# TOWARDS SCIENTIFIC WIRELESS SENSOR NETWORKS ENGINEERING

Philippe Bonnet
bonnet@diku.dk

# Outline

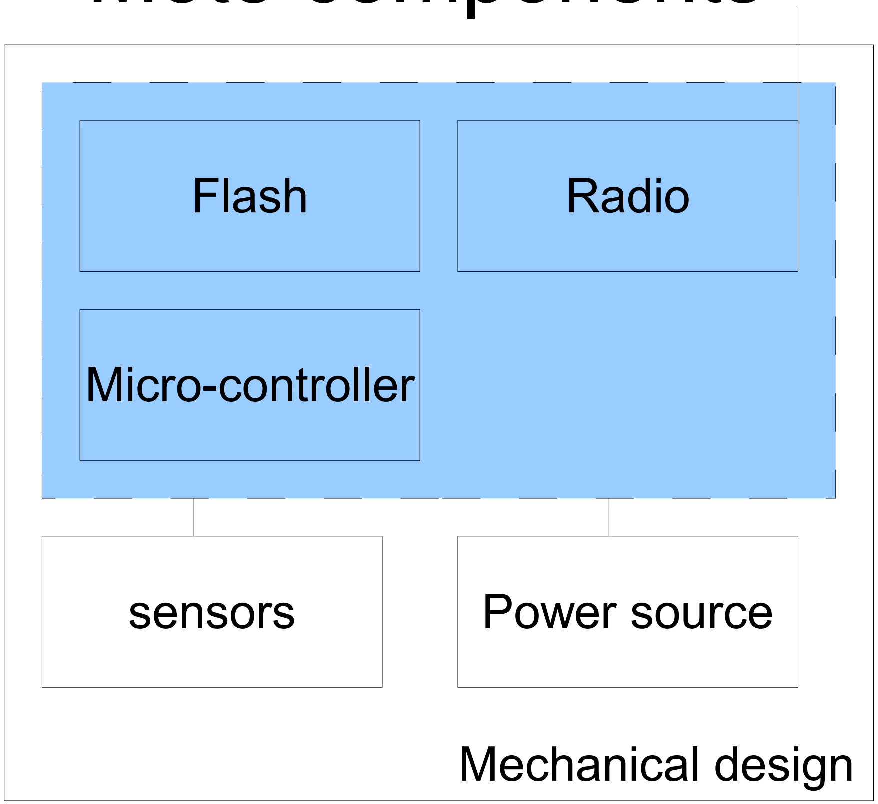
- Introduction
  - Wireless Sensor Networks
  - Scientific observations
  - Engineering Challenges
- Lessons from Hogthrob
- Characterizing Mote Performance
- Perspectives
  - The MANA project

# The WSN Vision

- D.Estrin (April 99)
  - Embed numerous distributed devices to monitor and interact with physical world
  - Leverage off pervasive physical locality between nodes and subject
- G.Bell (January 08)
  - WSN new computer class forming since 2002
    - TinyOS vs. Java
    - 802.15.4, 6LoWPAN
    - MEMS-based sensors
  - WSN to become a true class interconnecting everything vs. just unwiring the world ?

# Mote-Based Instrumentation

- Mote components

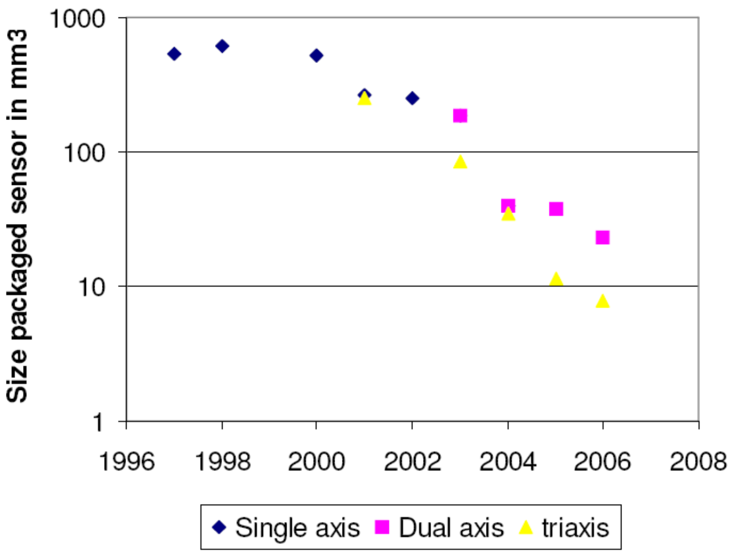



- Network structure
  - Gateways connect cluster of motes to Internet
  - Topology
    - Mesh
    - Epidemic replication (mobile motes)
- Mote Power consumption

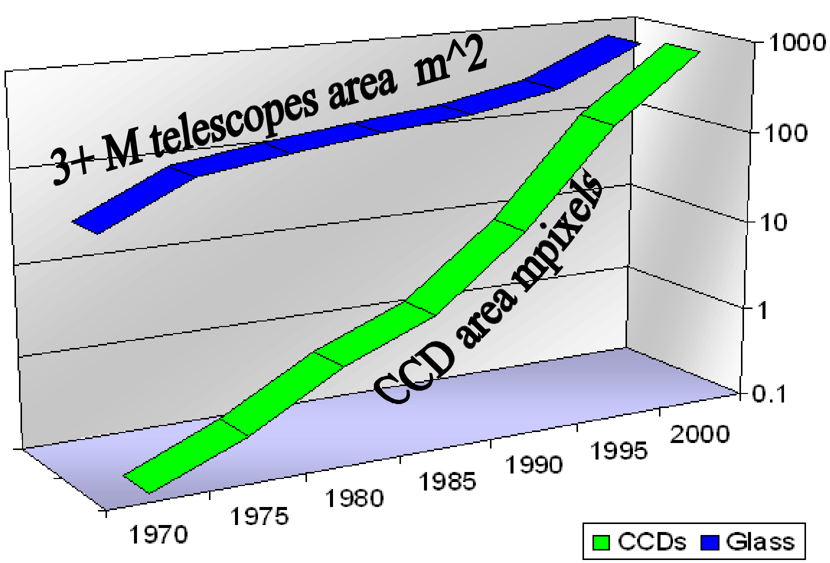
P_avg = P_active
+ P_sleep
+ P_wakeup

# The Rationale

Source: enablingMNT



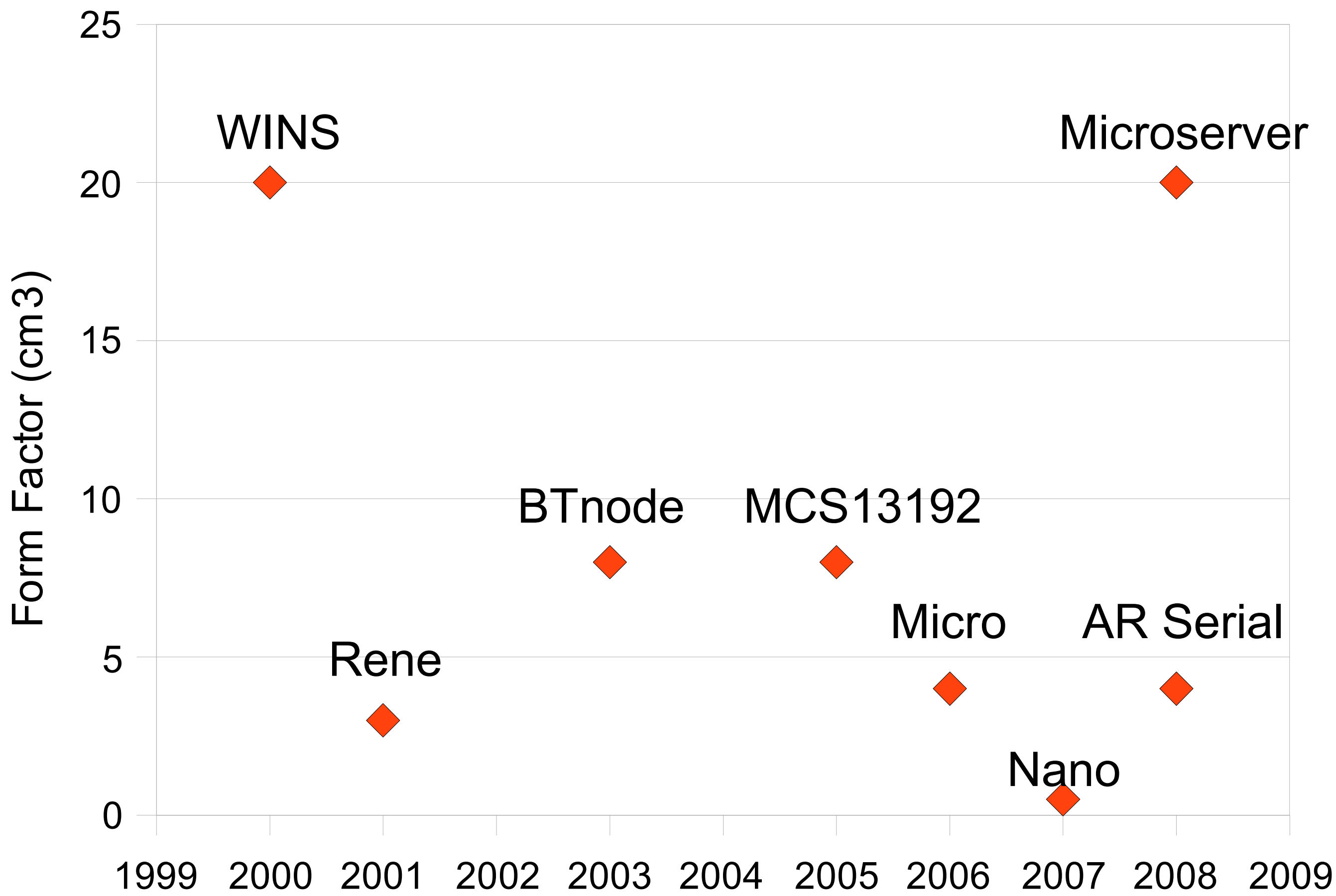

Astronomy showing the potential
of inexpensive sensor + computing




Graph courtesy of A.Szalay

- MEMS-based sensors
  - Mechanical, optical, thermal, chemical, biological
  - Integration with CMOS
  - Lower price, smaller smaller form factor, lower power consumption.
  - MEMS-based energy scavenging

# A Personal Journey




# Scientific Observation

- Ongoing efforts to integrate terrestrial, airborne and satellite observation systems
  - Earth science
  - Sensorweb: geoss, neon, saon


- WSN-based data acquisition systems promise to:
  - Improve in-situ sensing
    - Unwired: Sensors in new places
    - Taskable: Energy efficient, multi-purpose
    - Cheap: Higher coverage/density
  - Improve integrated observation systems
    - Online: live data assimilation

# Scientific WSN

- Mote-based system
  - Independant power-source
- Programmed for time series acquisition
- Provides telemetry/telecommand interface for the back-end infrastructure
- Meets scientist goals
  - E.g., form factor, lifetime, yield, unattended period

Maintaining high-utilization

cascading/conflicting goals

In order to scale the number/quality of deployed WSN,
we need a systematic, disciplined, quantifiable approach to
the design and operation of Scientific WSN

# Scientific WSN Engineering Issues

- What are the system components?
  - What are the modules a scientific WSN should be composed of?
  - What generic modules can be reused across scientific WSNs?
- What is the programming abstraction?
  - How to deal with time series collection?
  - How to deal with failures?
  - How to garantee some quality of service throughout a deployment?
- How to navigate the design space?
  - Which modules to reuse/develop?
  - How to evaluate a given design?
- What are design principles for scientific WSN?

# Related Work

- Scientific WSN
  - UCB, CENS, JHU, EPFL, Envisense, ..
- WSN Engineering
  - SNAP @ UCB/Stanford
  - Macroscopic Simulation models @ EPFL
  - Deployment support networks @ ETHZ
  - Autonomous system @ MIT

# Outline

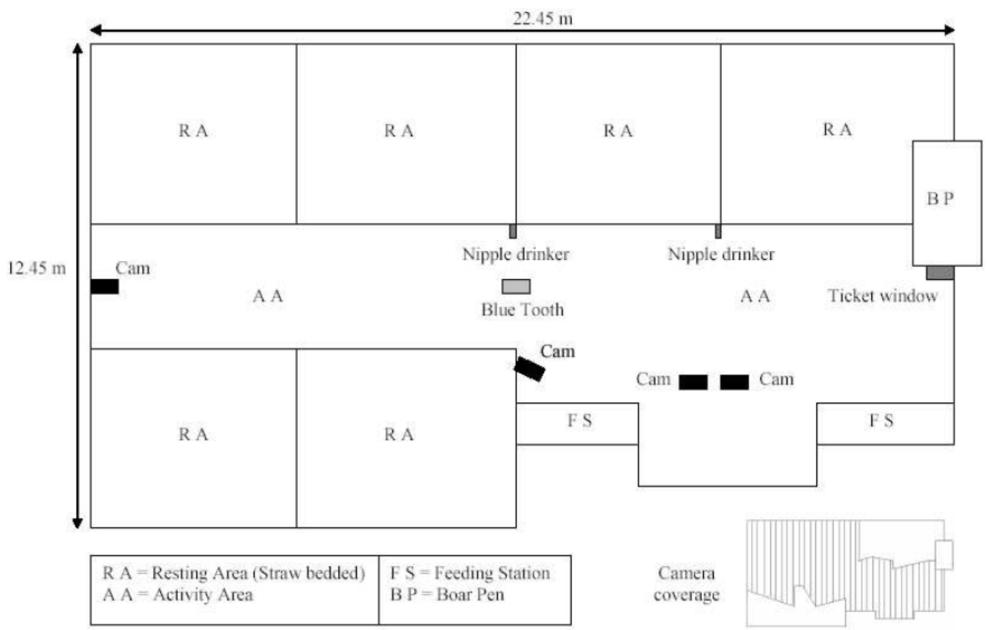
- Introduction
  - Wireless Sensor Networks
  - Scientific observations
  - Engineering Challenges
- Lessons from Hogthrob
  - [WiDeploy08] Chang, Madsen, Cornou, Bonnet
- Characterizing Mote Performance
- Perspectives
  - The MANA project

# Hogthrob Context

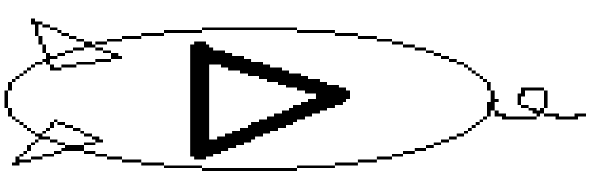
- Infrastructure for sow monitoring
- Today, RFID-based ear-tags to monitor food-intake
- Tomorrow:
  - Life-cycle monitoring: heat, farrowing, localization, injury detection.


Consortium: DTU, KVL, DIKU, National Comittee for pig production
Students: Klaus Madsen, Eske Christiansen,
Marcus Chang


# Hogthrob WSN

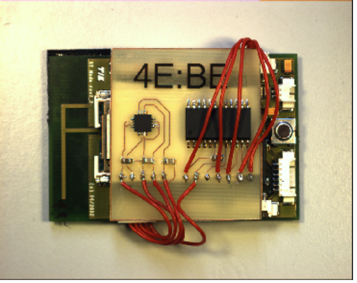



<u>Functionality</u>: Data Acquisition so that veterinarians can model the behaviour of group-housed sows.



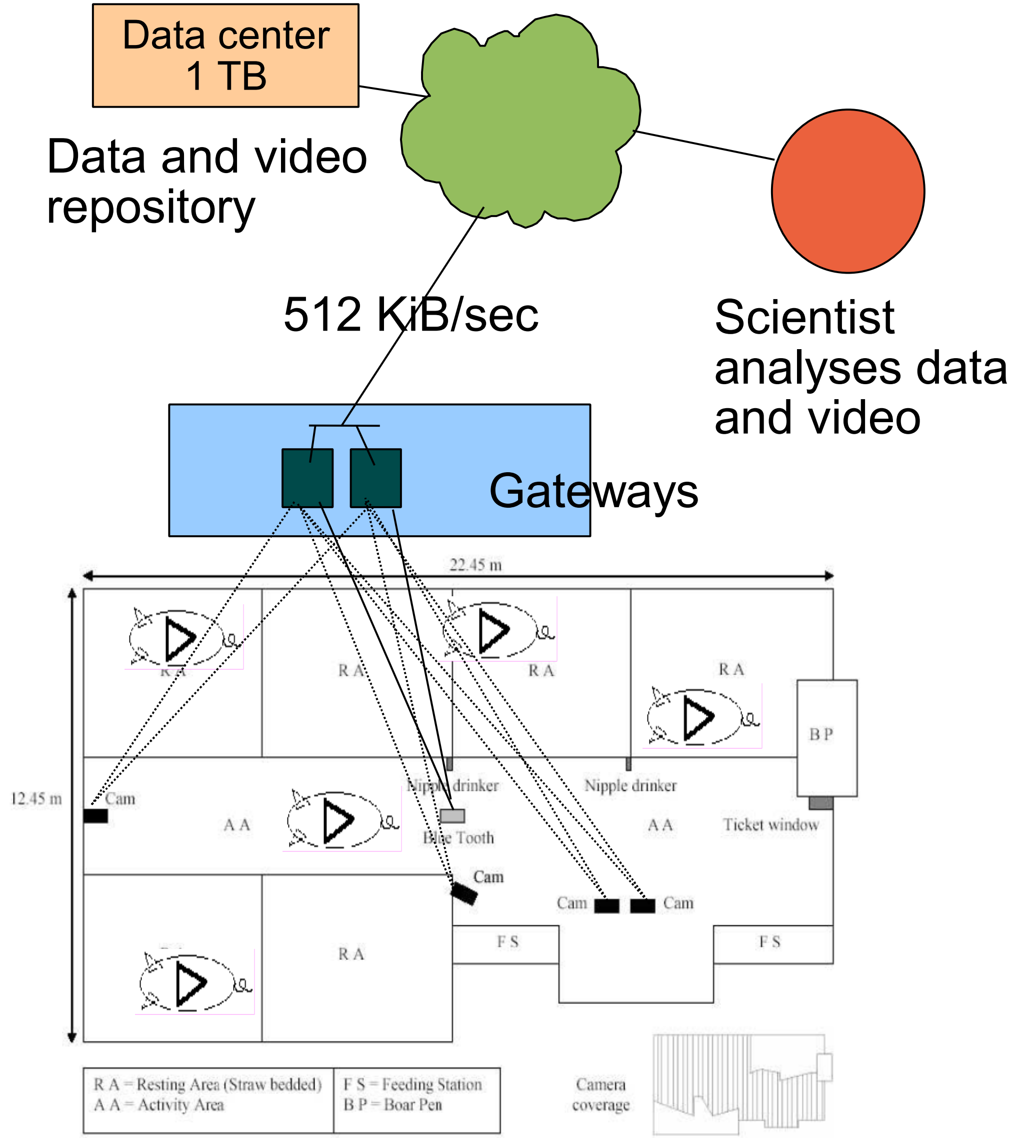
# Deployment#1 (Feb 2005)

- Goal:
  - 3 weeks
  - 5 sows
  - Collect time series
    - 2D and 3D accelerometers
    - 4 Hz sampling ~200 MiB / mote
  - Video as ground truth
    - BW Surveillance camera (AVC301a)
    - 8KiB per PNG frame
    - 32 KiB/sec
    - Total: 80 GiB per camera



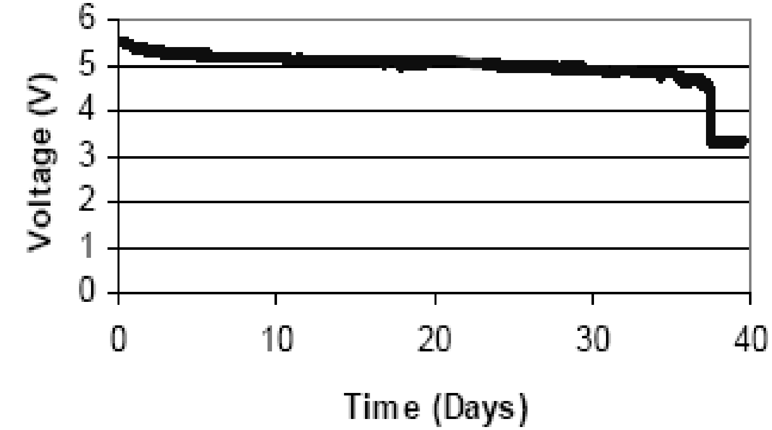
# Deployment #1

- Approach
  - Using BTNodes V2 running TinyOS
    - Bluetooth radio
    - Internal flash (~100KiB)
    - Aggressive duty cycling: sample and transmit when flash is full
    - 4 NiMH cells (1.25V, 2400mA)
  - Logical time stamp on motes - physical time stamp on gateway



# Deployment #1

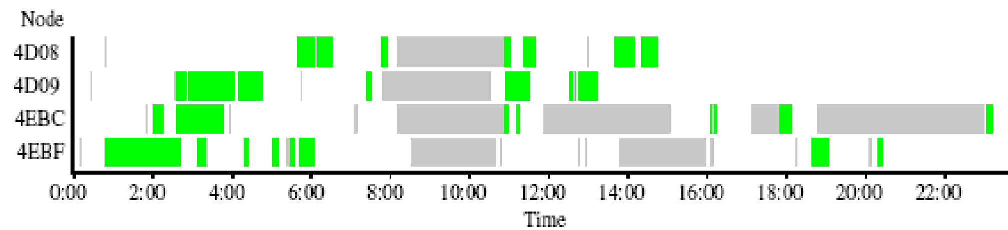
- Results
  - Lifetime requirement met
  - Disappointing yield
  - Painful post-processing
    - Node reboots
    - Time difference across the two gateways



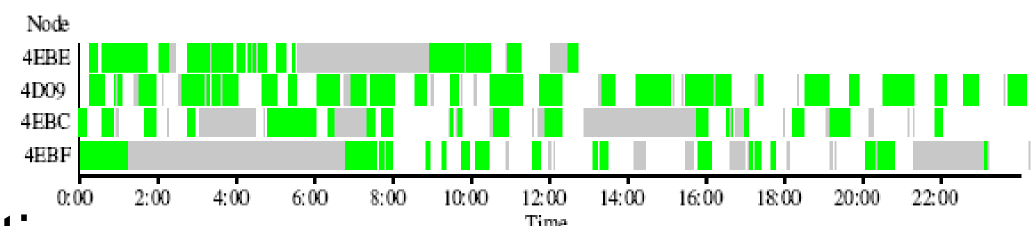

| Node | First Measurement | Last Measurement | Days | Coverage |
|---|---|---|---|---|
| 4D08 | 2005/02/28 14:05 | 2005/03/13 20:31 | 14 | 44.607% |
| 4D09 | 2005/02/28 09:59 | 2005/03/19 22:47 | 20 | 71.756% |
| 4D0B | 2005/02/28 10:06 | 2005/03/19 22:48 | 20 | 64.567% |
| 4EBC | 2005/02/28 10:28 | 2005/03/19 21:06 | 20 | 56.139% |
| 4EBE | 2005/03/15 11:07 | 2005/03/19 21:56 | 5 | 61.910% |
| 4EBF | 2005/02/28 10:09 | 2005/03/19 23:02 | 20 | 66.717% |

# Deployment #1

- Results (cont.)
  - Missing data largely correlated with sleep periods
  - Despite the low yield, promising data for activity model
    - Using elastic burst detection [Zhu and Shasha 2003]



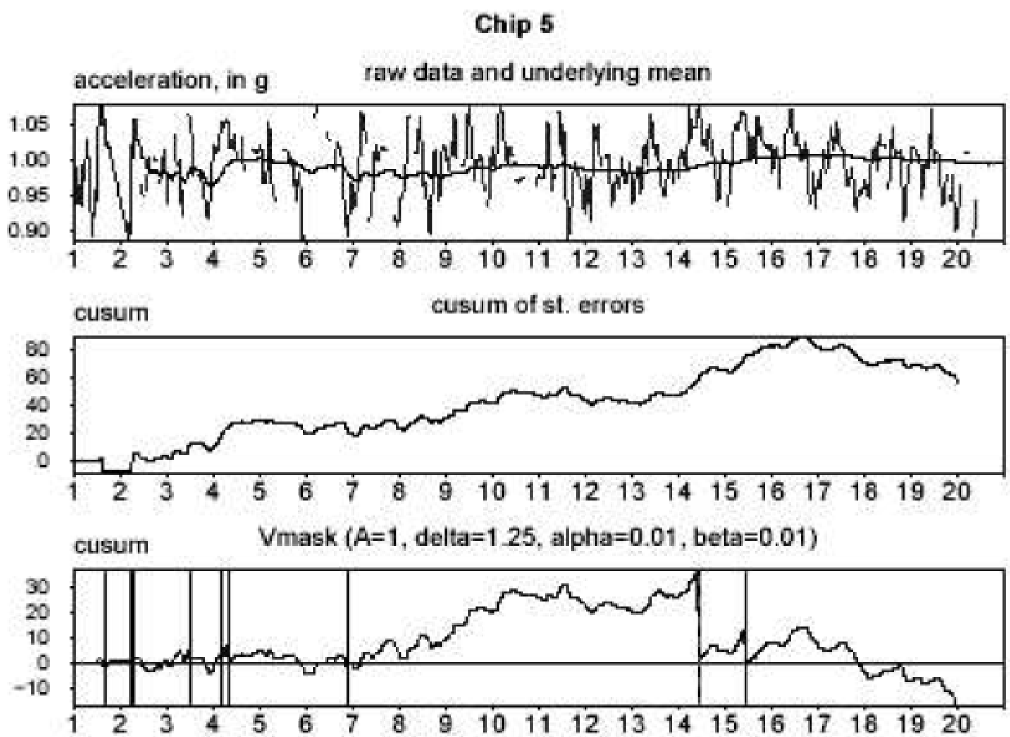

(a) 1st of March




(b) A day in the middle of the heat-period

**Lesson learned**:
Design should focus on improving yield
- Define degraded modes
- Improve on-mote storage

# Experiment #1



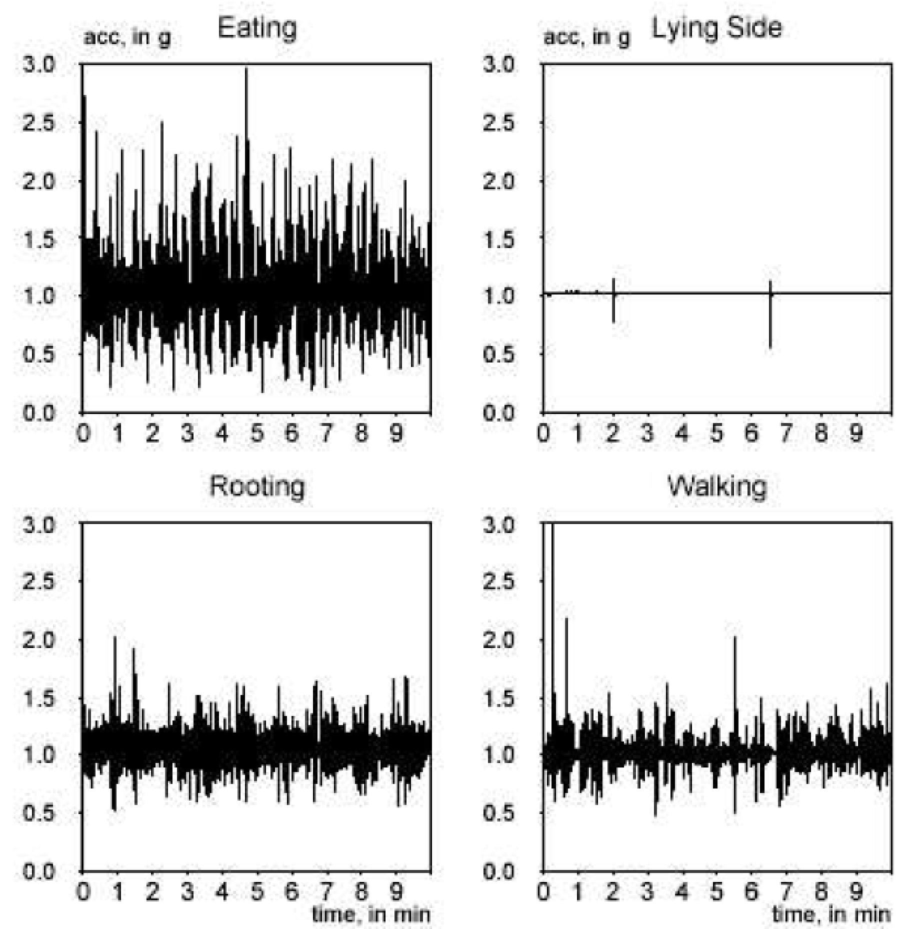



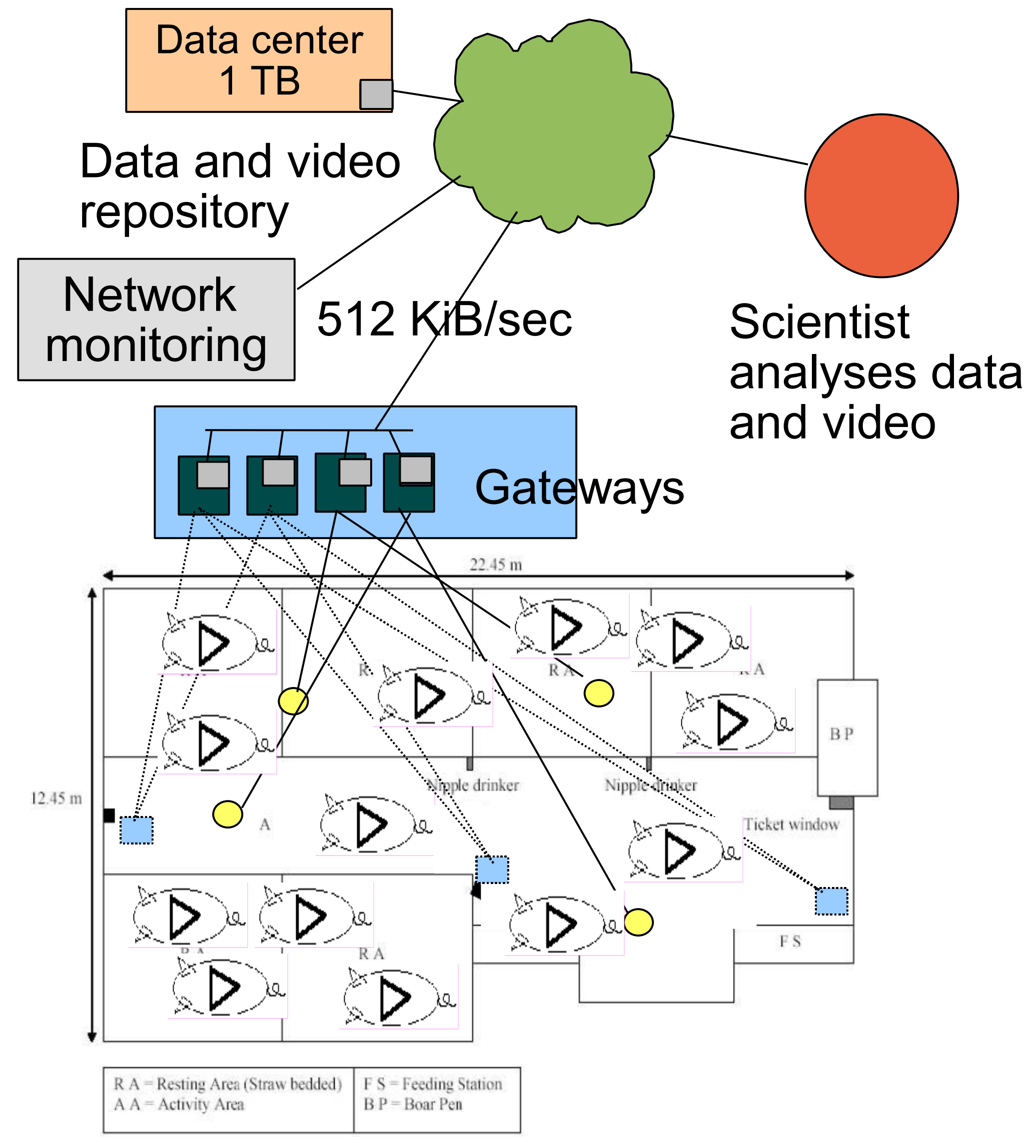

Figure courtesy of Cecile Cornou

# Deployment #2 (Feb 2007)

- Goal:
  - 3 weeks
  - 12 sows
    - Batch1: 6 sows
    - Batch2: 6 sows
  - Collect time series
    - 3D accelerometers
    - 4 Hz sampling
    - <u>90% yield</u>

# Deployment #2

- Approach:
  - Sensinode micro running TinyOS
    - 802.154 radio
    - 512 KiB flash
    - Accelerometer board
    - 2 NiMH cells (1.25V, 2400mA)
    - Compression
    - Control loops over sample, compress, store, send
  - Access points with external antennas



# Deployment #2

- Batch1

| mote | samples | received | % | clean | % | errors | promille |
|---|---|---|---|---|---|---|---|
| 33cb | 6659775 | 6155871 | 92.4 | 5725802 | 86.0 | 10224 | 1.5 |
| 4228 | 6664812 | 6202083 | 93.1 | 5853400 | 87.8 | 13372 | 2.0 |
| 5d01 | 6659416 | 6316440 | 94.8 | 6295749 | 94.5 | 8842 | 1.3 |
| 5f21 | 6660201 | 6310652 | 94.8 | 6289002 | 94.4 | 9338 | 1.4 |
| 6902 | 6664218 | 6237412 | 93.6 | 5908038 | 88.7 | 13325 | 2.0 |
| 7a40 | 6667783 | 5947416 | 89.2 | 5358573 | 80.4 | 4963 | 0.7 |

- Batch 2

| mote | samples | received | % | clean | % | errors | promille |
|---|---|---|---|---|---|---|---|
| 472b | 6525533 | 6236216 | 95.6 | 6196884 | 95.0 | 6373 | 1.0 |
| 4830 | 6693554 | 6336382 | 94.7 | 6310683 | 94.3 | 2438 | 0.4 |
| 4e78 | 6692651 | 6352667 | 94.9 | 6352204 | 94.9 | 6903 | 1.0 |
| 72ca | 6694049 | 6434630 | 96.1 | 6394319 | 95.5 | 12599 | 1.9 |
| 73c9 | 5910142 | 5398960 | 91.4 | 5019267 | 84.9 | 11704 | 2.0 |
| 7c60 | 6694338 | 6230244 | 93.1 | 5722659 | 85.5 | 2279 | 0.3 |

# Lessons Learned

- Degraded modes
  - Kick in case of failure / pressure
  - Lossy mode based on how collected data is to be used
- Key mote characteristics
  - Amount of storage
  - Time during which the radio must be on
  - Power consumption in sleep mode

# Lessons Learned

- Post-processing factored in
  - Cleaning, timestamping, uncompression
  - Association of Raw data to calibration info
  - Data export (format transformation)
- Beyond best effort (future work)
  - Collected data analyzed online
    - Detection of error / interesting events
  - Acquisition method adjusted
  - Supervised or autonomous

# Outline

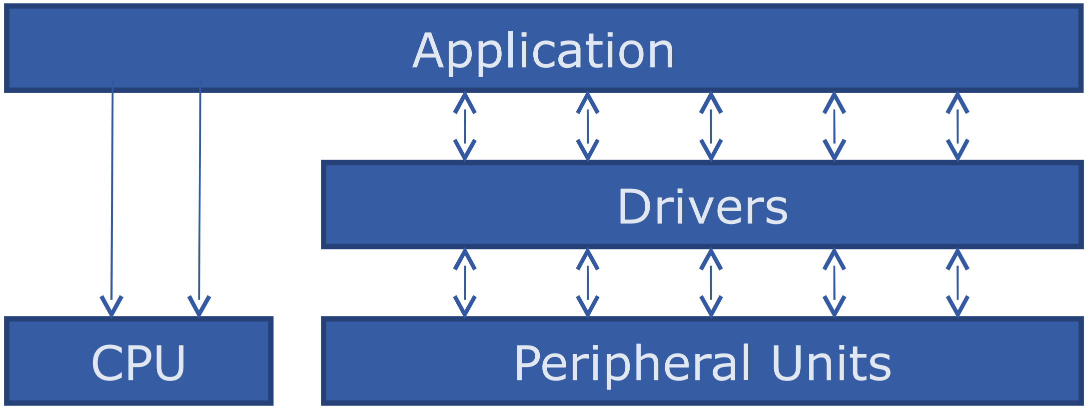
- Introduction
  - Wireless Sensor Networks
  - Scientific observations
  - Engineering Challenges
- Lessons from Hogthrob
- Characterizing Mote Performance
  - [EWSN08] Leopold, Chang, Bonnet
- Perspectives
  - The MANA project

# What if we were to use a SoC mote?

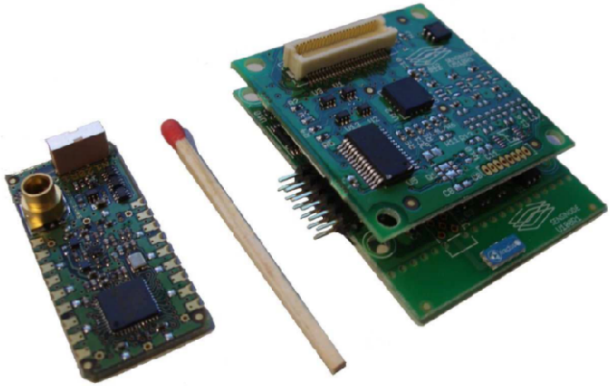
- Problem: How to navigate the design space?
  - Which mote to pick? Can we use a CC2430 based Nano mote instead of the MSP based Micro?
- Approach [Seltzer et al. 1999]
  - Programs are a linear combination of primitive operations of fixed execution time
  - Program performance expressed as the scalar product of a platform vector and an application vector
  - Our hyptothesis in Sensor Networks:
    - Energy = $\underline{MV} \cdot \underline{AV}$

# Vector-based Approach

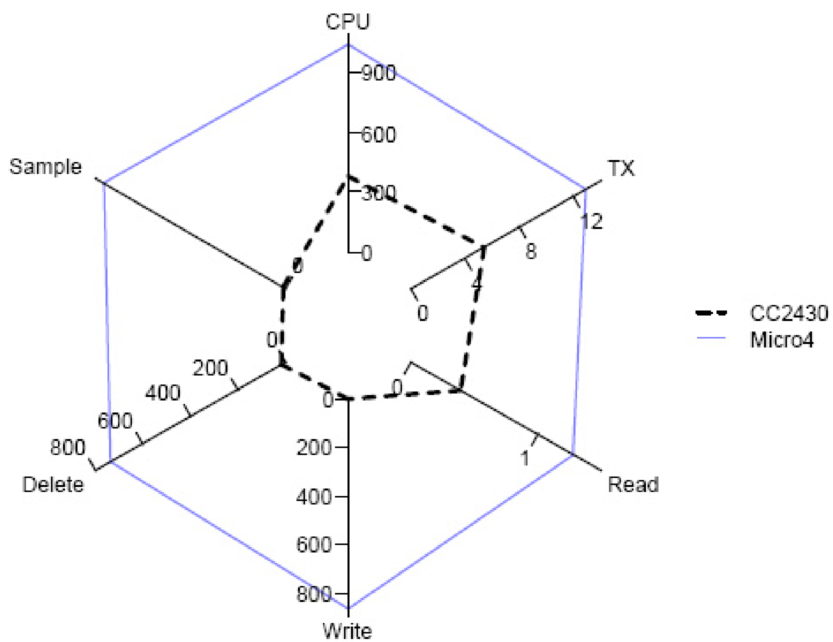
- Application implemented on top of generic drivers (HIL)
- Primitive operations defined for:
  - CPU
  - Peripheral units
- Mote vector:
  - Cost of primitive operations obtained via micro-benchmark
- Application vector:
  - Mote-independant characterization of the application in terms of utilizatoin of the basic operations



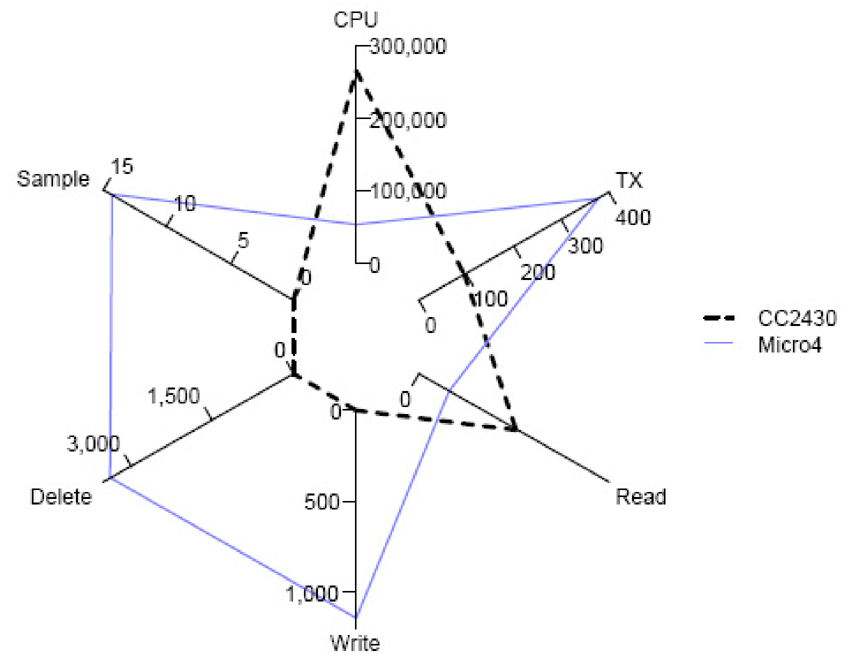
# Mote Vectors



Micro: Sensinode Micro.4 (MSP430) vs
Nano: Chipcon CC2430 (8051)



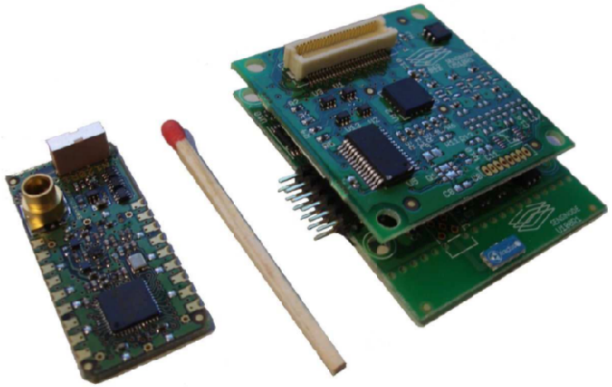

(a) Time Mote Vectors (ms)

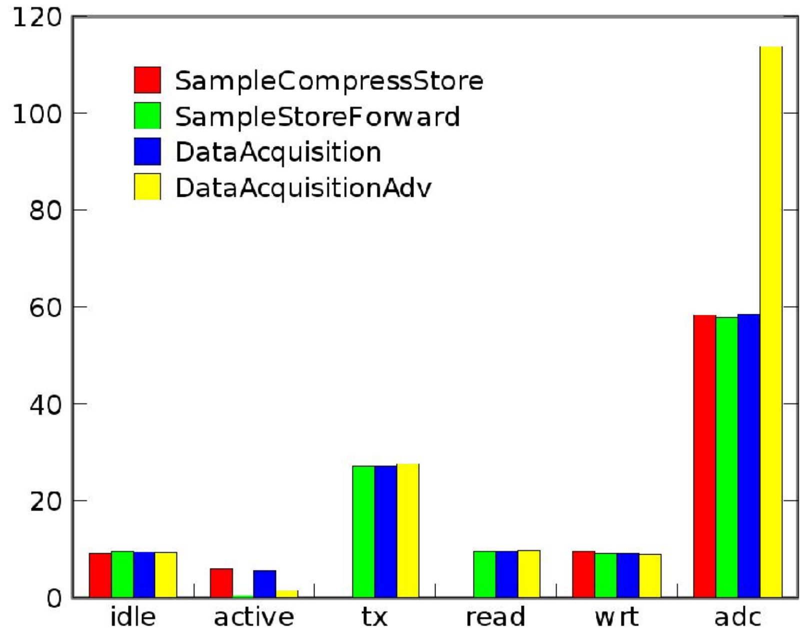



(b) Energy Mote Vectors (mC)

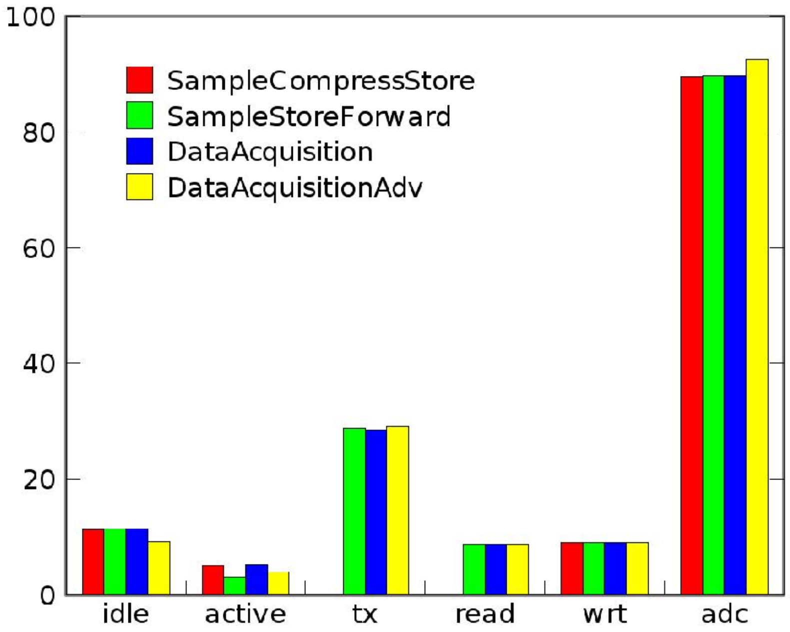
http://www.tinyos8051wg.ne

# Application Vectors



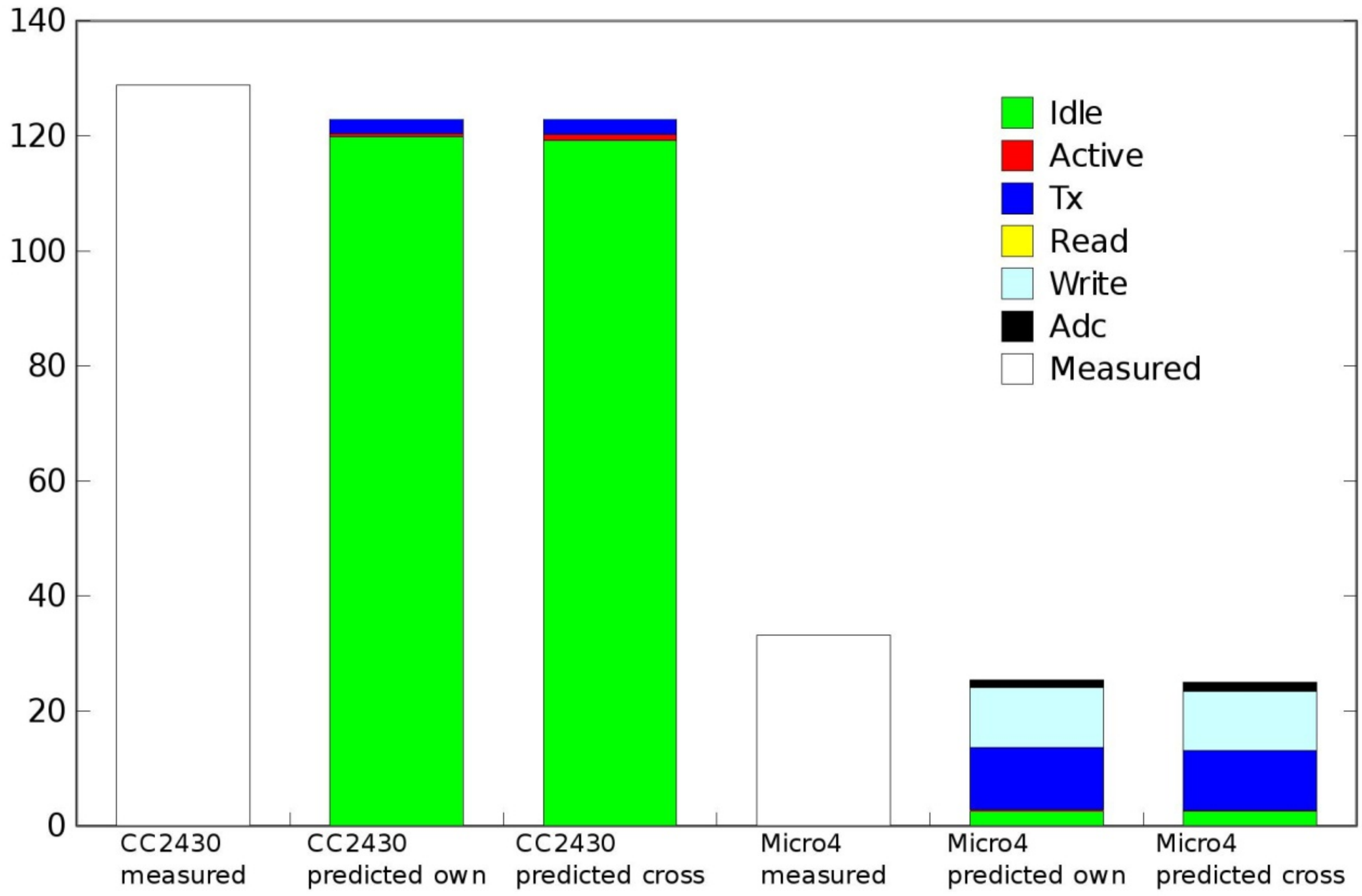



CC2430




Micro.4

# Performance Prediction



# Characterizing Mote Performance

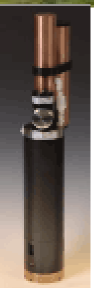
- Benchmarking methodology tailored for measuring the speed and energy consumption of motes
  - Enables direct comparison of motes
- A platform independant application description
  - Usable for debugging, and characterization
- Cross-platform energy prediction
  - Predict the energy consumption of an application when run on a new platform
- Method based on experimental data and not simulation
  - Data can be collected either in the lab or in the field

# Outline

- Introduction
  - Wireless Sensor Networks
  - Scientific observations
  - Engineering Challenges
- Lessons from Hogthrob
- Characterizing Mote Performance
- Perspectives
  - The MANA project
    - [SAON08] Bonnet, Chang, Christoffersen

# Perspectives

- Generic modules emerging
  - Arch Rock PhyNet (6LoWPAN)
- Autonomous System to control data collection
  - Corrrelate data across space/time/modalities
    - Detect measurement errors / interesting events
  - Adjust sampling rate based on trade-off between scientific value of measurements and energy consumption
  - Deal with measurement errors and failures
  - Mission Planning as programming abstraction



# MANA

- Partners: RU, Arch Rock, KU FBL, DIKU, Dan-system.
- Lake monitoring at Zackenberg
- First deployment in August 08
  - Test buoy since March
- Components
  - AR PhyNet
  - Autonomous data acquisition engine
    - Interesting events / measurement errors

http://mana.escience.dk/

Funded by the Strategic Research Council

# 1. Mote Vectors

Runnning example:

- Mote with ADC and only transmitting data: Active, Sleep, ADC, Tx

$$\underline{MV_t} = \begin{bmatrix} t_{active} \\ t_{idle} \\ t_{adc} \\ t_{tx} \end{bmatrix}$$

# 2. Measure Mote Vectors

- Timing: $\underline{MV}_t$
  - Run a specific application for each primitive operation
  - Collect trace for each application
- Energy: $\underline{MV}_e$
  - Measure power consumption of each operation
  - Problem: CPU contribution is measured as well

# 3. Collect Application Trace

Application State:

- Binary encoding of primitive operations running
  e.g. *active* = 1000, *idle*=0100, *adc*=0010, *tx*=0001

Trace time spent in each application state:

- A *m*-dim. mote vector reveals a $2^m$-dim. trace vector $\underline{T}$

$$\underline{T} = \begin{bmatrix} active & & & & \\ idle & & & & \\ adc & & & & \\ tx & & & & \\ adc & \& & tx & & \\ active & \& & adc & & \\ active & \& & tx & & \\ active & \& & adc & \& & tx \\ & & \dots & & \end{bmatrix}$$

# 4. Calculate Application Vector

$$\underline{AV} = \begin{bmatrix} active \\ idle \\ adc \\ tx \end{bmatrix}$$

Application Vector:

- Number of calls to each operation

Problems converting trace vector $\underline{T}$ to application vector $\underline{AV}$:

- Over counting:
  1. Shared resources: Parallel vs. sequential
  2. Driver CPU usage is counted twice: Trace vs. mote vectors

- Solution:
  1. Architecture Matrix, AM
  2. CPU Matrix, CPU

$$\underline{AV} = (\mathbf{CPU} \times (\mathbf{AM} \times \underline{T})) \circ \underline{MV}^{-1}$$

# 4. Architecture Matrix

Encode hardware congestion into an Architecture Matrix, AM:

- Row: Primitive operation
- Col: Application state
- Collapses the $2^m$-dim. trace vector to a m-dim. vector
- Result: Vector containing raw total time in each operation

$$\underline{AV} = (\mathbf{CPU} \times (\mathbf{AM} \times \underline{T})) \circ \underline{MV^{-1}}$$

# 4. Architecture Matrix (cont'd)

Example:

- ADC and radio share the same bus
- Divide time in state: *adc & tx* evenly between *adc* and *tx*

$$\mathbf{AM} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \vdots \\ 0 & 0 & 1 & 0 & \frac{1}{2} & 1 & 0 & \frac{1}{2} & \\ 0 & 0 & 0 & 1 & \frac{1}{2} & 0 & 1 & \frac{1}{2} & \end{bmatrix} \qquad \underline{T} = \begin{bmatrix} active \\ idle \\ adc \\ tx \\ adc \ \& \ tx \\ active \ \& \ adc \\ active \ \& \ tx \\ active \ \& \ adc \ \& \ tx \\ \cdots \end{bmatrix}$$

$$\underline{AV} = (\mathbf{CPU} \times (\mathbf{AM} \times \underline{T})) \circ \underline{MV^{-1}}$$

# 4. CPU Matrix

Differentiate between application and driver CPU usage:

- Remove double counting of drivers' CPU usage
- CPU Active then corresponds solely to the pure computation

$$\begin{bmatrix} 1 & 0 & -\frac{CPU[adc]}{MV_t[adc]} & -\frac{CPU[tx]}{MV_t[tx]} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \times \begin{bmatrix} t_{active} \\ t_{idle} \\ t_{adc} \\ t_{tx} \end{bmatrix}$$

$$\underline{AV} = (\mathbf{CPU} \times (\mathbf{AM} \times \underline{T})) \circ \underline{MV^{-1}}$$

# Application Vector

Final step:

- Divide total time of each operation with the unit time

$$\underline{AV} = (\mathbf{CPU} \times (\mathbf{AM} \times \underline{T})) \circ \underline{MV^{-1}}$$

- The application vector contains the number of times each primitive operation has been called.

# 5. Performance Prediction

- Application Vector is platform independent
- Performance prediction can be done with Mote Vectors

- Cross-platform predictions

$$Energy = \underline{MV_e} \cdot \underline{AV}$$

$$Execution\ time = \underline{MV_t} \cdot \underline{AV}$$