Single Node Architecture
Single Node Architecture

Sensor Node Architecture
Sensor Node Hierarchy

- **Web interfaces, databases**
  - The Internet
  - A few gateway nodes

- **Cameras, microphones**
  - Dozens of high-bandwidth sensors

- **Door, window, motion sensors**
  - Hundreds of generic sensor nodes

- **Asset tags**
  - Thousands of special-purpose sensors
Components of a Sensor Node

- **Power unit**
  - Power generator

- **Location finding system**
  - Mobilizer

- **Sensing unit**
  - Sensor
  - ADC

- **Processing unit**
  - Processor
  - Storage

- **Transceiver unit**
Example: Tmote Sky Sensor Module
Example: Tmote Sky Sensor Module

- Texas Instruments MSP430 F1611 microcontroller
- 48-bit silicon serial ID
- 2-pin SVS connector
- USB Flash (2kB)
- 32kHz oscillator
- ST Code Flash (1MB)
But there are much more …

- BEAN
- BTnode
- COTS
- Dot
- Ember
- Eyes
- FireFly
- Fleck
- IMote
- Imote2
- KMote
- Mica
- Mica2
- Mica2Dot
- MicaZ
- Mulle
- Nymph
- Particles
- Rene
- ScatterWeb
- Sensinode
- SHIMMER
- SquidBee
- SunSPOT
- Telos
- TinyNode 584
- T-Mote Sky
- T-Nodes
- WeBee
- weC
- XYZ
- WINS
- WiseNet
Example Generic Sensor Hardware

Source: http://www.btnode.ethz.ch/Projects/SensorNetworkMuseum
MANETs: Ad hoc Node Architecture

- More powerful
  - Processor: Strong ARM, Pentium, …
  - RAM
- Much more additional equipment
  - Hard disk, display, keyboard, voice interface, camera, …
- Essentially: a laptop-class device
WSN Nodes use Controllers, not Processors

- Main options:
  - Microcontroller – general purpose processor, optimized for embedded applications
    - Low power consumption
    - Low speed
    - Low cost
    - On-board RAM
  - DSPs – optimized for signal processing tasks, not suitable here
  - FPGAs – may be good for testing
  - ASICs – only when peak performance is needed, no flexibility

- Example microcontrollers
  - Texas Instruments MSP430
    - 16-bit RISC core, up to 4 MHz, versions with 2-10 kbytes RAM, several DACs, RT clock, prices start at 0.49 US$
  - Atmel ATMega
    - 8-bit controller, larger memory than MSP430, slower
Communication device

- Which transmission medium?
  - Electromagnetic at radio frequencies? ✓
  - Electromagnetic, light?
  - Ultrasound?

- Radio transceivers
  - Transmit a sequence of bits or bytes as radio wave
  - Receive it, convert it back into bit-/byte stream
A Question: How long does a WSN last?

- **Tmote sky product description**
  - Processor: 500 uA
  - Radio TX: 17.4 mA

- **Standard AA Alkaline Battery**
  - Single battery: 2200 mAh
  - Module’s battery pack: 4400 mAh

- **Sensor node’s lifetime**
  - Current consumption: $0.5 + 17.4 = 17.9$ mA
  - Lifetime: $4400/17.9$ mAh/mA $\approx 246$ h $\approx 10.2$ days
How to extend the lifetime?

- Idea: sample periodically

- Tmote sky product description
  - Processor: 500 uA
  - Radio TX: 17.4 mA
  - Radio on: 365 uA

- Example: one 16 bit sample per second; idealized channel
  - Fraction used for transmission: \( X = \frac{16}{250 \times 1024} \)
  - Current consumption: \( Y = 0.5 + 17.4 \times X + 0.365 \times (1-X) \approx 0.87 \text{ mA} \)
  - Lifetime: \( \frac{4400}{Y} \approx 5080 \text{ h} \approx 211.7 \text{ days} \)
Can they Survive a whole Year?

- Idea: switch off unnecessary current consumers

```java
while(true) {
    turn on consumers;
    x = read sensor;
    transmit x;
    turn off consumers;
    await next sec;
}
```

- Tmote sky product description
  - Processor: 500 μA
  - Radio TX: 17.4 mA
  - Radio power down: 1 μA

- Example: one 16 bit sample per second; idealized channel
  - Fraction used for transmission: \( X = \frac{16}{250 \times 1024} \)
  - Current consumption: \( Y = 0.5 + 17.4 \times X + 0.001 \times (1-X) \approx 0.5 \) mA
  - Lifetime: \( \frac{4400}{Y} \approx 8763 \) h \( \approx 365.1 \) days
Can they Survive for many Years?

- Idea: set MCU in sleep mode

- Tmote sky product description
  - Processor: 500 \(\mu\)A
  - Processor sleep@32kHz: 2.6 \(\mu\)A
  - Radio TX: 17.4 mA
  - Radio Idle: 365 \(\mu\)A
  - Radio power down: 1 \(\mu\)A
  - Processor wakeup: 6 us
  - Radio oscillator startup: 580 us

- Example: one 16 bit sample per second; idealized channel
  - Transmission time = \(\frac{16}{(250 \times 1024)}\) s = 62.5 us

\[
Y = \frac{6}{10^6} \times (500+1) + \frac{580}{10^6} \times (500+365) + \frac{62.5}{10^6} \times (500+17400) + \frac{(10^6-6-580-62.5)}{10^6} \times (2.6+1)
\]

\[
Y = 5.22 \mu A
\]

- Lifetime: \(4400 \text{ mAh} / 0.00522 \text{ mA} \approx 842912 \text{ h} \approx 35121 \text{ days} \approx 96 \text{ years (simplified model!)}\)

In practice lifetime of a few years:
More sources of power dissipation
Synchronization of communication nodes
Battery looses current!!!
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
## Example Sensor Network Transceivers

<table>
<thead>
<tr>
<th></th>
<th>CC1000</th>
<th>CC1021</th>
<th>CC2420</th>
<th>TR1000</th>
<th>XE1205</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bit Rate [kbps]</strong></td>
<td>76.8</td>
<td>153.6</td>
<td>250</td>
<td>115.2</td>
<td>1.2 - 152.3</td>
</tr>
<tr>
<td><strong>Sleep Mode [uA]</strong></td>
<td>0.2 - 1 (osc. core off)</td>
<td>1.8 (core off)</td>
<td>1</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td><strong>RX [mA]</strong></td>
<td>9.3 (433MHz) / 11.8 (868MHz)</td>
<td>19.9</td>
<td>19.7</td>
<td>3.8 (115.2kbps)</td>
<td>14</td>
</tr>
<tr>
<td><strong>TX Min [mA]</strong></td>
<td>8.6 (-20dBm)</td>
<td>14.5 (-20dBm)</td>
<td>8.5 (-25dBm)</td>
<td>33 (+5dBm)</td>
<td></td>
</tr>
<tr>
<td><strong>TX Max [mA]</strong></td>
<td>25.4 (+5dBm)</td>
<td>25.1 (+5dBm)</td>
<td>17.4 (0dBm)</td>
<td>12 (+1.5dBm)</td>
<td>62 (+15dBm)</td>
</tr>
</tbody>
</table>

How much is 250 kbps?

- 250 kbit transmit bit rate can serve
  - Two 128 kbit 320 x 240 thumbnail video streams
  - Four 64 kBit digital high quality audio streams
  - 250 * 1024 / 16 = 16000 16 bit Samples per second

- Can we use 250 kbps?
  - Error correcting codes
  - Retransmissions
  - Shared media

Parallel stream transmissions

Serial stream transmission
What is dBm?

- Logarithmic expression of power in mW
- Conversion
  - \( P \text{ mW} \rightarrow x \text{ dBm} : x = 10 \log_{10}(P) \)
  - \( x \text{ dBm} \rightarrow P \text{ mW} : P = 10^{(x/10)} \)
- Examples (from wikipedia)

<table>
<thead>
<tr>
<th>dBm level</th>
<th>Power</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 dBm</td>
<td>100 kW</td>
<td>Typical transmission power of a <a href="https://en.wikipedia.org/wiki/FM_broadcasting">FM radio</a> station</td>
</tr>
<tr>
<td>60 dBm</td>
<td>1 kW = 1000 W</td>
<td>Typical RF power inside a <a href="https://en.wikipedia.org/wiki/Microwave_oven">microwave oven</a></td>
</tr>
<tr>
<td>36 dBm</td>
<td>4 W</td>
<td>Typical maximum output power for a <a href="https://en.wikipedia.org/wiki/Citizens'_band">Citizens' band radio</a> station (27 MHz) in many countries</td>
</tr>
<tr>
<td>30 dBm</td>
<td>1 W = 1000 mW</td>
<td>Typical RF leakage from a microwave oven - Maximum output power for DCS 1800 MHz mobile phone</td>
</tr>
<tr>
<td>27 dBm</td>
<td>500 mW</td>
<td>Typical <a href="https://en.wikipedia.org/wiki/Cordless_telephone"> cellular phone</a> transmission power</td>
</tr>
<tr>
<td>21 dBm</td>
<td>125 mW</td>
<td>Maximum output from a <a href="https://en.wikipedia.org/wiki/Universal_Mobile_Communications_System">UMTS/3G</a> mobile phone (Power class 4 mobiles)</td>
</tr>
<tr>
<td>20 dBm</td>
<td>100 mW</td>
<td><a href="https://en.wikipedia.org/wiki/Bluetooth">Bluetooth</a> Class 1 radio, 100 m range (maximum output power from unlicensed FM transmitter)</td>
</tr>
<tr>
<td>4 dBm</td>
<td>2.5 mW</td>
<td>Bluetooth Class 2 radio, 10 m range</td>
</tr>
<tr>
<td>0 dBm</td>
<td>1.0 mW = 1000 µW</td>
<td>Bluetooth standard (Class 3) radio, 1 m range</td>
</tr>
<tr>
<td>−70 dBm</td>
<td>100 pW</td>
<td>Typical range (~60 to ~80 dBm) of Wireless signal over a network</td>
</tr>
<tr>
<td>−111 dBm</td>
<td>0.008 pW</td>
<td>Thermal noise floor for commercial <a href="https://en.wikipedia.org/wiki/Global_Positioning_System">GPS</a> signal bandwidth (2 MHz)</td>
</tr>
<tr>
<td>−127.5 dBm</td>
<td>0.000178 pW</td>
<td>Typical received signal power from a GPS satellite</td>
</tr>
<tr>
<td>−174 dBm</td>
<td>0.000004 fW</td>
<td>Thermal noise floor for 1 Hz bandwidth</td>
</tr>
</tbody>
</table>
Example radio transceivers for ad hoc networks

- Ad hoc networks: Usually, higher data rates are required
- Typical: IEEE 802.11 b/g/a is considered
  - Up to 54 MBit/s
  - Relatively long distance (100s of meters possible, typical 10s of meters at higher data rates)
  - Works reasonably well (but certainly not perfect) in mobile environments
  - Problem: expensive equipment, quite power hungry
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 “titled” to stop reflecting
  - Allows data to be sent back to laser source

200 μm
Communication with the Base Station

Modulated Downlink Data or Unmodulated Interrogation Beam for Uplink

Downlink Data In ➔ Laser ➔ Lens ➔ CCD Image Sensor Array ➔ Modulated Reflected Beam for Uplink

Signal Selection and Processing

Uplink Data Out₁ ➔ Uplink Data Outₙ

Base-Station Transceiver

Photo-detector ➔ Downlink Data Out

Corner-Cube Retroreflector ➔ Uplink Data In

Dust Mote
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 to determine node’s distances
  - 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?
Single Node Architecture

Energy Supply and Consumption
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
  - Capacity under load
  - Efficient recharging at low current
  - Voltage stability (to avoid DC-DC conversion)
## Battery examples

- Energy per volume (Joule per cubic centimeter):

<table>
<thead>
<tr>
<th>Primary batteries</th>
<th>Chemistry</th>
<th>Zinc-air</th>
<th>Lithium</th>
<th>Alkaline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy (J/cm³)</td>
<td></td>
<td>3780</td>
<td>2880</td>
<td>1200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secondary batteries</th>
<th>Chemistry</th>
<th>Lithium</th>
<th>NiMHD</th>
<th>NiCd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy (J/cm³)</td>
<td></td>
<td>1080</td>
<td>860</td>
<td>650</td>
</tr>
</tbody>
</table>
What is one Joule?

- $1 \text{ J} = 1 \text{ Nm} = 1 \text{ kg m}^2/\text{s}^2 = 1 \text{ W s}$

- In words
  - Energy expended by a force of one newton moving one meter along the direction of the force
    - Example: $1 \text{ N}$ is the force of earth's gravity on an object with a mass of about $102 \text{ g}$ (such as a small apple).
  - Or energy required to apply the power of one Watt for the duration of one second
    - Example: human climbing a flight of stairs is doing work at a rate of about $200 \text{ watts}$
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 \( \mu W/cm^2 \) and 15 mW/cm\(^2\)
  • Temperature gradients – 80 \( \mu W/cm^2 \) @ 1 V from 5K difference
  • Vibrations – between 0.1 and 10000 \( \mu W/cm^3 \)
  • Pressure variation (piezo-electric) – 330 \( \mu W/cm^2 \) 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>$1050 \text{ – } 1560 \text{ mWh/cm}^3$</td>
</tr>
<tr>
<td>Batteries (rechargable lithium)</td>
<td>$300 \text{ mWh/cm}^3$ (at $3 \text{ – } 4 \text{ V}$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy source</th>
<th>Power density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solar (outdoors)</td>
<td>$15 \text{ mW/cm}^2$ (direct sun)</td>
</tr>
<tr>
<td></td>
<td>$0.15 \text{ mW/cm}^2$ (cloudy day)</td>
</tr>
<tr>
<td>Solar (indoors)</td>
<td>$0.006 \text{ mW/cm}^2$ (standard office desk)</td>
</tr>
<tr>
<td></td>
<td>$0.57 \text{ mW/cm}^2$ (&lt;60 W desk lamp)</td>
</tr>
<tr>
<td>Vibrations</td>
<td>$0.01 \text{ – } 0.1 \text{ mW/cm}^3$</td>
</tr>
<tr>
<td>Acoustic noise</td>
<td>$3 \cdot 10^{-6} \text{ mW/cm}^2$ at 75 Db</td>
</tr>
<tr>
<td></td>
<td>$9.6 \cdot 10^{-4} \text{ mW/cm}^2$ at 100 Db</td>
</tr>
<tr>
<td>Passive human-powered systems</td>
<td>$1.8 \text{ mW}$ (shoe inserts)</td>
</tr>
<tr>
<td>Nuclear reaction</td>
<td>$80 \text{ mW/cm}^3$, $10^6 \text{ mWh/cm}^3$</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!
  - 8 MHz : $8 \times 10^6$ instructions $\Rightarrow 10 \times 10^9 / 8 \times 10^6 = 1250$ sec $= 20.83$ min

- Lifetime
  - Or: Require a single day operational lifetime = $24 \times 60 \times 60 = 86400$ s
  - $1$ Ws / $86400$ s = $11.5 \mu W$ as max. sustained power consumption!
  - $11500$ nJ / sec $\Rightarrow 11500$ Instructions / sec $\Rightarrow 11.5$ kHz

- 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 operational 1.2 mW
    - Deepest sleep mode 0.3 \( \mu \)W – 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 \( \mu \)W
Switching between modes

- Simplest idea: Greedily switch to lower mode whenever possible
- Problem: Time and power consumption required to reach higher modes not negligible
  - Introduces overhead
  - Switching only pays off if $E_{\text{saved}} > E_{\text{overhead}}$

![Diagram showing energy consumption](image)
Alternative: Dynamic voltage scaling

- Switching modes complicated by uncertainty about sleep periods
- Alternative: Low supply voltage & clock
  - *Dynamic voltage scaling* (DVS)
- Rationale:
  - Power consumption $P$ depends on
    - Square of supply voltage (see right box)
    - Clock frequency (simplified example: working with double speed means doubling power consumption (J/sec))
    - $P \sim f V^2$ (i.e. $P = C f V^2$)
  - Lower clock allows lower supply voltage
  - Easy to switch to higher clock
  - But: execution takes longer

- Some Power Formulas
  - $P = V^2 / R$
  - $P = R I^2$
  - $P = V I$

- It follows when $I$ and $R$ is fixed:
  - $P \sim V^2$

- Use infinitesimal small $V$ to run circuit?
  - Speed at which circuit can change from “low” to “high” is proportional to voltage differential
  - Thus, max allowed frequency is proportional to $V$
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 ($\eta = \frac{P_{\text{tx}}}{P_{\text{amp}}}$) at maximum output power
- In addition: transmitter electronics needs power $P_{\text{txElec}}$
- Time to transmit n bits: $n / (R \times R_{\text{code}})$
  - $R$ nominal data rate, $R_{\text{code}}$ coding rate
- To leave sleep mode
  - Time $T_{\text{start}}$, average power $P_{\text{start}}$

$$E_{\text{tx}} = T_{\text{start}} P_{\text{start}} + \frac{n}{(R \times R_{\text{code}})} (P_{\text{txElec}} + \alpha_{\text{amp}} + \beta_{\text{amp}} P_{\text{tx}})$$

- 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 $n / (R \times R_{\text{code}})$
- Receiver electronics needs $P_{\text{rxElec}}$
- Plus: energy to decode $n$ bits $E_{\text{decBits}}$

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>μAMPS-1 [559]</th>
<th>WINS [670]</th>
<th>MEDUSA-II [670]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{amp}$</td>
<td>Eq. (2.4)</td>
<td>174 mW</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>$\beta_{amp}$</td>
<td>Eq. (2.4)</td>
<td>5.0</td>
<td>8.9</td>
<td>7.43</td>
</tr>
<tr>
<td>$P_{amp}$</td>
<td>Amplifier pwr.</td>
<td>179 – 674 mW</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>$P_{rxElec}$</td>
<td>Reception pwr.</td>
<td>279 mW</td>
<td>368.3 mW</td>
<td>12.48 mW</td>
</tr>
<tr>
<td>$P_{rxIdle}$</td>
<td>Receive idle</td>
<td>N/A</td>
<td>344.2 mW</td>
<td>12.34 mW</td>
</tr>
<tr>
<td>$P_{start}$</td>
<td>Startup pwr.</td>
<td>58.7 mW</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>$P_{txElec}$</td>
<td>Transmit pwr.</td>
<td>151 mW</td>
<td>≈ 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>$T_{start}$</td>
<td>Startup time</td>
<td>466 µs</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 (DMS): 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, …
Single Node Architecture

Runtime Environments for Sensor Networks
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
  - Processes 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

• Need an abstraction to group functionality
  • Replacing “processes” for this purpose
  • E.g.: individual functions of a networking protocol

• One option: *Components*
  • Here: In the sense of TinyOS
  • Typically fulfill only a single, well-defined function
  • Main difference to processes:
    • Component does not have an execution
    • Components access same address space, no protection against each other
  • NOT to be confused with component-based programming!
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 to power-safe mode?
  - Problem: Time & energy overhead associated with wakeup; greedy sleeping is not beneficial
  - Scheduling approach
- Question: How to control dynamic voltage scaling?
  - More aggressive; stepping up voltage/frequency is easier
  - Deadlines usually bound the required speed form 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
Single Node Architecture

Case Study: TinyOS
Case study embedded OS: TinyOS & nesC

- TinyOS developed by UC Berkeley 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 emit commands & events
- Hierarchically arranged
  - Events pass upward from hardware to higher-level components
  - Commands are passed downward
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 for 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
Structuring commands/events into interfaces

- Many commands/events can add up
- nesC solution: Structure corresponding commands/events into *interface types*
- Example: Structure timer into three interfaces
  - StdCtrl
  - Timer
  - Clock
- Build configurations by wiring together corresponding interfaces
Building components out of simpler ones

- Wire together components to form more complex components out of simpler ones
- New interfaces for the complex component
Defining modules and components in nesC

```c
interface StdCtrl {
    command result_t init();
}

interface Timer {
    command result_t start (char type, uint32_t interval);
    command result_t stop ();
    event result_t fired();
}

interface Clock {
    command result_t setRate (char interval, char scale);
    event result_t fire ();
}

module TimerComponent {
    provides {
        interface StdCtrl;
        interface Timer;
    }
    uses interface Clock as Clk;
}
```
Wiring components to form a configuration

```java
configuration CompleteTimer {
  provides {
    interface StdCtrl;
    interface Timer;
  }
  implementation {
    components TimerComponent, HWClock;
    StdCtrl = TimerComponent.StdCtrl;
    Timer = TimerComponent.Timer;
    TimerComponent.Clk = HWClock.Clock;
  }
}
```
Single Node Architecture

Summary
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