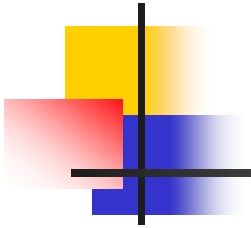
Information Management I: Sensor Database, Querying, Publish & Subscribe, Information Summarization + SIGMOD 2003 paper

CS428: Information Processing for Sensor Networks, and STREAM meeting

Presented by Itaru Nishizawa (Hitachi, Ltd.)
nishizawa@home.email.ne.jp

May 16, 2003



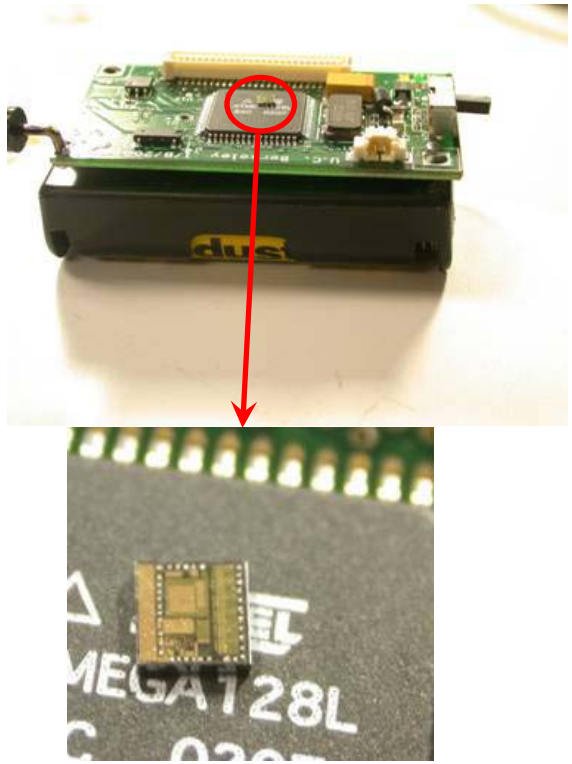
Papers

1. [madden02b]:
   S. Madden, M. Franklin, J. Hellerstein, and W. Hong: "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks", In Proc. of the 5th Annual Symposium on Operating Systems Design and Implementation (OSDI 2002).
2. [hellerstein03]:
   J. Hellerstein, W. Hong, S. Madden, and K Stanek: "Beyond Average: Towards Sophisticated Sensing with Queries", In Proc. of the 2nd Int. Workshop on Information Processing in Sensor Networks (IPSN 03).
3. [madden03a]:
   S. Madden, M. Franklin, J. Hellerstein, and W. Hong: "The Design fo and Acquisitional Query Processor For Sensor Networks", In Proc. of the 22nd ACM International Conference on Management of Data (SIGMOD 2003).

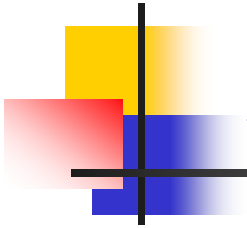
* All papers are from Berkeley & Intel Research.

Paper 1: Background

- Berkeley motes
  - 2cmx4cmx1cm in size
  - Radio, processor, memory, battery pack, and sensors
- TinyOS
  - Ad-hoc networks
  - Device detection
  - Dynamic routing

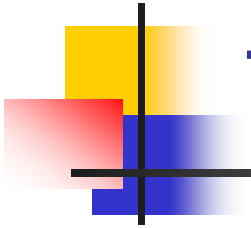


Next Generation's single chip mote: 2mmx2.5mm, http://www.cs.berkeley.edu/~jhill/spec/index.htm



Application and Motivation

- Applications
  - Civil engineers: Building integrity monitoring
  - Biologists: Habitat monitoring
  - Comp. Admins: Data Center monitoring
- Motivation
  - Needs summary rather than raw data
  - Aggregation has to be provided as a *core service* by the system software

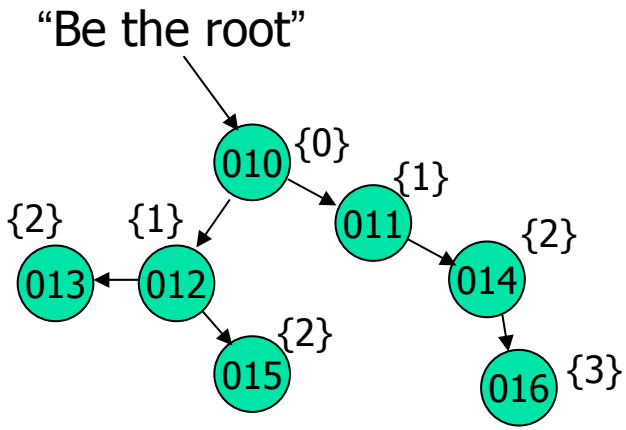


TAG approach

- A simple, declarative query model
  - Like SQL without joins
- Process aggregates in the network
  - To reduce the data flow in the network
- Query processing operations:
  - Users pose queries to a powered basestation
  - Queries are distributed into the network
  - Sensors route data back through a routing tree rooted at the basestation

Ad-Hoc Routing

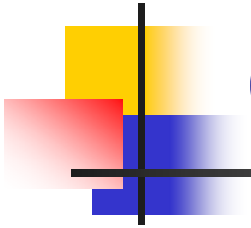
- Tree-based routing scheme



- The root broadcasts a message to organize a routing tree
- The message contains id and level
- Any node without assigned the level set its level and parent if they hear the message

A sensor node id {level}

- Routing messages are periodically broadcast from the root



Query Model and Environment

- Query:
  - SQL-style syntax
  - Query refers single table "`sensors`"
    (i.e. Query doesn't contain join)
- Table and Attributes:
  - `sensors` table is append only
  - Attributes are sensor inputs (e.g. temperature, light)
  - Each mote stores a small catalog of attributes
  - Central query processor stores all attributes

Query Example

- Query in TAG:

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

- Query in English:
  - Reports all rooms on the 6th floor where the average volume is over a specific *threshold*. Updates are delivered every 30 seconds.

TAG Query Semantics

- Same as SQL except for,
  1. EPOCH DURATION clause
  2. Output is stream of values
     <group id, aggregate value>
     Each group is time-stamped
  3. More aggregate functions

     SQL: `COUNT`, `MIN`, `MAX`, `SUM`, and `AVERAGE`

     TAG: `COUNT`, `MAX`, `MIN`, `SUM`, `AVERAGE`, `MEDIAN`, and `HISTGRAM`

Aggregates Taxonomy

	MAX, MIN	COUNT, SUM	AVERAGE	MEDIAN	COUNT DISTINCT	HISTOGRAM
Duplicate Sensitive[1]	No	Yes	Yes	Yes	No	Yes
Exemplary (E), or Summary (S)[2]	E	S	S	E	S	S
Monotonic[3]	Yes	Yes	No	No	Yes	No
Partial State[4]	D	D	A	H	U	C

[1] This property is used in some optimization such as redundant reporting

[2] Exemplary aggregates behave unpredictably in the data loss

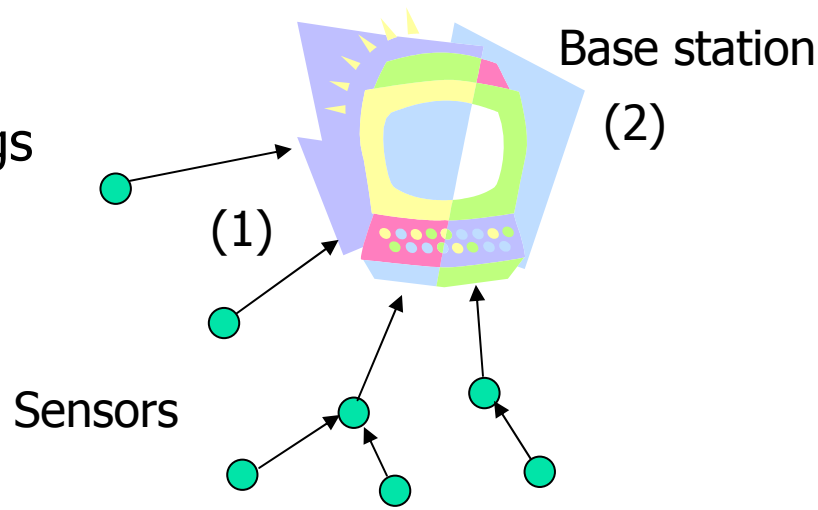
[3] s' is combined partial state record of s1 and s2, then $\forall$s1, s2, e(s') $\geqq$ MAX(e(s1), e(s2)) or $\forall$s1, s2, e(s') $\leqq$ MIN(e(s1), e(s2)).

[4] "Partial State" relates to the amount of state required for each partial state record. D: Distributive (partial state record size = final aggregate record size), A: Algebraic (partial state size = constant), H: Holistic (partial state record size = proportional in size to the set of data), U: Unique (similar to Holistic), C: Content-Sensitive (proportional to some property of the data values)

Network Aggregation - Naïve

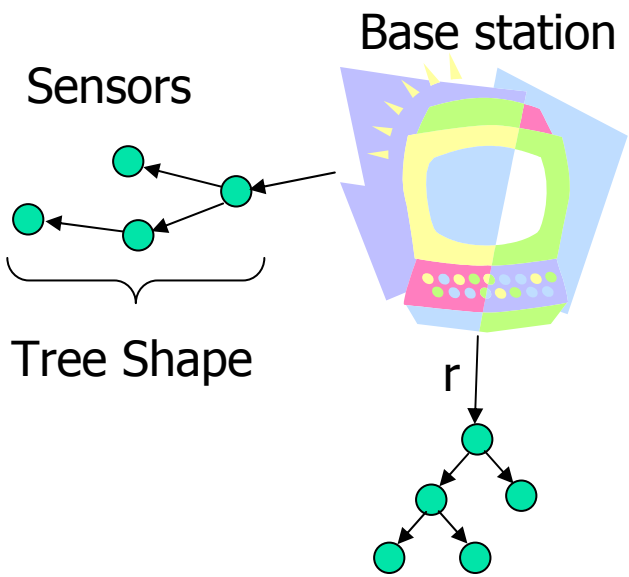
- Naïve implementation (centralized, server-based approach)

(1) Sensors send all the readings to the base station
(2) Base station calculates aggregate values



Network Aggregation - TAG I

- Distribution phase
  - Requests are pushed down into the network

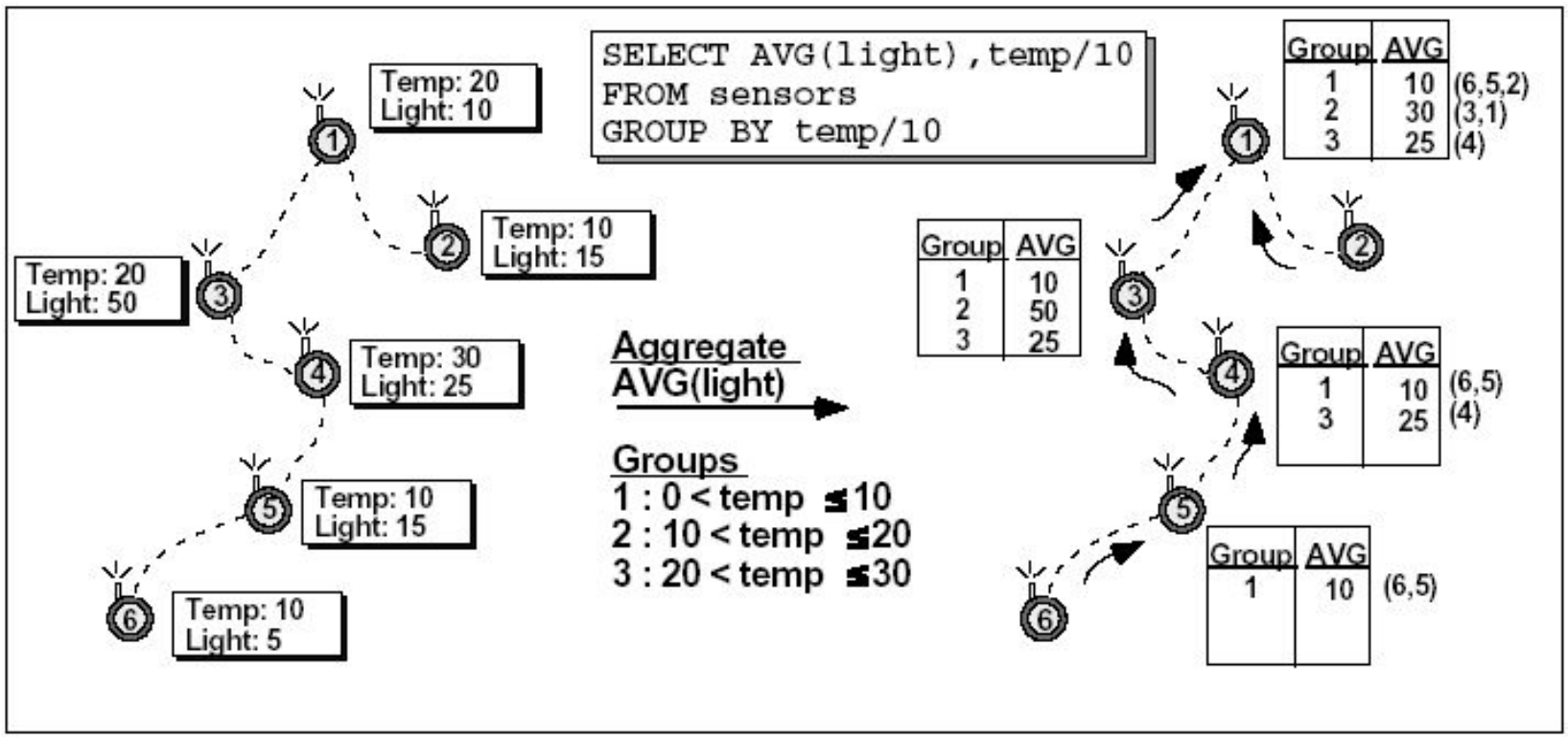


Request r :=
query + interval when the sender is expecting to hear the partial state

Enables scheduling of listening

Network Aggregation - TAG II

- Collection phase
  - Aggregate values are continually routed up



Performance Comparison

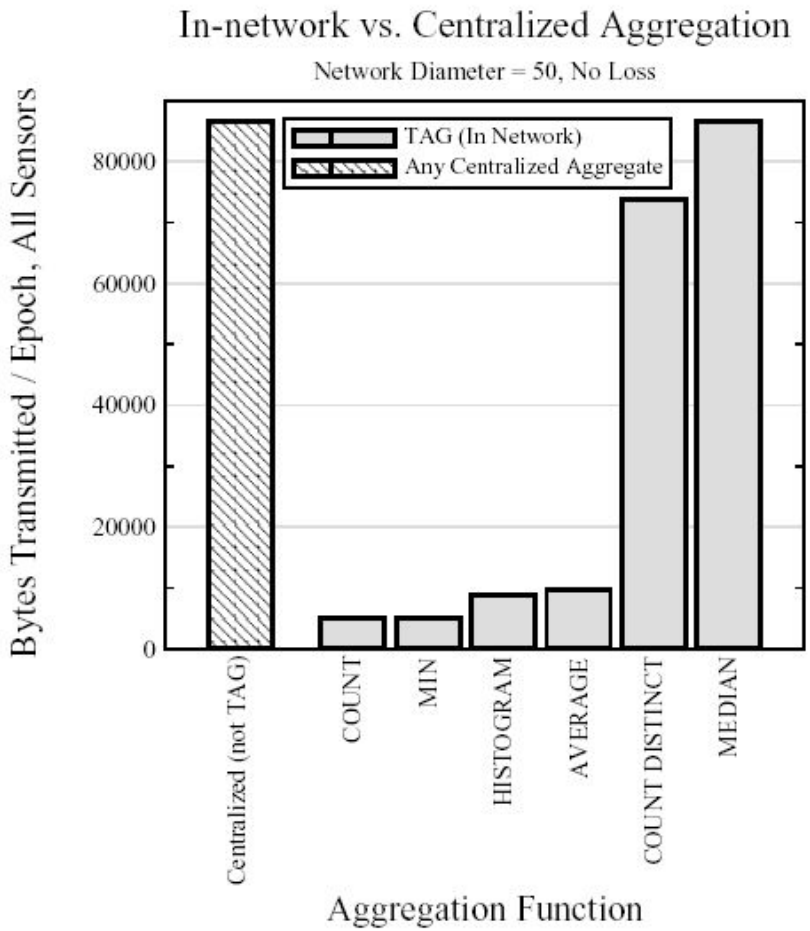


Figure 4: *In network Vs. Centralized Aggregates*

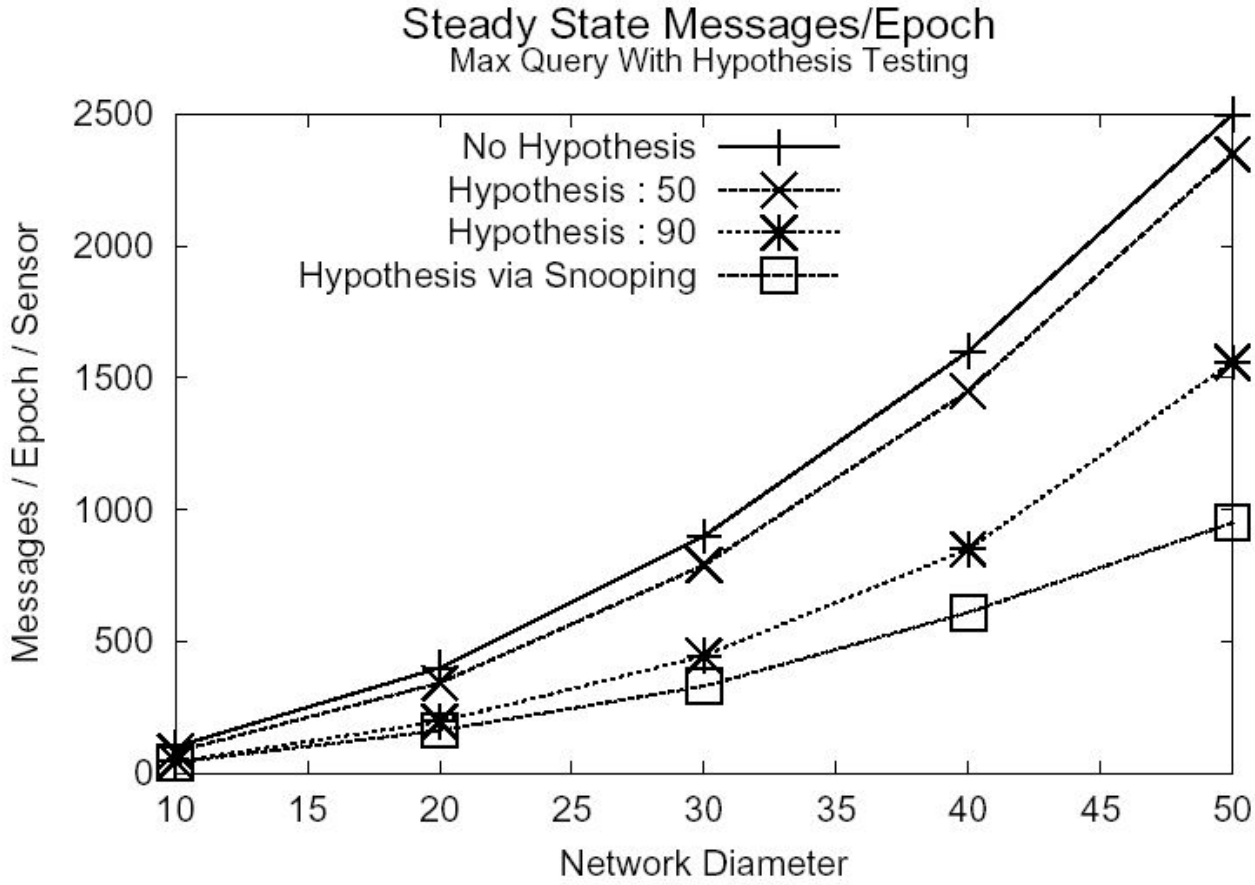
- 2500 nodes (d=5)

- ✓ Benefit of TAG depends on network topology
- ✓ Ex. TAG = Centralized in single-hop env.

Optimizations - Methods

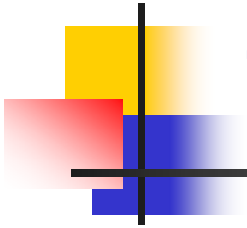
- Snooping: Allow nodes to examine messages not directly addressed to them
  - ⇨ Nodes can initiate aggregation even after missing the start request
  - ⇨ Enables to reduce the number of messages
    (Ex. If a node hears a peer reporting a maximum value greater than its local value, the node doesn't send it)
- Hypothesis Testing: Compute an exemplary local value and issue a new request using the value
  (Ex. MIN: compute the minimum sensor value *m* over the highest levels of the subtree, and issue a new request asking for values less than *m* over the whole tree.)

Optimizations - Results



- Sensor values are uniformly distributed over the range [0..100]
- Hypothesis is made at the root

Figure 5: *Benefit of Hypothesis Testing for* MAX

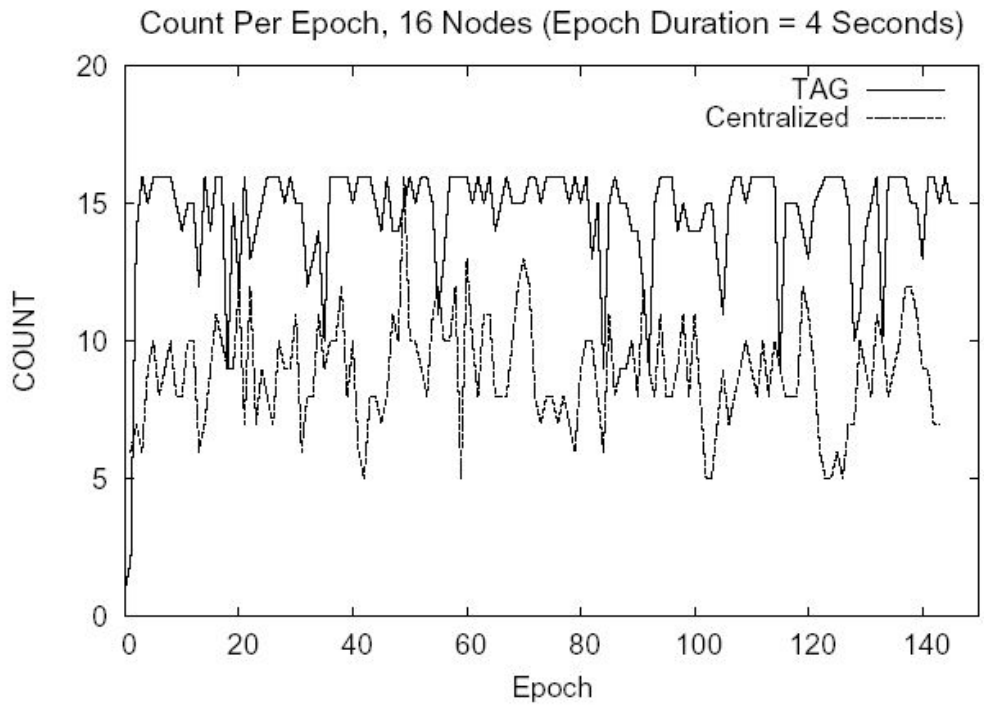


Tolerance to Loss

Communication loss is essential in sensor domain

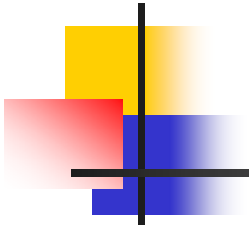
- Networking Faults monitoring and adaptation:
  - Maintain neighbors list and monitors the quality of the link
- For tolerance loss
  - Child cache: parents remember the partial state records reported by their children
  - Redundant reporting: report to 2 parents

Prototype and Experiments



- 16 motes arranged in a depth 4 tree
- TAG is better due to reduced radio contention. (Cent. Approach requires 4685mes., TAG: 2330 mes. i.e. 50% reduction)

Figure 8: *Comparison of Centralized and TAG based Aggregation Approaches in Lossy, Prototype Environment Computing a COUNT over a 16 node network.*



Paper2: Overview

- Status report
- Extend TAG framework
  - Model and Language
- Apply TinyDB to three sensing applications
  - Topographic Mapping
  - Wavelet-based compression
  - Vehicle Tracking

Extending TAG framework

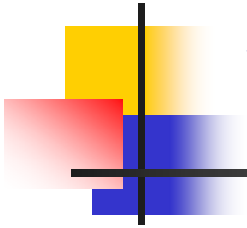
- `SELECT` expr1, expr2, …
  `FROM sensors`
  `WHERE` pred1 `[AND | OR]` pred2 …
  `GROUP BY` groupExpr1, groupExpr2, …
  `HAVING` havingPred1 `[AND | OR]` havingPred2 …
  `SAMPLE PERIOD` t

  "EPOCH DURATION" in Paper1

- temporal aggregates: support inter-epoch aggregation using `window size` and `sliding distance`
  - e.g. `winavg(window_size, sliding_dist, arg)`
  - `winave(10, 1, light)`: computes the 10-sample running average of light sensor readings

New Language Features

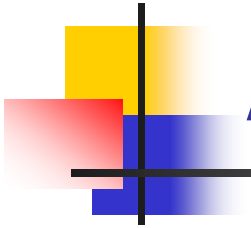
- Events: Initiate automatic response
  - ```
    ON EVENT bird-detect(loc):
      SELECT AVG(light), AVG(temp)
      FROM sensors AS s
      WHERE dist(s.loc, event.loc) < 10m
      SAMPLE INTERVAL 2s FOR 30s
    ```
- Storage Points: Store information locally
  - ```
    CREATE
      STORAGE POINT recentLight SIZE 5s
      AS (SELECT nodeid, light
            FROM sensors
            SAMPLE INTERVAL 1s)
    ```
  - Query can refer the storage points
    ```
    SELECT MAX(lignt) FROM recentLight
    ```



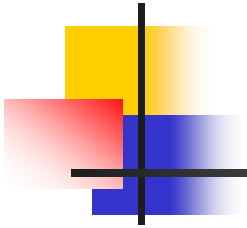
Vehicle Tracking

- Simple tracking problem
  - Single target
  - Target is detected when the running average is beyond a pre-defined threshold
  - Target location is reported as the node location with the largest running average of the sensor
  - The application expects to receives a time series of data from the sensor network once a target is detected

Advantages of TinyDB-based Impl.



- Applications can mix and match existing aggregates and filters of TinyDB's generic query language
- Applications can run multiple queries at the same time
- TinyDB takes care of a lot of system programming issues
- User-defined aggregates are reusable in a natural way
- TinyDB's query optimization techniques can benefit tracking queries



Implementation

- Attributes in TinyDB SQL
  - *mag*: magnetometer reading
  - *time*: current timestamp
  - *nodeid*: unique identifier of each node (The basestation can map *nodeid* to some spatial coordinate)
  - *winavg(10, 1, mag)*: 10-sample running average for the magnetometer readings
  - *max2(agvmsg, nodeid)*: *nodeid* with the largest average magnetometer reading

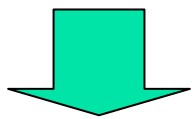
Naïve Implementation

```
// Create storage point holding 1 second worth of running average of
// magnetometer readings with a sample period of 100 milliseconds and
// filter the running average with the target detection threshold.
CREATE STORAGE POINT running_avg_sp SIZE 1s AS
 (SELECT time, nodeid, winavg(10, 1, mag) AS avgmag
  FROM sensors
  GROUP BY nodeid
  HAVING avgmag > threshold
  SAMPLE PERIOD 100ms);

// Query the storage point every second to compute target location
// for each timestamp.
SELECT time, max2(avgmax, nodeid)
FROM running_avg_sp
GROUP BY time/10 // <- to accommodate minor time
                 //    variations between nodes
SAMPLE PERIOD 1s;
```

Naïve Implementation Problem

- All sensor nodes must continuously sample magnetometer every 100ms
- Sampling the magnetometer of a large percentage of nodes is useless (The magnetometer consumes 15mW per sample)



- Query-Handoff Implementation:
  - Start sampling when the target is near and stop the query when the target moves away

Query-Handoff Implementation 1

// Create an empty storage point.

```
CREATE STORAGE POINT running_avg_sp
SIZE 1s (time, nodeid, avgmag)
```

// When the target is detected, run query to compute running average.

```
ON EVENT target_detected DO
SELECT time, nodeid, winavg(10, 1, mag) AS avgmag
INTO running_avg_sp // <- created above
FROM sensors
GROUP BY nodeid
HAVING avgmag > threshold
SAMPLE PERIOD 100ms
UNTIL avgmag <= threshold;
```

Query-Handoff Implementation 2

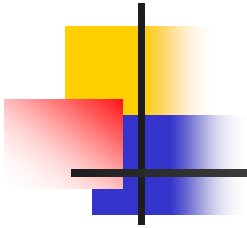
// Query the storage point every sec. to compute target location;
// send result to base and signal target_approaching to the possible
// places the target may move next.

```
SELECT time, max2(avgmag, nodeid)
FROM running_avg_sp
GROUP BY time/10
SAMPLE PERIOD 1s
OUTPUT ACTION
SIGNAL EVENT target_approaching
WHERE location IN
(SELECT next_location(time, nodeid, avgmag)
 FROM running_avg_sp ONCE);
```

Query-Handoff Implementation 3

// When target_approaching event is signaled, start sampling and
// inserting results into the storage point.

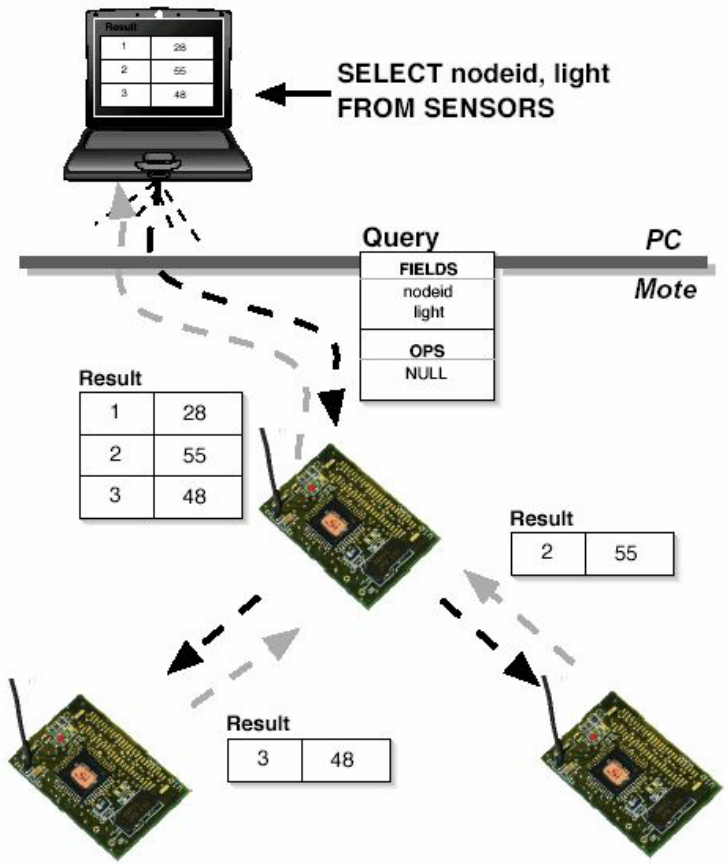
```
ON EVENT target_approaching DO
SELECT time, nodeid, winavg(8, 1, mag) AS agvmag
INTO running_avg_sp
FROM sensors
GROUP BY nodeid
HAVING avgmag > threshold
SAMPLE PERIOD > 100ms
UNTIL avgmag <= threshold;
```



Paper3: Overview

- Present “Acquisitional Query Processing (ACQP)” for sensor networks
- Acquisitional issues:
  - Where, when, and how often data is physically acquired (sampled) and delivered to query processing operators
- Focus on the locations and costs of acquiring data
  - Power based query optimization
  - Semantic Routing Trees

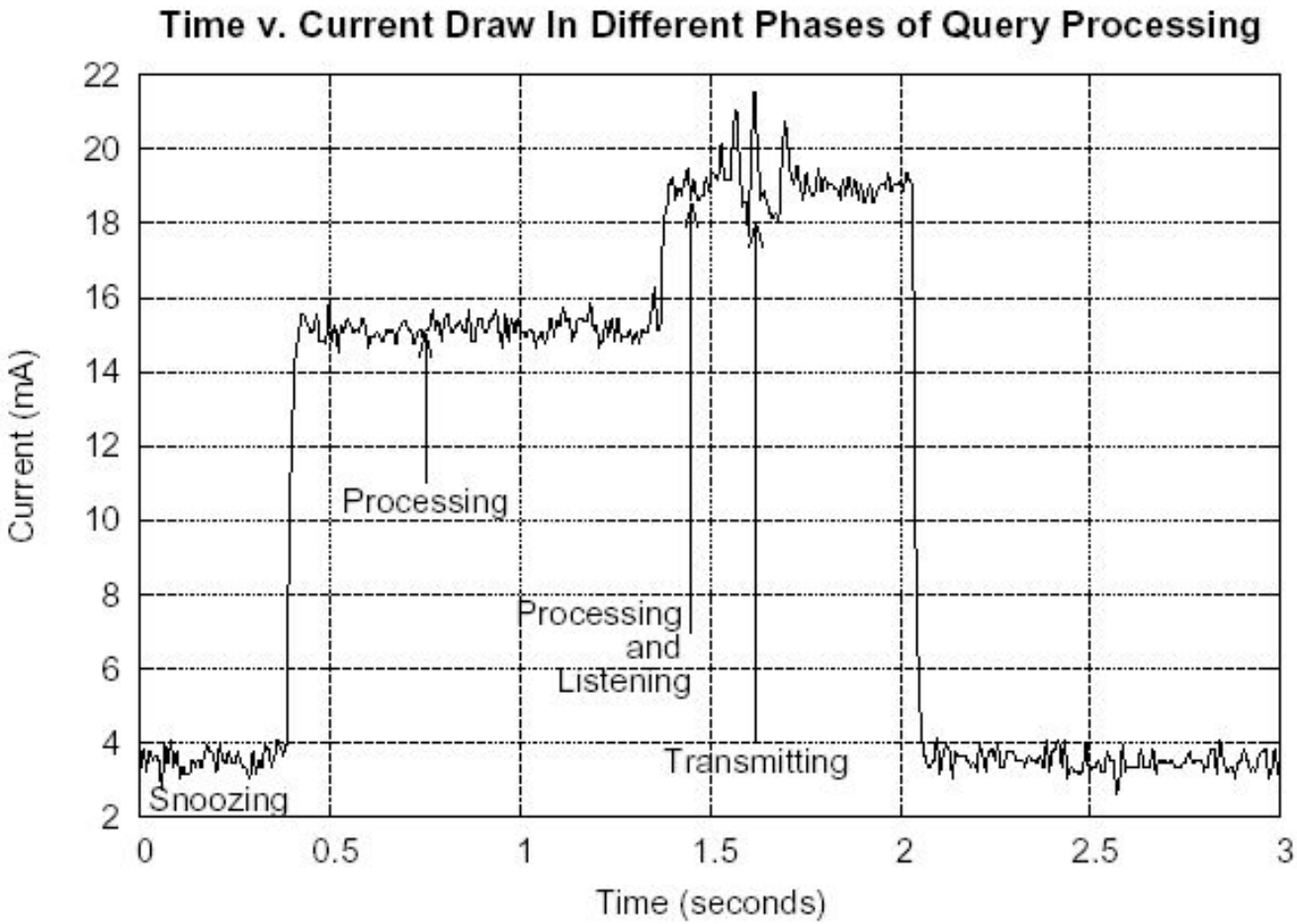
Basic Query Processing Architecture



- Queries are submitted, parsed and optimized at the base station
- They are sent into the sensor network, disseminated and processed
- Results are flowed up the routing tree

Figure 1: A query and results propagating through the network.

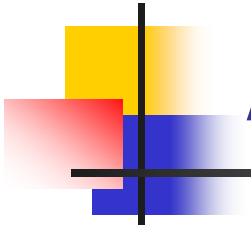
Power Consumption in Sensors



Acquisitional Query Language 1

- SELECT-FROM-WHERE Queries:
  - Support selection, projection, aggregation, and join (different from TAG paper)
  - Explicit support for sampling, windowing, and subqueries
  - Windows in TinyDB are defined as fixed-size materialization points over the streams
    ex.

```
CREATE STORAGE POINT recentlight SIZE 8
    AS (SELECT nodeid, light FROM sensors
        SAMPLE INTERVAL 10s)
```



Acquisitional Query Language 2

- Event-Based Queries:

  Starting on events:

```
ON EVENT bird-detect(log):
   SELECT AVG(lignt), AVG(temp),
  event.loc
   FROM sensors AS s
  WHERE dist(s.loc, event.loc) < 10m
  SAMPLE INTERVAL 2s FOR 30s
```

  Stopping on events:

```
STOP ON EVENT(param) WHERE cond(param)
```

Acquisitional Query Language 3

- Lifetime-Based Queries:

```
SELECT nodeid, accel
  FROM sensors
  LIFETIME 30 days
```

  - Query semantics: The network should run for at least 30 days, sampling light and acceleration sensors at a rate that is as quick as possible and still satisfies this goal.
  - TinyDB performs lifetime estimation to satisfy a lifetime clause
  - Sample rate is calculated by TinyDB

Power-Based Query Optimization 1

- Base station performs a simple cost-based query optimization to minimize overall power consumption
- Ordering of sampling and predicates:

```
SELECT accel, mag
  FROM sensors
  WHERE accel > c1
  AND mag > c2
  SAMPLE INTERVAL 1s
```

  - Possible query plans:
    1. Sample the magnetometer and the accelerometer before applying selections
    2. Sample the magnetometer and apply selection over its reading before the accelerometer is sampled and filtered
    3. Opposite to 2.
  - Calculate each plan's cost and choose minimum one

Power-Based Query Optimization 2

- Event Query Batching:

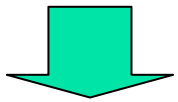
```
ON EVENT e(nodeid)
  SELECT a1
  FROM sensors AS s
  WHERE s.nodeid = e.nodeid
  SAMPLE INTERVAL d FOR k
```

  - It is possible for multiple instance of the internal query to be running at the same time
  - ⇨ Multi-query optimization

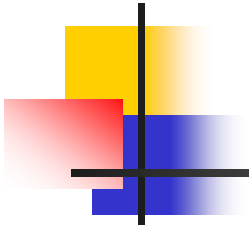
Power-Based Query Optimization 3

- Query Rewriting:

```
ON EVENT e(nodeid)
  SELECT a1
  FROM sensors AS s
  WHERE s.nodeid = e.nodeid
  SAMPLE INTERVAL d FOR k
```



```
SELECT s.a1
  FROM sensors AS s, events AS e
  WHERE s.nodeid = e.nodeid
  AND e.type = e
  AND s.time - e.time <= k AND s.time > e.time
  SAMPLE INTERVAL d
```



Power Sensitive Dissemination & Routing

- Semantic Routing Tree (SRT)
  - Objective: To determine if any of the nodes below it will need to participate in a given query
  - Conceptually: An index over some attributes A that can be used to locates nodes that have data relevant to the query

SRT Example

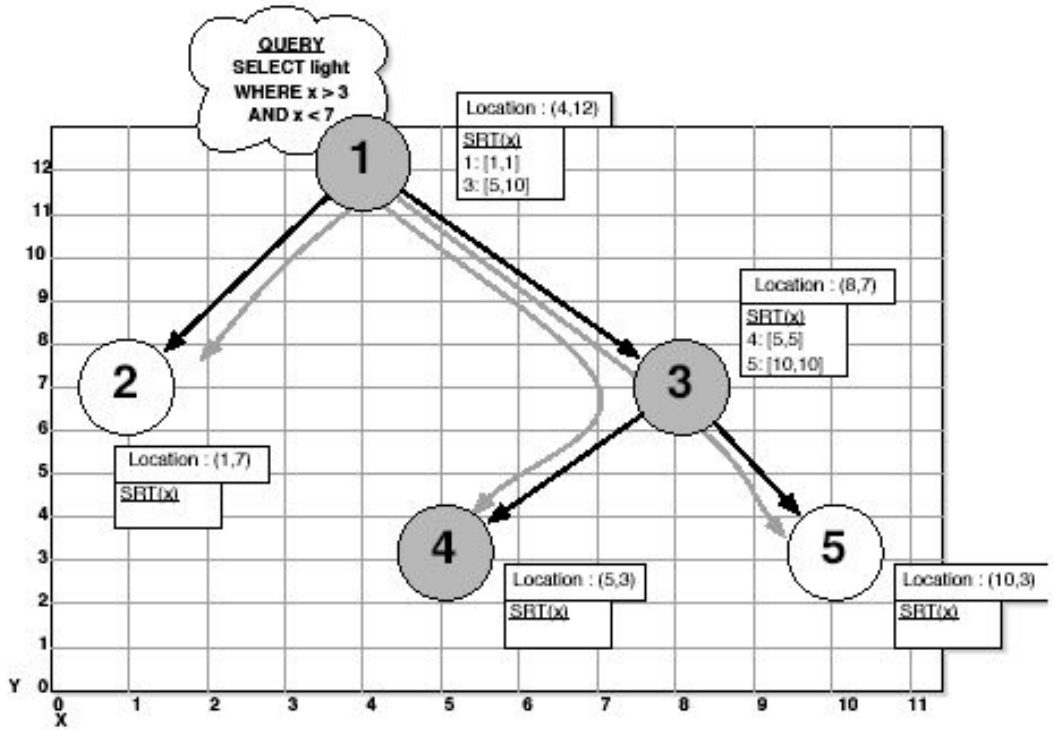
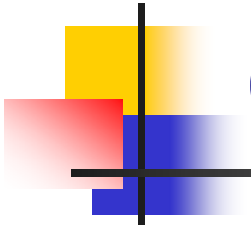


Figure 6: A semantic routing tree in use for a query. Gray arrows indicate flow of the query down the tree, gray nodes must produce or forward results in the query.



Query Processing

- Prioritizing Data Delivery:
  - Results at a node are enqueued onto a radio queue
  - Send a tuple that will most improve the "quality" of the answer
  - Prioritization schemes
    - naive: no tuple is considered more valuable -> FIFO
    - winavg: create average of two tuple values and drop one
    - delta: calculate tuple scores by the difference from the most recent (in time) value successfully transmitted, and drop the lowest one
- Adapting Rates and Power Consumption:
  - Compute *predicted battery voltage* for a time *t*
  - Compare its current voltage to the predicted battery voltage and re-run lifetime calculation if necessary

Signal Approximations

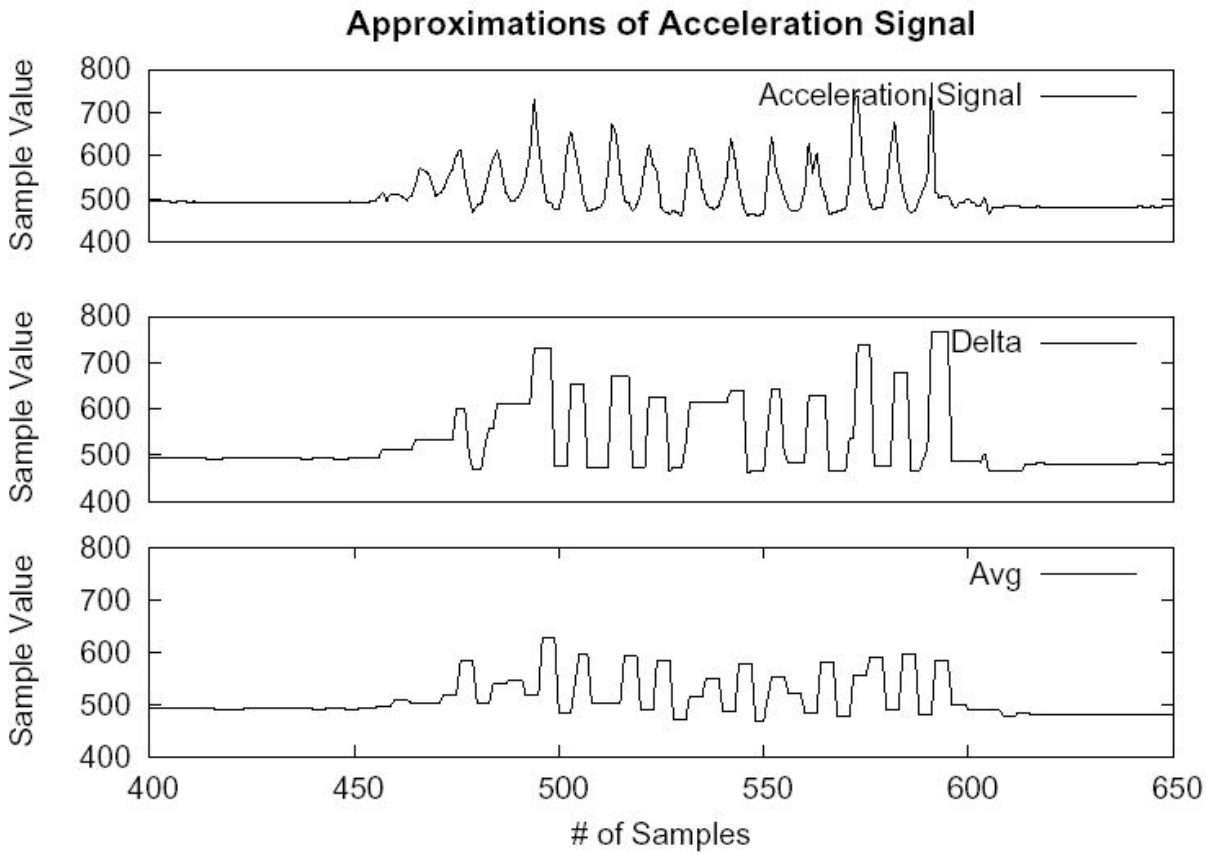
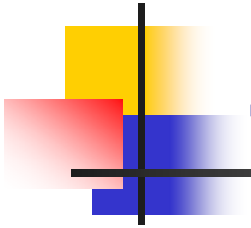


Figure 8: An acceleration signal (top) approximated by a delta (middle) and an average (bottom), K=4.



Summary

1. Paper 1: [madden02b]
   - ✓ Present the Tiny AGgregation (TAG) service
   - ✓ Declarative query, single table
   - ✓ Ad-hoc routing, Network aggregation
   - ✓ Performance: TAG > Centralized approach
2. Paper 2: [hellerstein03]
   - ✓ Status report of TinyDB
   - ✓ New language features: Events, Storage points
   - ✓ TinyDB Applications: Topographic mapping, wavelet-based compression, vehicle tracking
3. Paper 3: [madden03a]
   - ✓ Present an Acquisitional query processing framework
   - ✓ Discuss techniques such as “Power-based query optimization”, “Semantic Routing Trees” to reduce power consumption of sensor devices