


# Data Gathering


## Chapter 4




ETH  
Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

### The PermaSense Project

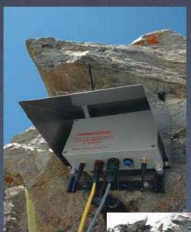
#### Matterhorn Field Site Installations



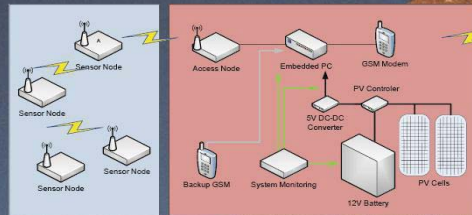
TIK  
Computer Engineering and  
Networks Laboratory




Sensor node installations targeting 3 years unattended lifetime




Base station mounted under a combined sun/rain hood





Base station and solar panels on the field site at Matterhorn



Base station power supply, system monitoring and a backup GSM modem are housed separately

## Rating

- Area maturity

First steps

Text book

- Practical importance

No apps

Mission critical

- Theory appeal

Booooooring

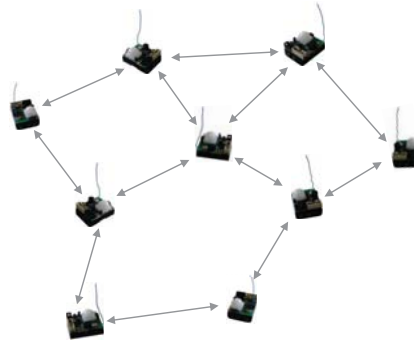
Exciting

## Overview

- Motivation
- Data gathering
  - Max, Min, Average, Median, ...
- Universal data gathering tree
- Energy-efficient data gathering: Dozer

## Sensor networks

- Sensor nodes
  - Processor & memory
  - Short-range radio
  - **Battery powered**
- Requirements
  - Monitoring geographic region
  - Unattended operation
  - **Long lifetime**



What kind of traffic patterns may occur in a sensor network?



## Data Gathering

- Different traffic demands require different solutions
- **Continuous** data collection
  - Every node sends a sensor reading once every two minutes
- Database-like network **queries**
  - “Which sensors measure a temperature higher than 21°C?”
- **Event** notifications
  - A sensor sends an emergency message in case of fire detection.



## Sensor Network as a Database

- Use paradigms familiar from relational databases to simplify the “programming” interface for the application developer.

```
SELECT roomno, AVERAGE(light), AVERAGE(volume)
FROM sensors
GROUP BY roomno
HAVING AVERAGE(light) > l AND AVERAGE(volume) > v
EPOCH DURATION 5min
```

- TinyDB is a service that supports SQL-like queries on a sensor network.
  - Flooding/echo communication
  - Uses in-network aggregation to speed up result propagation.

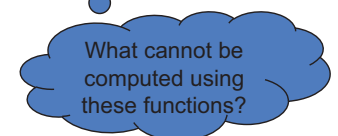
```
SELECT <aggregates>, <attributes>
[FROM {sensors | <buffer>}]
[WHERE <predicates>]
[GROUP BY <exprs>]
[SAMPLE PERIOD <const> | ONCE]
[INTO <buffer>]
[TRIGGER ACTION <command>]
```

## Distributed Aggregation

- Growing interest in **distributed aggregation**
  - Sensor networks, distributed databases...
- Aggregation functions?
  - **Distributive** (max, min, sum, count)
  - **Algebraic** (plus, minus, average)
  - **Holistic** (median, k<sup>th</sup> smallest/largest value)
- **Combinations** of these functions enable complex queries.
  - „What is the average of the 10% largest values?“



What cannot be computed using these functions?

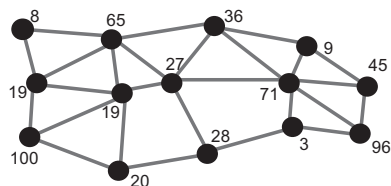


## Aggregation Model

- How difficult is it to compute these aggregation primitives?
- Model:
  - All nodes hold a **single element**.
  - A **spanning tree** is available
    - Shortest path tree (SPT), all nodes on shortest path to sink, radius  $D$
  - Messages can only contain 1 or 2 elements.

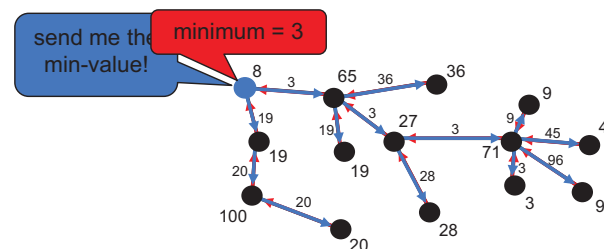
Can be generalized to an arbitrary number of elements!

$O(1)$



## Computing the Minimum Value...

- Use a simple *flooding-echo* procedure → **convergecast**



- Time complexity:  $\Theta(D)$
- Number of messages:  $\Theta(n)$

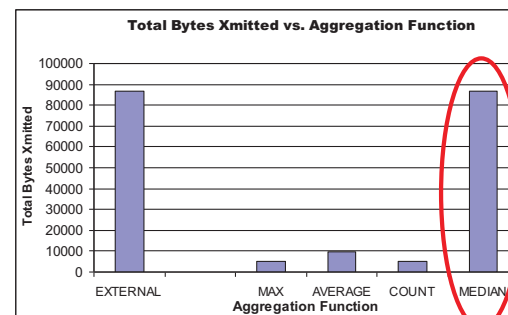
## Distributive & Algebraic Functions

How do you compute the sum of all values?  
 ... what about the average?  
 ... what about a random value?  
 ... or even the median?



## Holistic Functions

- It is widely believed that holistic functions are **hard** to compute using in-network aggregation.
  - Example: **TAG** is an aggregation service for sensor networks. It is fast for other aggregates, but not for the **MEDIAN** aggregate.



„Thus, we have shown that (...) in network aggregation can reduce communication costs by an order of magnitude over centralized approaches, and that, even in the worst case (such as with **MEDIAN**), it provides **performance equal to the centralized approach**.“

TAG simulation: 2500 nodes in a 50x50 grid

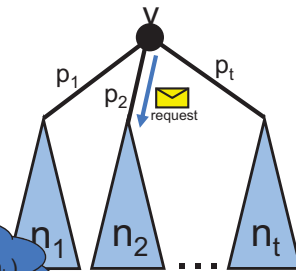
## Randomized Algorithm

- Choosing elements **uniformly at random** is a good idea...
  - How is this done?

- Assuming that all nodes know the sizes  $n_1, \dots, n_t$  of the subtrees rooted at their children  $v_1, \dots, v_t$ , the request is forwarded to node  $v_i$  with probability:

$$p_i := n_i / (1 + \sum_k n_k)$$

With probability  $1 / (1 + \sum_k n_k)$  node  $v$  chooses itself.



- Key observation: Choosing an element randomly **requires  $O(D)$  time!**
  - Use pipe-lining to select **several random elements!**

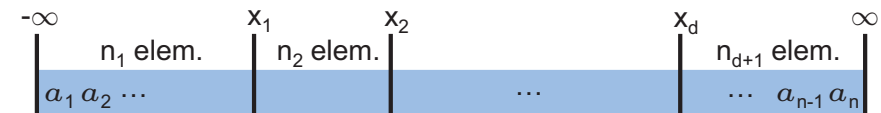
$D$  elements in  $O(D)$  time!

## Randomized Algorithm

- The algorithm operates in phases
  - A **candidate** is a node whose element is possibly the solution.
  - The set of **candidates** decreases in each phase.
- A phase of the randomized algorithm:

- Count the **number of candidates** in all subtrees
- Pick  **$O(D)$  elements**  $x_1, \dots, x_d$  uniformly at random
- For all those elements, count the **number of smaller elements!**

Each step can be performed in  $O(D)$  time!



## Randomized Algorithm

- Using these counts, the **number of candidates** can be reduced by a factor of  $D$  in a constant number of phases with high probability.

➔ The time complexity is  $O(D \cdot \log_D n)$  w.h.p.

With probability at least  $1 - 1/n^c$  for a constant  $c \geq 1$ .

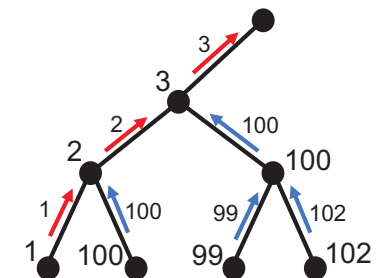
- It can be shown that  $\Omega(D \cdot \log_D n)$  is a lower bound for distributed  $k$ -selection (finding the  $k^{\text{th}}$  smallest element).
  - This simple randomized algorithm is asymptotically optimal.
- The only remaining question: Is randomization needed, or, what can we do **deterministically?**



## Deterministic Algorithm

- Why is it difficult to find a good deterministic algorithm?
  - Finding a good selection of elements that provably reduces the set of candidates is hard.
- Idea: Always propagate the median of all received values.
- Problem: In one phase, only the  **$h^{\text{th}}$  smallest element** is found if  $h$  is the height of the tree...
  - Time complexity:  $O(n/h)$

One could do a lot better!!! (Not shown in this course.)



## Median Summary

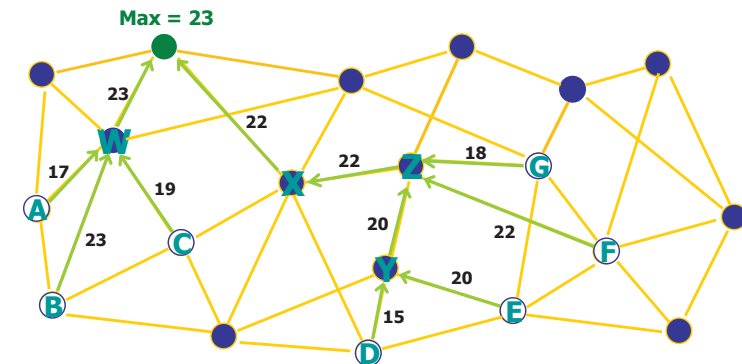
- Simple randomized algorithm with time complexity  $O(D \cdot \log_D n)$  w.h.p.
  - Easy to understand, easy to implement...
  - Asymptotically optimal. Lower bound shows that no algorithm can be significantly faster.
- Deterministic algorithm with time complexity  $O(D \cdot \log_D^2 n)$ .
  - If  $\exists c \leq 1: D = n^c$ , k-selection can be solved efficiently in  $\Theta(D)$  time even deterministically.



Recall the 50x50 grid used to evaluate TAG

## Sensor Network as a Database

- We do not always require information from all sensor nodes.
  - `SELECT MAX(temp) FROM sensors WHERE node_id < "H".`

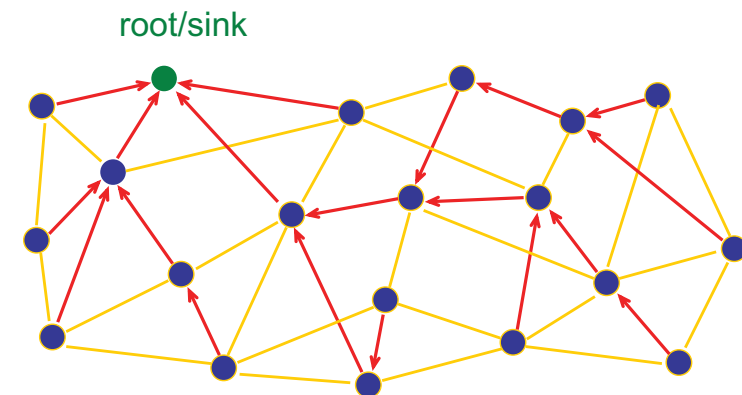


## Selective data aggregation

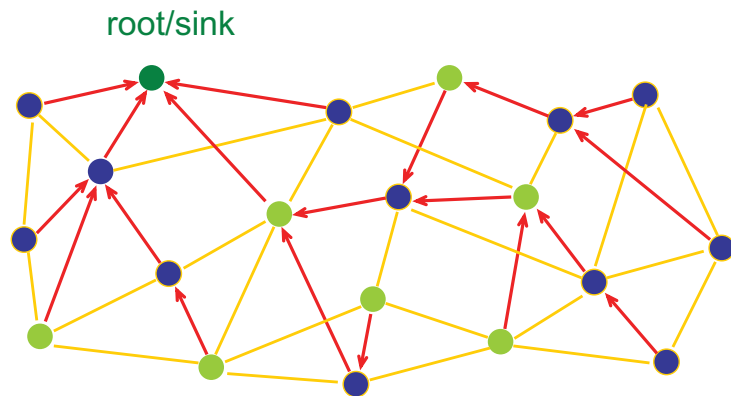
- In sensor network applications
  - Queries can be frequent
  - **Sensor groups are time-varying**
  - Events happen in a dynamic fashion
- Option 1: Construct aggregation trees for each group
  - Setting up a good tree incurs communication overhead
- Option 2: Construct a single spanning tree
  - When given a sensor group, simply use the **induced tree**
  - In other words, cut all the branches that are not used

## Example

- The **red** tree is the universal spanning tree. All links cost 1.

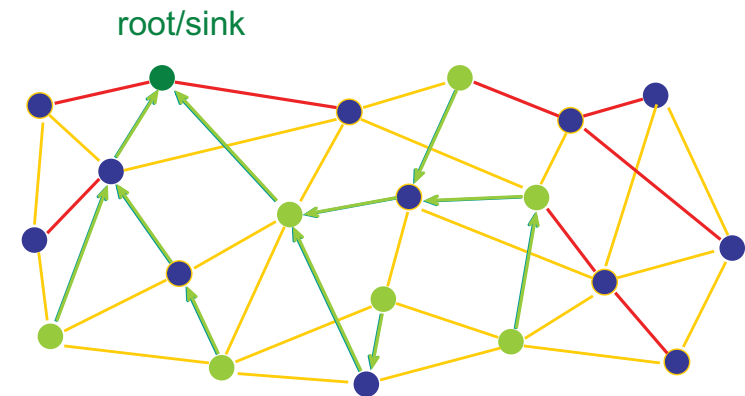


## Given the lime subset...



## Induced Subtree

- The cost of the induced subtree for this set S is 11. The optimal is 8.

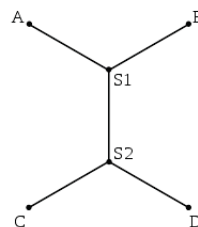


## Group-Independent (Universal) Spanning Tree Problem

- Given
  - A set of nodes  $V$  in the Euclidean plane (or in a metric space)
  - A root node  $r \in V$
  - Define stretch of a **universal spanning tree**  $T$  to be

$$\max_{S \subseteq V} \frac{\text{cost}(\text{induced tree of } S+r \text{ on } T)}{\text{cost}(\text{minimum Steiner tree of } S+r)}$$

- We're looking for a spanning tree  $T$  on  $V$  with minimum stretch.
- Remark: A **Steiner tree** for a set of nodes  $S$  is like a MST, except that it may use nodes and edges outside  $S$  to help.
  - Example: Steiner Tree for nodes  $A, B, C, D$ , with potentially all points in the plane helping



## Main results

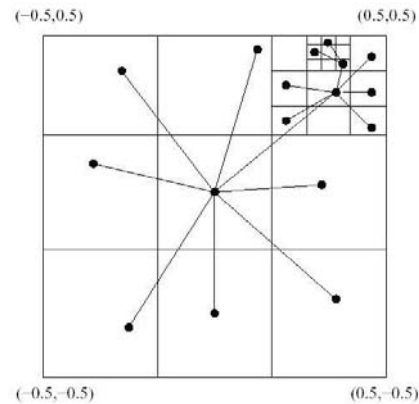
- Upper bound:** For the minimum UST problem in Euclidean plane, with edge cost being distance, an approximation of  $O(\log n)$  can be achieved.
- Lower bound:** No polynomial time algorithm can approximate the minimum UST problem with stretch better than  $\Omega(\log n / \log \log n)$ .
- [Jia, Lin, Noubir, Rajaraman and Sundaram, STOC 2005]
- Question: Why are MST or SPT not good as UST?
  - Again, nodes in the plane, cost Euclidean distance



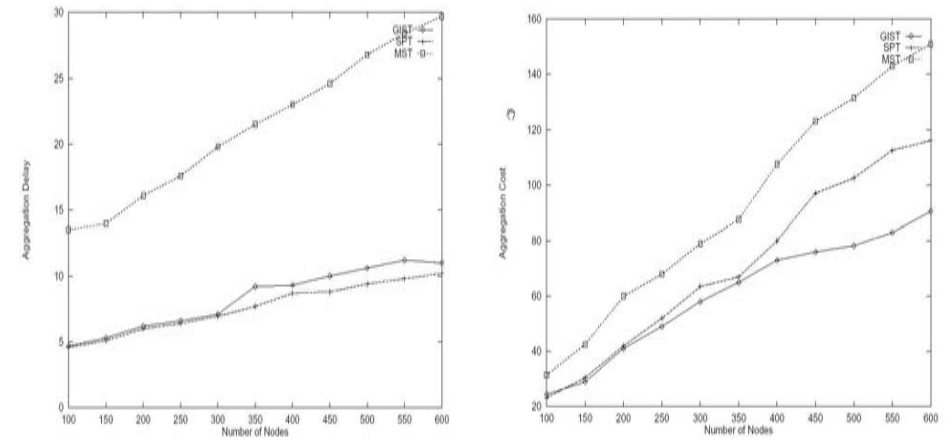
## Algorithm sketch

- For the simplest Euclidean case:
- Recursively divide the plane and select random node.

- Results: The induced tree has logarithmic overhead. The aggregation delay is also constant.



## Simulation with random node distribution & random events



## Continuous Data Gathering



- Long-term measurements
- Unattended operation
- Low data rates
- Battery powered
- ~~Network latency~~
- ~~Dynamic bandwidth demands~~

Energy conservation is crucial to prolong network lifetime

## Energy-Efficient Protocol Design

- Communication subsystem is the main energy consumer
  - Power down radio as much as possible

TinyNode	Power Consumption
uC sleep, radio off	0.015 mW
Radio idle, RX, TX	30 – 40 mW



- Issue is tackled at various layers
  - MAC
  - Topology control / clustering
  - Routing

➔ Orchestration of the whole network stack to achieve radio duty cycles of ~1%

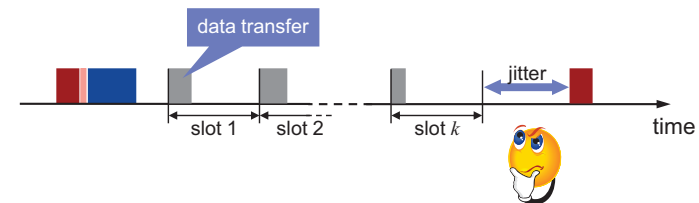
## Dozer System

- Tree based routing towards data sink
  - No energy wastage due to multiple paths
  - Current strategy: Shortest Path Tree
- “TDMA based” link scheduling
  - Each node has two independent schedules
  - No global time synchronization
- The parent initiates each TDMA round with a beacon
  - Enables integration of disconnected nodes
  - Children tune in to their parent’s schedule



## Dozer System

- Parent decides on its children data upload times
  - Each interval is divided into upload slots of equal length
  - Upon connecting each child gets its own slot
  - Data transmissions are always acknowledged
- No traditional MAC layer
  - Transmissions happen at exactly predetermined point in time
  - Collisions are explicitly accepted
  - Random jitter resolves schedule collisions



## Dozer System

- Lightweight backchannel
  - Beacon messages comprise commands
- Bootstrap
  - Scan for a full interval
  - Suspend mode during network downtime
- Potential parents
  - Avoid costly bootstrap mode on link failure
  - Periodically refresh the list

periodic channel activity check



## Dozer System

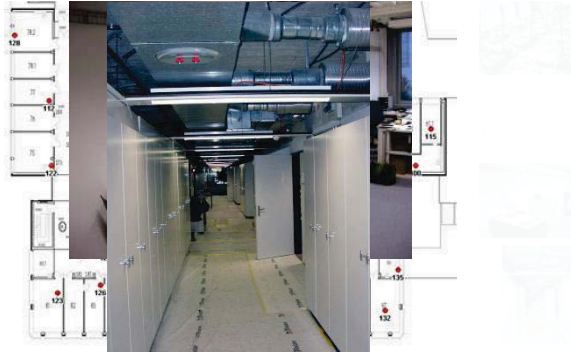
- Clock drift compensation
  - Dynamic adaptation to clock drift of the parent node
- Application scheduling
  - Make sure no computation is blocking the network stack
  - TDMA is highly time critical
- Queuing strategy
  - Fixed size buffers



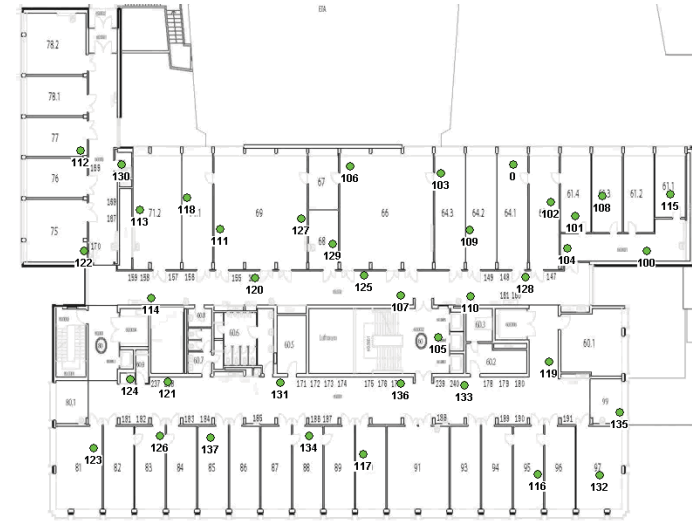


## Evaluation

- Platform
  - TinyNode
    - MSP 430
    - Semtech XE1205
  - TinyOS 1.x
- Testbed
  - 40 Nodes
  - Indoor deployment
  - > 1 month uptime
  - 30 sec beacon interval
  - 2 min data sampling interval

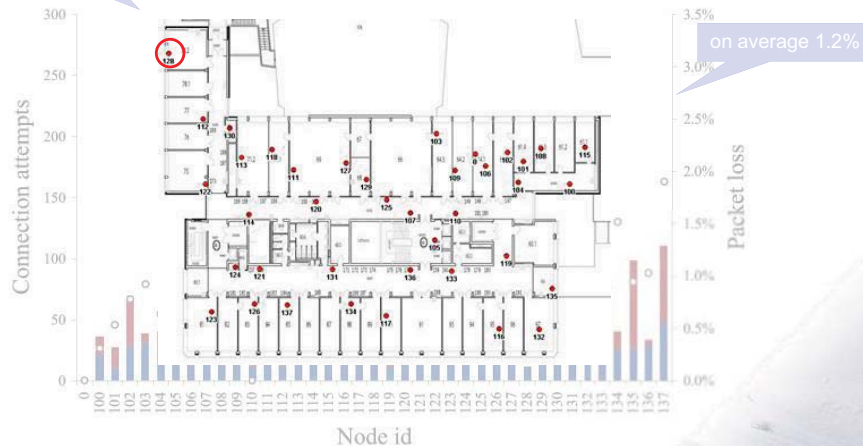


## Dozer in Action



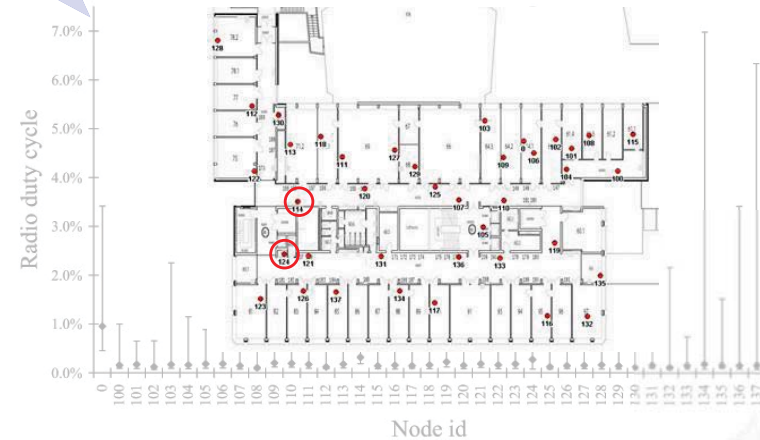
## Tree Maintenance

1 week of operation



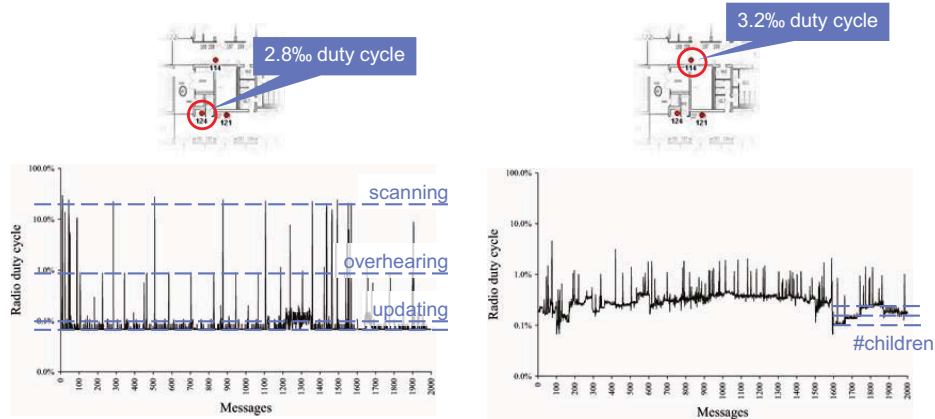
## Energy Consumption

on average 1.67%



➔ Mean energy consumption of 0.082 mW

## Energy Consumption



- Leaf node
- Few neighbors
- Short disruptions

- Relay node
- No scanning

## More than one sink?

- Use the **anycast** approach and send to the closest sink.
- In the simplest case, a source wants to minimize the number of hops. To make anycast work, we only need to implement the regular distance-vector routing algorithm.
- However, one can imagine more complicated schemes where e.g. sink load is balanced, or even intermediate load is balanced.

## Dozer Conclusions & Possible Future Work

- Conclusions
  - Dozer achieves duty cycles in the magnitude of 1%.
  - Abandoning collision avoidance was the right thing to do.
- Possible Future work
  - Optimize delivery latency of sampled sensor data.
  - Make use of multiple frequencies to further reduce collisions.

## Open problem

- Continuous data gathering is somewhat well understood, both practically and theoretically, in contrast to the two other paradigms, event detection and query processing.
- One possible open question is about **event detection**. Assume that you have a battery-operated sensor network, both sensing and having your radio turned on costs energy. How can you build a network that raises an alarm quickly if some large-scale event (many nodes will notice the event if sensors are turned on) happens? What if nodes often sense false positives (nodes often sense something even if there is no large-scale event)?