Motes vs. Traditional Computing

- Lossy, Adhoc Radio Communication
  - Low Bandwidth Shared Radio Channel
  - Very lossy: 30% base loss rate
  - Somewhat different communication patterns
    - Multi-hop communication
    - Broadcast communication

- Sensing Hardware
  - “Acquisitional”: can control when/what to sample

- Severe Power Constraints
  - Lasts only days to weeks depending on usage
  - Communication perhaps most expensive
Data Acquisition in Sensor Networks

- Previously TinyOS-based (or other OSes)
  - Long-running, customized programs
    - In many cases, involve aggregation/fusion
  - Written in C? (maybe Java)
  - Hard to adjust, modify
  - Must deal with:
    - Limited power budget (e.g. adjust sampling rates if power goes down)
    - Lossy communication
    - Distributed algorithms-related issues
    - Limited development tools

- Data collection should be more central to WSNs!
TAG/TinyDB

- TAG: Tiny Aggregation Service; evolved into TinyDB
- Simple, declarative interface for data collection and aggregation
  - Based on SQL
- Abstraction layer that hides the details of the WSNs
  - Query distribution, lossy communication etc. . .
- Advantages:
  - Simple, declarative language; Energy-efficient
  - Abstract away the details:
    - “independence” from “sensor network details”
    - Can transparently apply new optimizations
    - Can change/modify the physical hardware without affecting applications
    - Very critical step in making things more usable
Routing

- Using a tree constructed by broadcasting a message

Declarative query language

Example:

```
SELECT AVG(volume), room FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(volume) > threshold
EPOCH DURATION 30s
```

General form:

```
SELECT \{agg(expr), attrs\} FROM sensors
WHERE \{selPreds\}
GROUP BY \{attrs\}
HAVING \{havingPreds\}
EPOCH DURATION i
```
Declarative query language
   Only single relation queries over a virtual table
   Attributes:
   ⟨ time, sensorid, temp, humidity, light, . . . , power level, list-of-neighbors, . . . ⟩
   Each node maintains a local catalog; basestation maintains global catalog
   Heterogenous catalogs at sensor nodes okay
   Query result is a “stream”
   Epoch duration must be at least 30ms. Why ?
The query language doesn’t support “hardware events” easily.

From: “Towards Sensor Database Systems”; Bonnet, Gehrke, Seshadri;

A temperature sensor might support two functions directly in hardware:

- `getTemperature()`
- `detectAlarmTemperature(threshold)`

How to write a query that returns all events?

- Must use polling or implement in software

TinyDB supported this later.
Three functions:

- initialize: Initialize a partial state record
- merge: Merge two partial state records
- evaluate: Evaluate the final result

Quite similar to how relational DB do it

- PostgreSQL user-defined aggregates require implementing similar set of functions

Given this:

- Each node *initializes* a PSR from its observed value
- Each node *merges* its PSR with its children’s PSRs
- Each node sends the PSR to parent
- The basestation *evaluates*
Remember: Goal is to push as much processing in the network as possible

Understanding the behavior of the aggregates is the key

Dimensions:
- Duplicate sensitivity, exemplary vs summary, monotonic
- Distributive vs algebraic vs holistic vs unique vs content-sensitive
Duplicate-insensitive: Can be made more robust (by sending multiple copies etc)

Summary: Can use sampling

Monotonic: Some optimizations based on the actual values
  - Can use snooping to avoid sending messages
  - “Having” can be pushed down in the network in some cases

Distributive vs . . .
  - Better understanding of the memory/communication requirements
Query distribution

- Main issue: setting the delivery times

Figure 1: Partial state records flowing up the tree during an epoch.
Full pipelining also possible.

Grouping

*Partial pre-aggregation*
  - Technique developed elsewhere
  - Can be used to deal with memory constraints
Broadcast communication: snooping to avoid sending messages

Hypothesis testing
- Send data into the network instead of out of it
- An extreme version:
  - “Collecting Correlated Information from a Sensor Network”; Micah Adler; SODA 2005
  - Base station essentially asks questions
  - Sensor nodes communicate very few bits

Improving tolerance
- Through child caches
  - Can use more fancy predictive models instead
  - e.g. Auto-regressive models
- Send multiple copies
  - Works very well for duplicate-insensitive aggregates
  - So MIN, MAX ??? Is that it ?
  - Approximate **DISTINCT COUNT** can be done as well..
**DISTINCT COUNT using FM-Sketches**

- Flajolet-Martin Sketch: Count distinct number of values in a sequence in *one pass* with minimum memory
  - An example of a “frequency moment”
  - General case: Alon, Matias, Szegedy; STOC 1996.

- $N =$ Length of the sequence
- $n =$ Number of distinct values

**Naive Approach:**
- Keep a list of all distinct values, and update incrementally
- $O(n)$

**FM-Sketches:** Approximate counting in $O(\log(n))$ space
DISTINCT COUNT using FM-Sketches

Algorithm:

- Use a bitmap, $B$, of size $k$, where $k$ is $\approx \theta(\log_2(n))$
  - Aren’t we trying to estimate $n$?
  - Use a rough upper bound. Even if you overestimate by a factor of 4, you only use 2 more bits.
Algorithm:

- Use a bitmap, $B$, of size $k$, where $k \approx \theta(\log_2(n))$
  - Aren’t we trying to estimate $n$?
  - Use a rough upper bound. Even if you overestimate by a factor of 4, you only use 2 more bits.
- Need a uniform hash function: $h(x)$ maps values in the sequence to $\{0, \cdots, 2^k - 1\}$.
- For each value, $v$ in the sequence, find $h(v)$. 
**Algorithm:**

- Use a bitmap, $B$, of size $k$, where $k \approx \theta(\log_2(n))$
  - Aren’t we trying to estimate $n$?
  - Use a rough upper bound. Even if you overestimate by a factor of 4, you only use 2 more bits.

- Need a uniform hash function: $h(x)$ maps values in the sequence to $\{0, \cdots, 2^k - 1\}$.
- For each value, $v$ in the sequence, find $h(v)$.
- Let $l(h(v))$ denote the least-significant 1 bit in $h(v)$.
  - $k = 6, h(v) = 000100$, then $l(v) = 3$.
  - $k = 6, h(v) = 000101$, then $l(v) = 1$. 
Algorithm:

- Use a bitmap, $B$, of size $k$, where $k \approx \theta(\log_2(n))$.
  - Aren’t we trying to estimate $n$?
  - Use a rough upper bound. Even if you overestimate by a factor of 4, you only use 2 more bits.

- Need a uniform hash function: $h(x)$ maps values in the sequence to $\{0, \cdots, 2^k - 1\}$.

- For each value, $v$ in the sequence, find $h(v)$.

- Let $l(h(v))$ denote the least-significant 1 bit in $h(v)$.
  - $k = 6$, $h(v) = 000100$, then $l(v) = 3$.
  - $k = 6$, $h(v) = 000101$, then $l(v) = 1$.

- Set $B(l(v)) = 1$.

- Note: Duplicate values will just set the same bit again: “duplicate-insensitive”
Algorithm (Cntd):
- At the end, let $c$ be the least-significant (right-most) 0 in $B$
- $1.2928 \times 2^c$ is an estimator for the number of distinct values
Algorithm (Cntd):

- At the end, let $c$ be the least-significant (right-most) 0 in $B$.
- $1.2928 \times 2^c$ is an estimator for the number of distinct values.
- Why?
  - Choose a number, $x$, uniformly between 0 to $2^k - 1$.
  - $\text{Prob}(l(x) = c) = 1/2^{c+1}$
  - Hash function is assumed to map values in the sequence uniformly onto the above range as well.
DISTINCT COUNT using FM-Sketches

Algorithm (Cndt):
- At the end, let $c$ be the least-significant (right-most) 0 in $B$
- $1.2928 \times 2^c$ is an estimator for the number of distinct values
- Why?
  - Choose a number, $x$, uniformly between 0 to $2^k - 1$.
  - $\text{Prob}(l(x) = c) = 1 / 2^{c+1}$
  - Hash function is assumed to map values in the sequence uniformly onto the above range as well
- Use multiple hash functions for more confidence
- Space: $O(\log(n))$
- Choosing hash functions?
  - Tricky: uniform hash functions take a lot of space
  - Much work on relaxing the requirement
Duplicate-insensitive aggregates

Can use FM-Sketches directly to find DISTINCT COUNT in sensor networks

Reading:

- S. Nath, P. B. Gibbons, S. Seshan and Z. Anderson; Synopsis Diffusion for Robust Aggregation in Sensor Networks; SenSys’04

More work since then