – Exploit compression schemes, intelligent coding schemes, …
Outline

- Sensor node architecture
- Energy supply and consumption
- *Runtime environments for sensor nodes*
- Case study: TinyOS
Operating system challenges in WSN

- Usual operating system goals
  - Make access to device resources abstract (virtualization)
  - Protect resources from concurrent access

- Usual means
  - Protected operation modes of the CPU – hardware access only in these modes
  - Process with separate address spaces
  - Support by a memory management unit

- Problem: These are not available in microcontrollers
  - No separate protection modes, no memory management unit
  - Would make devices more expensive, more power-hungry

→ ???
Operating system challenges in WSN

Possible options

- Try to implement “as close to an operating system” on WSN nodes
  - In particular, try to provide a known programming interface
  - Namely: support for processes!
  - Sacrifice protection of different processes from each other
    → Possible, but relatively high overhead
- Do (more or less) away with operating system
  - After all, there is only a single “application” running on a WSN node
  - No need to protect malicious software parts from each other
  - Direct hardware control by application might improve efficiency

Currently popular verdict: no OS, just a simple run-time environment

- Enough to abstract away hardware access details
- Biggest impact: Unusual programming model
Main issue: How to support concurrency

- Simplest option: No concurrency, sequential processing of tasks
  - Not satisfactory: Risk of missing data (e.g., from transceiver) when processing data, etc.
  - Interrupts/asynchronous operation has to be supported

- Why concurrency is needed
  - Sensor node’s CPU has to service the radio modem, the actual sensors, perform computation for application, execute communication protocol software, etc.
Traditional concurrency: Processes

- Traditional OS: processes/threads
  - Based on interrupts, context switching
  - But: not available – memory overhead, execution overhead

- But: concurrency mismatch
  - One process per protocol entails too many context switches
  - Many tasks in WSN small with respect to context switching overhead

- And: protection between processes not needed in WSN
  - Only one application anyway
Event-based concurrency

- **Alternative:** Switch to event-based programming model
  - Perform regular processing or be idle
  - React to events when they happen immediately
  - Basically: interrupt handler

- **Problem:** must not remain in interrupt handler too long
  - Danger of loosing events
  - Only save data, post information that event has happened, then return
    - *Run-to-completion* principle
  - Two contexts: one for handlers, one for regular execution
Components instead of processes

需创建抽象来分组功能
- 替换“过程”用于此目的
- 例如：网络协议的个别功能

一个选项： Components
- 这里：意指TinyOS
- 通常仅执行单一，定义明确的功能
- 主要与过程的差异:
  - 组件没有执行
  - 组件访问相同的地址空间，没有彼此保护
- 不要与组件编程混淆！
API to an event-based protocol stack

- **Usual networking API: sockets**
  - Issue: blocking calls to receive data
  - Ill-matched to event-based OS
  - Also: networking semantics in WSNs not necessarily well matched to/by socket semantics

- **API is therefore also event-based**
  - E.g.: Tell some component that some other component wants to be informed if and when data has arrived
  - Component will be posted an event once this condition is met
  - Details: see TinyOS example discussion below
Dynamic power management

- Exploiting multiple operation modes is promising
- Question: When to switch in power-safe mode?
  - Problem: Time & energy overhead associated with wakeup; greedy sleeping is not beneficial (see exercise)
  - Scheduling approach
- Question: How to control dynamic voltage scaling?
  - More aggressive; stepping up voltage/frequency is easier
  - Deadlines usually bound the required speed from below
- Or: Trading off fidelity vs. energy consumption!
  - If more energy is available, compute more accurate results
  - Example: Polynomial approximation
    - Start from high or low exponents depending where the polynomial is to be evaluated
Outline

- Sensor node architecture
- Energy supply and consumption
- Runtime environments for sensor nodes
- Case study: TinyOS
Case study embedded OS: TinyOS & nesC

- TinyOS developed by UC Berkely as runtime environment for their “motes”
- nesC as adjunct “programming language”
- **Goal: Small memory footprint**
  - Sacrifices made e.g. in ease of use, portability
  - Portability somewhat improved in newer version
- **Most important design aspects**
  - Component-based system
  - Components interact by exchanging asynchronous events
  - Components form a program by *wiring* them together (akin to VHDL – hardware description language)
TinyOS components

- Components
  - Frame – state information
  - Tasks – normal execution program
  - Command handlers
  - Event handlers
- Handlers
  - Must run to completion
  - Form a component’s interface
  - Understand and emits commands & events
- Hierarchically arranged
  - Events pass upward from hardware to higher-level components
  - Commands are passed downward

Diagram:
- TimerComponent
  - setRate
  - fire
  - init
  - start
  - stop
  - fired
  - Command handlers
  - Frame
  - Task
  - Event handlers
Handlers versus tasks

- Command handlers and events must run to completion
  - Must not wait an indeterminate amount of time
  - Only a request to perform some action
- Tasks, on the other hand, can perform arbitrary, long computation
  - Also have to be run to completion since no non-cooperative multi-tasking is implemented
  - But can be interrupted by handlers
    → No need for stack management, tasks are atomic with respect to each other
Split-phase programming

- **Handler/task characteristics and separation has consequences on programming model**
  - How to implement a blocking call to another component?
  - Example: Order another component to send a packet
  - Blocking function calls are not an option

- **Split-phase programming**
  - First phase: Issue the command to another component
    - Receiving command handler will only receive the command, post it to a task for actual execution and returns immediately
    - Returning from a command invocation does not mean that the command has been executed!
  - Second phase: Invoked component notifies invoker by event that command has been executed
  - Consequences e.g. for buffer handling
    - Buffers can only be freed when completion event is received
Building components out of simpler ones

- Wire together components to form more complex components out of simpler ones
- New interfaces for the complex component
Summary

- For WSN, the need to build cheap, low-energy, (small) devices has various consequences for system design
  - Radio frontends and controllers are much simpler than in conventional mobile networks
  - Energy supply and scavenging are still (and for the foreseeable future) a premium resource
  - Power management (switching off or throttling down devices) crucial
- Unique programming challenges of embedded systems
  - Concurrency without support, protection
  - De facto standard: TinyOS
Thank you

(and thanks go also to Holger Karl for providing slides)

Wireless Sensor Networks
Christian Schindelhauer
schindel@informatik.uni-freiburg.de

6th Lecture
14.11.2006