TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks

Abhinaya Kalyanaraman
Motivation

Smart Sensors

Devices that measure real world phenomena (temperature, buildings movement in earthquake)

Wireless, battery powered, full-fledged computers

Motes

Smart sensor developed in UC Berkeley

TinyOS: ease the deployment of motes in ad-hoc networks

Ad-hoc networks: networks not relying on pre-existing infrastructures (routers, access point)
Motivation

Challenges working with smart sensors:

- Limited power supply while needs long lived deployment & zero maintenance
- Power consumptions dominated by transmitting/receiving messages
- Users have to write low level and error-prune code to collect and aggregate data from the network

Goals:

- Have more power-conserving algorithm to reduce radio communication
- Provide a high-level programming abstraction to hide low level details
Tiny AGgregation (TAG)

- Generic aggregation service for *ad hoc* networks of TinyOS motes
- Provides a SQL-like declarative languages for data collection and aggregation
- Intellignet execution of aggregation queries to save time and power
- In network aggregation: calculates aggregation in each node as data flows through it
- Users inject query into a storage-rich basestation (root)
Ad-Hoc Routing Algorithm

- A routing tree is needed for sensor to route data
- A root is assigned with level 0. Root broadcasts message \{id, level\} to other nodes in its range
- Upon receiving a message, a sensor (without level) will update its level and parent node, and does the broadcast again
- Algorithm ends when all nodes have a level
Query Model and Environment

- SQL-style query syntax
- Example: microphone sensor network for monitoring volume

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

- Queries in TAG have the following form:

```sql
SELECT { agg (expr), attrs }
FROM sensors
WHERE { selPreds }
GROUP BY { attrs }
HAVING { havingPreds }
EPOCH DURATION i
```

Supports only aggregates and not arbitrary joins
Query Model and Environment

- The output of a TAG query is a stream of values, rather than a single aggregate value.
- Each record consists of one <group id, aggregate value> pair per group.
- Each group is time-stamped.
- Readings used to calculate an aggregate all belong to the same time-interval epoch.
- EPOCH DURATION specifies the amount of time devices wait before sending samples to other devices.
Structure of Aggregates

TAG implements aggregates via three functions:

- a merging function $f$
- an initializer $i$
- an evaluator $e$

In general, $f$ has the following structure:

$$< z > = f ( < x >, < y >)$$

Partial-state record resulting from the application of $f$ function to $<x>$ and $<y>$.

Partial state records - intermediate state over those values that will be required to compute an aggregate.
Structure of Aggregates

Example: AVERAGE Aggregate

\[ f ( < S_1, C_1 > , < S_2, C_2 > ) = < S_1 + S_2, C_1 + C_2 > \]

- Merge function for a function, e.g. AVERAGE
- Partial state records

Initializer \( i (x) \) returns the tuple \( < x, 1 > \).

For AVERAGE, the evaluator \( e ( < S, C >) \) returns \( S/C \).
Aggregate functions

SQL supports the following functions:

- COUNT
- MIN
- MAX
- SUM
- AVERAGE

Want TAG to support more

Solution: Generic classification of aggregate functions using the proposed dimensions
Aggregate functions

In order to evaluate the performance of TAG:

Dimensions proposed:

- **Duplicate sensitive**: It determines if aggregates can be affected by duplicate readings from a single device
- **Exemplary aggregates**: return one or more representative values from the set of all values
- **Summary aggregates**: compute some property over all values
- **Monotonic aggregates**: determines whether some predicates (such as HAVING) can be applied in network
- **Partial state**: relates to the amount of state required for each partial state record which is inversely related to TAG’s performance
In-Network Aggregation

Tree-like Topography & Level based Routing
Distribution Phase
In-Network Aggregation

Tree-like Topography & Level based Routing
Collection Phase

Root, Level 0
Level 1
Level 2
Reply
In-Network Aggregation

Tree-like Topography & Level based Routing

Collection Phase

Parents need to wait until they heard from their children before routing aggregate up for the current epoch.

Solution: Subdivide epoch s.t. children are required to delivered their partial state records during a parent-specified time interval
In-Network Aggregation

**SELECT MAX(temp) FROM sensors**

Without TAG

- Total Messages: 14
- Numbers: [5,7,4,8,9,3,1]

```
SELECT MAX(temp) FROM sensors
```

MAX=9

With TAG

- Total Messages: 9
- Epoch 1: [7,8,3]
- Slot 2: [4,1,9]
- Slot 3: [8,9,5]

```
SELECT MAX(temp) FROM sensors
```

MAX=9
Evaluation

TAG Performance

In-network vs. Centralized Aggregation

Network Diameter = 50, No Loss

Bytes Transmitted / Epoch, All Sensors

Aggregation Function
Optimization

Multiple Parents

- Increased Reliability
  - Duplicate insensitive aggregates
  - Aggregates that can be expressed as a linear combination of parts

![Diagram showing multiple parents and increased reliability through linear combination of parts](image-url)
Improving Tolerance to Loss

Effect of Loss - Single loss

Monitor quality of links to neighbor

- By tracking the proportion of the packets received from each neighbor
- Assume parent fails if hasn’t heard from it for a certain period of time
- Pick a new parent according to the link quality

(a) Maximum Error  (b) Average Error
Improving Tolerance to Loss

Child Cache

- Increased Availability

Use old results when new results are not available
Comparison between WABD and TAG

- In the sensor network scenario, the motes have very little compute power — hence the focus is on aggregation where they only evaluate simple functions as opposed to WABD setting.

- The aggregation techniques used in TAG are also applicable in WABD.

- They only support aggregates and not arbitrary joins as opposed to WABD.
Conclusion

- In-network aggregation offers an order of magnitude reduction in bandwidth consumption compared to centralized aggregation.

- The declarative query enables users to use in-network aggregation with having to write low-level code.
Questions?