Tag：a Tiny Aggregation Service for Ad-Hoc Sensor Networks

Samuel Madden, Michael Franklin, Joseph Hellerstein,Wei Hong

UC Berkeley

Usinex OSDI’02

Outline

- Introduction
- The Tiny AGgregation Approach
- Aggregate structure and Implementation
- Additional Benefits of TAG
- Simulation
- Optimizations
- Experimental results
- Conclusion

Introduction

- Sensor networks is to extract data from the network environment
- Data could be raw sensor readings or summaries/aggregations of many readings
- Previous work view aggregation as an application specific mechanism
- It would be implemented on a per-need basis in error prone languages like C(aggregation)

Introduction(cont.)

- It must be provided as a ***core service*** by the system software (TAG)rather than a set of ***extensible C APIs***
- Useful to non networking or programming experts who can focus on their custom applications without worrying about underlying hardware/software and data collection mechanisms

This Paper Approach

- Hardware:Motes
  - Atmel 4MHz
  - RAM 4KB
  - FlashRom 128KB
  - EEPROM 512kB
  - 917MHzRFM
- SoftWare： TinyOs
- Routing：
  Ad-Hoc (AODV)
- Motivation **TAG Approach**

Tag Approach

- Essential Attributes
  - **SQL-style syntax**： Simple, declarative interface for data collection and aggregation
  - Time & power-efficient manner： Intelligently distributes and executes aggregation queries in the sensor network
  - Computing over the data as it flows through the sensors, discarding irrelevant information and combining relevant readings

Tag operation

- Users pose aggregation queries from a base station
- Messages propagate from the base station to all nodes through routing tree rooted at base station
- **Divide time into epoch** and in each epoch, Sensors route local and children data back to user using routing tree
- As data flows up the tree, it is aggregated according to aggregation function and value based partitioning specified in original query

SQL style query syntax

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

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

- **SELECT**: specifies an arbitrary arithmetic expression over one or more aggregation values
- ***expr***: The name of a single attribute
- **agg**: *A*ggregation function
- ***Attrs***: selects the attributes by which the sensor readings are partitioned
- **WHERE, HAVING**: Filters out irrelevant readings
- **GROUP BY**: specifies an attribute based partitioning of readings
- **EPOCH DURATION**: Time interval of aggr record computation
- Each record is a <*groupID, aggregate_value*> pair

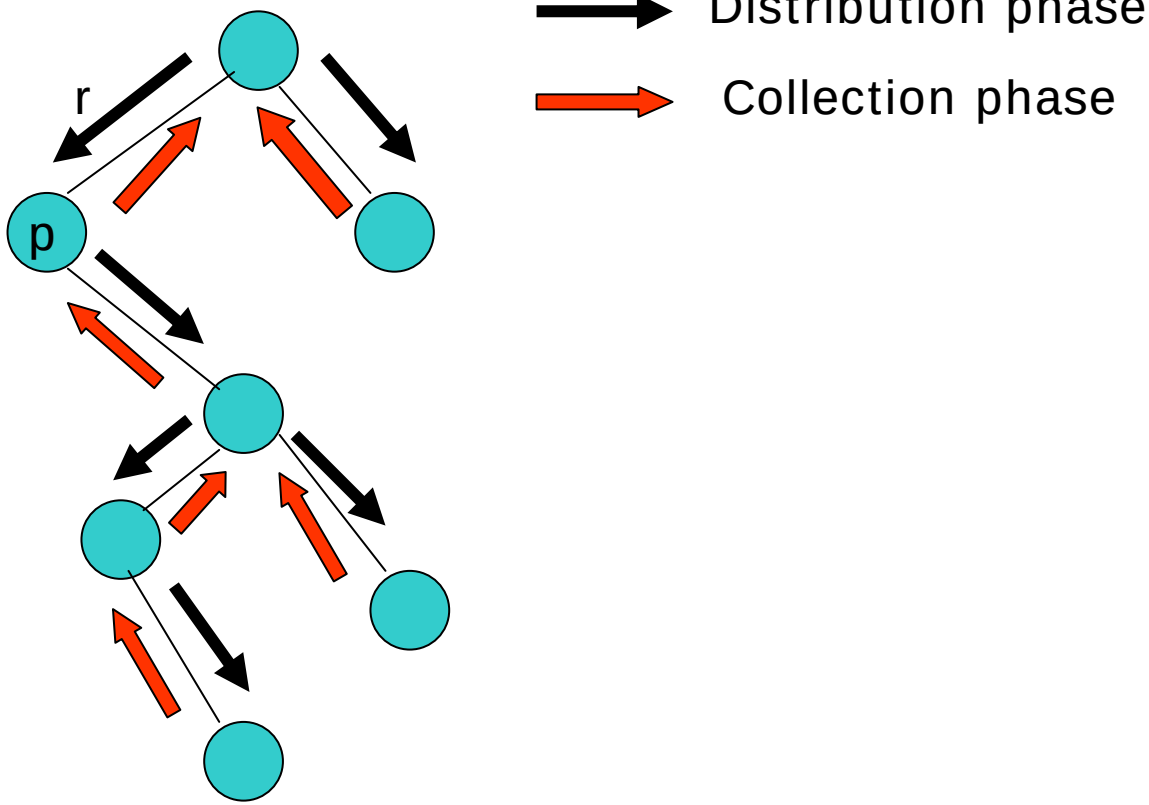
Aggregate structure

- Implemet **agg** via three function
  - **Merging function** f
    - <z>=f(<x>,<y>)
      - Ex.For Average case :SUM and COUNT
        f(<S1,C1>,<S2,C2>)=<S1+S2,C1+C2>
  - **Initializer** I
    - i(x)=<x,1>
  - **Evaluator** e
    - e(<S,C>)=S/C

Attribute Catalog

- Queries in TAG contain named attributes
- Each motes has a **small catalog of attributes** that can be searched for attributes of a specific name
- Central query processor caches or stores attributes of all motes it may access
- When a TAG sensor receives a query, it converts names fields into local catalog identifiers, or sends back a NULL if it is lacking the attribute
- Not all nodes have identical catalogs, so can be implemented for heterogeneous sensing

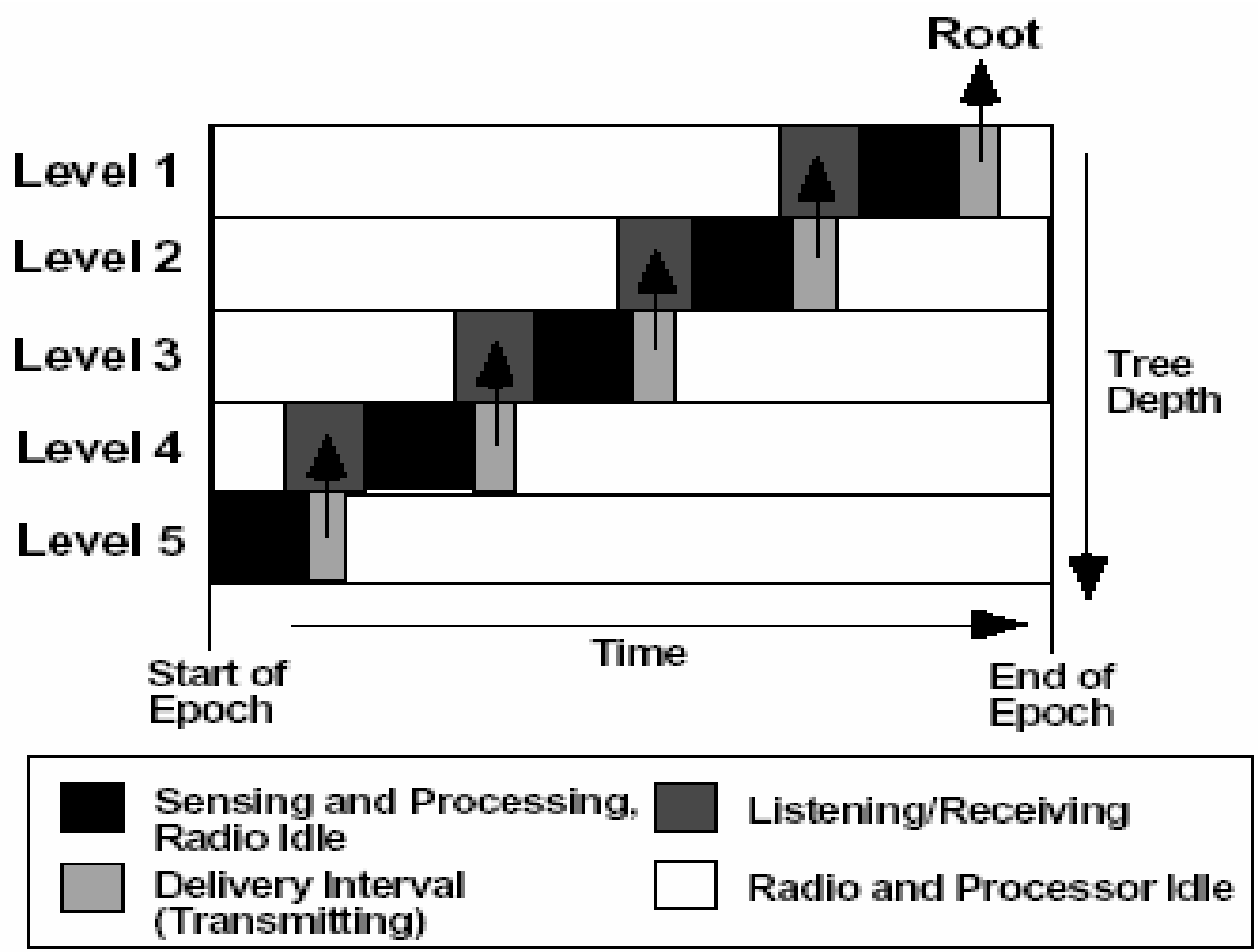
In Network Tiny Aggregation



In Network Tiny Aggregation

- Distribution and Collection Phases with goal to reduce messages
- Mote p receives a request r to aggregate, wakes up synchronizes clock
- p chooses sender of message as its parent **and r includes interval** in which p must **reply back** with partial state record
- P then sends the message down the network to its children, setting the **delivery interval** for its children to reply back by broadcasting r
- This propagation ends when all nodes in the network are queried
- During the epoch after the propagation, each mote listens for messages from its children **during the specified interval**.
- Then computes a partial state record consisting of any child values heard with its own local sensor readings.
- Each mote transmits this partial state record up the network
- Every epoch, new aggregate produced
- Most of the times, motes are idle and in low power state

Pipelining the communication schedule



Grouping

- Each sensor placed in exactly one group partitioned according to an expression over all attributes
- **Basic grouping technique:** push the query down network, as the nodes to choose which group they belong to, as they flow upwards, update aggregate values in appropriate groups, and now partial state records are attached with a group ID (groupid,value)

Grouping example

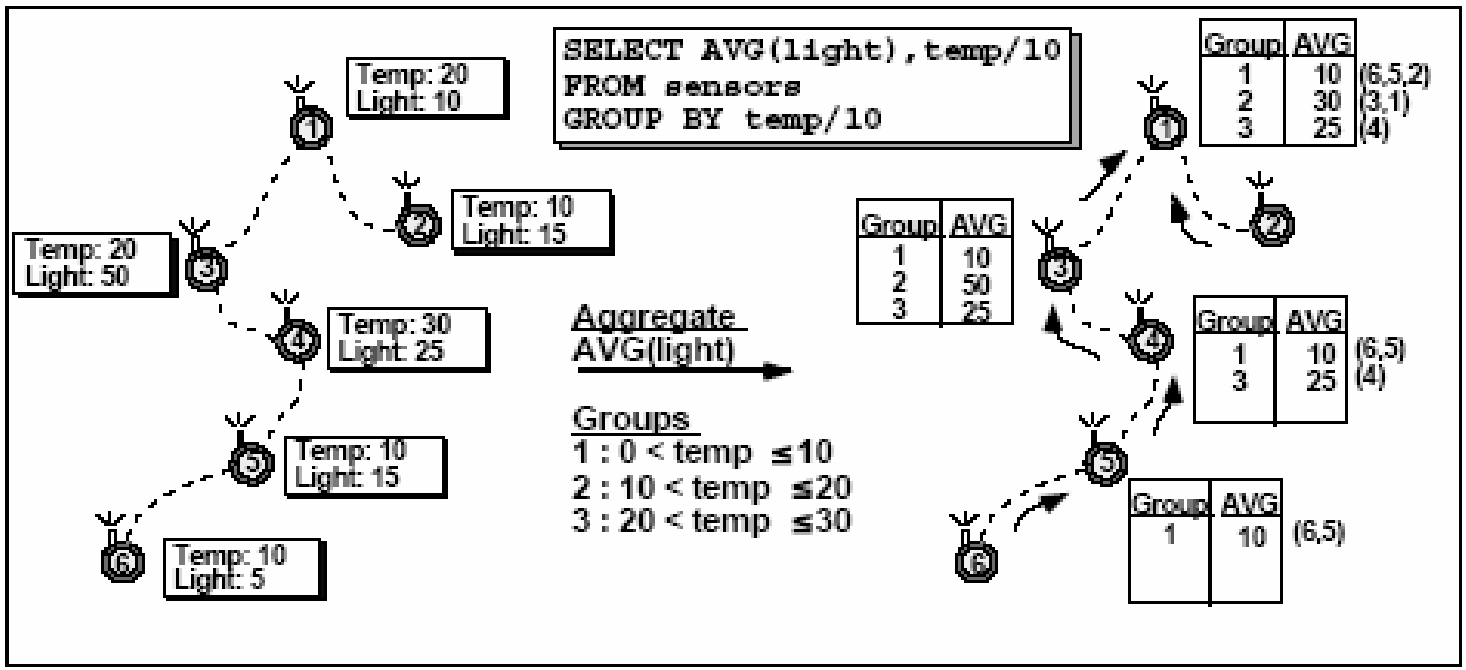


Figure 2: *A sensor network (left) with an in network, grouped aggregate applied to it (right).* Parenthesized numbers represent nodes that contribute to the average

Additional Benfits of TAG

- Ability to **tolerate disconnections** and **loss**, as lost nodes can **reconnect** to network by **listening to another node's state records**, even if not intended for them.
- **In TAG**, each node transmits **only one value**. In **non-TAG**, nodes at top of tree have to transmit **lot more values** and get drained faster
- Also, by dividing time into epochs, long idle times are present for the processor and radio to be put in deep sleep modes with little power.

Simulation Based Evaluation

- Simulated TAG in Java
  - **Simple:** nodes have perfect lossless communication with regularly placed neighbors
  - **Random:** nodes' placement random
  - **Realistic model** to capture actual behavior of radio and link layer on TinyOS motes (uses results from real world experiments to approximate actual loss of TinyOS radio) – has high loss rates
- Simulator models mote behavior at a course level: time divided in epochs, messages encapsulated in java objects
- Simulation cannot model fine grained details of network like real world characteristics and modeling radio contention at a byte level

Performance of TAG

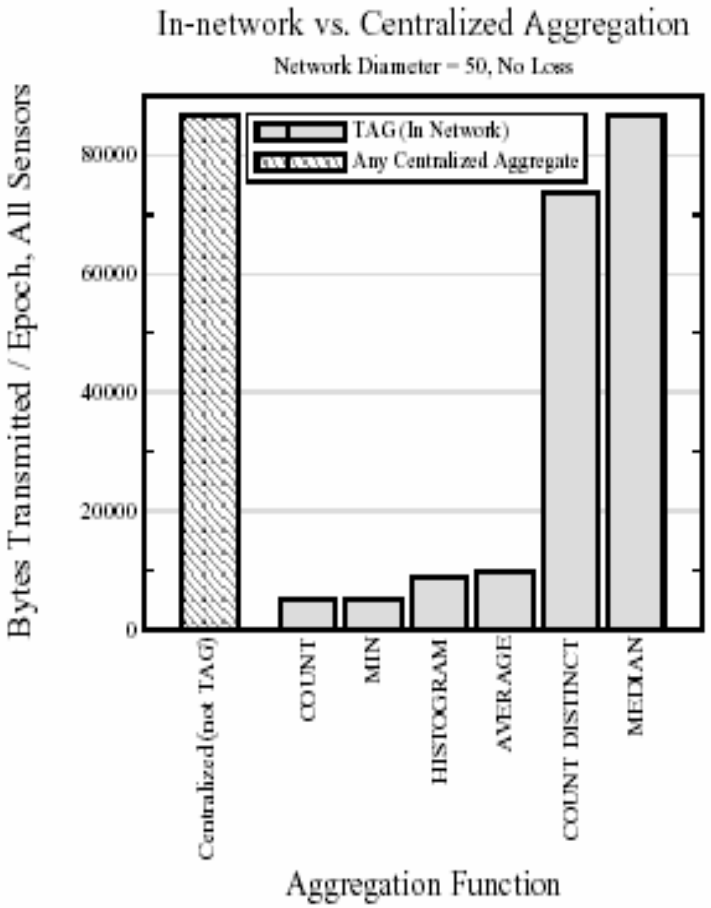


Figure 4: *In network Vs. Centralized Aggregates*

- MIN & COUNT – small as only one integer per partial state record
- Average – 2 integers, so double cost of distributive
- Median – same as centralized as parents have to forward all children's values to root
- Count Distinct – only slightly less expensive
- Histogram – size of fixed-width buckets = 10, sensor values ranged over interval [10..1000]

Optimizations

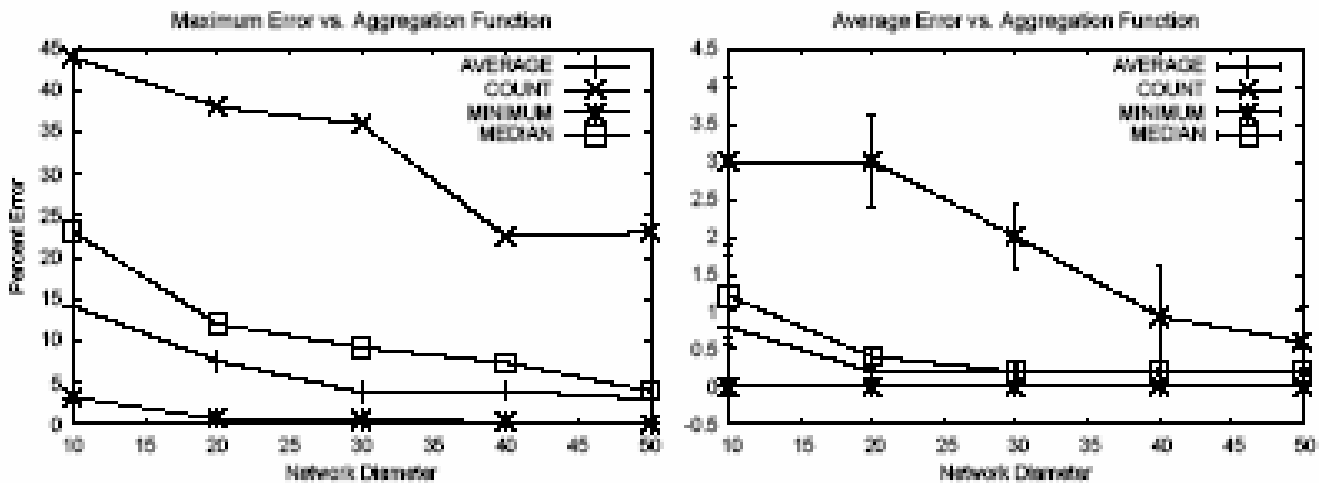
- **Channel sharing**:
  - If node misses initial request to aggregate, it can snoop (at regular intervals) network traffic and "**catch up**" and **include itself**
  - Can also reduce the number of messages if node sees a higher value reported that its own for MAX, it will not bother to send a message
  - **Reduces number of messages sent**

Improving tolerance to loss

- Wanted to see the effect of the **loss of a single node** and how long network takes to stabilize and **children of lost node to find new parents**

- Max loss of variable and some aggregates are more sensitive to a **single loss** than others.
- COUNT has large error in worst case: if node that connects the root to a **large portion** of the network is lost, temporary error is high.

Effect of a Single Loss



(a) Maximum Error (b) Average Error

Figure 6: *Effect of a Single Loss on Various Aggregate Functions.* Computed over a total of 100 runs at each point. Error bars indicate standard error of the mean, 95% confidence intervals.

Child Caching

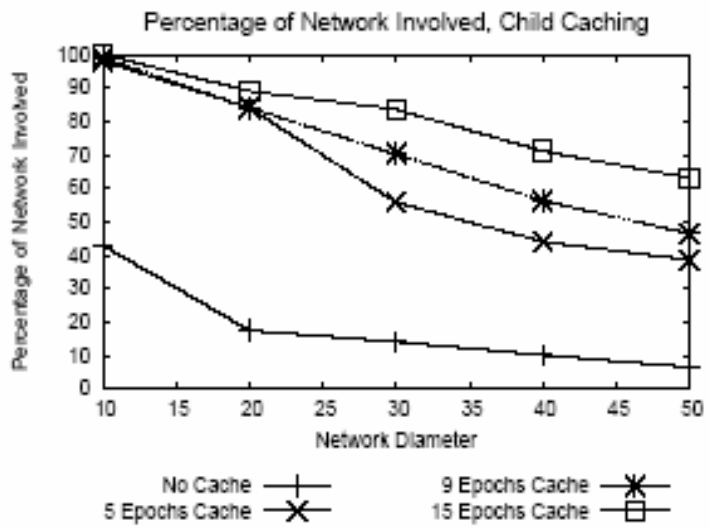
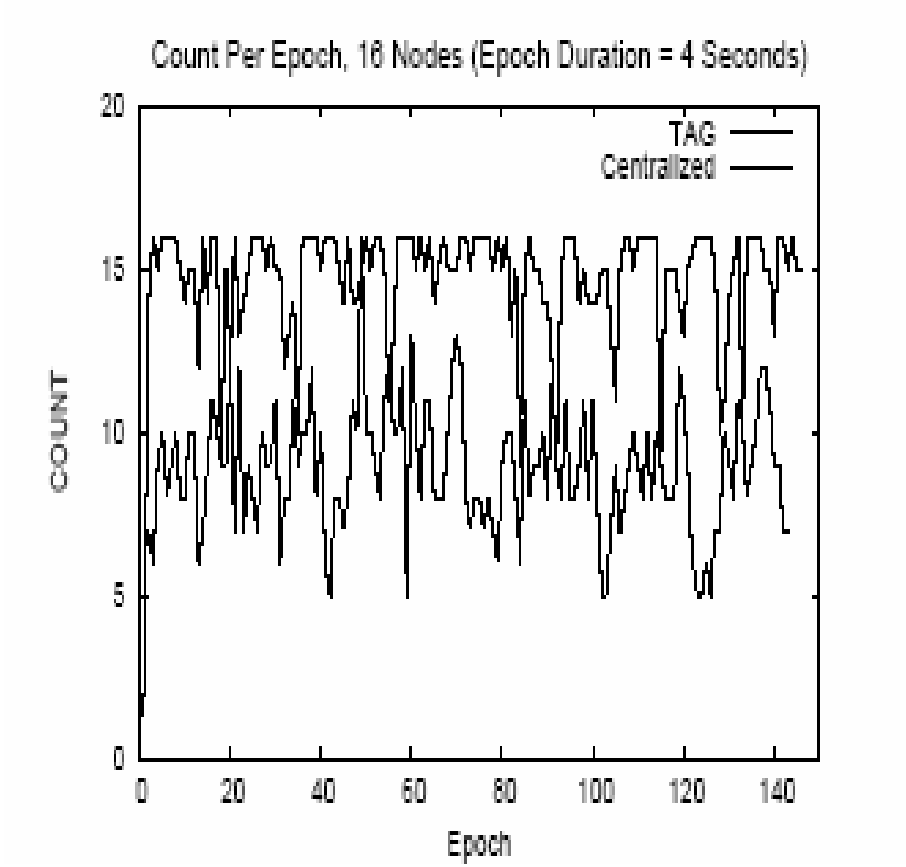


Figure 7: *Percentage of Network Participating in Aggregate For Varying Amounts of Child Cache*

- Parents remember the partial state records their children reported for some number of rounds
- Use those previous values when new values are unavailable due to lost child messages

Experimental Results

- 16 nodes, depth 4 tree, COUNT aggregate
- Number of messages
  - Centralized: 4685
  - TAG: 2330
  - 50% better!!
- Centralized approach: Increased network contention, Per hop loss rates = 15% (TAG = 5%)
- Centralized approach : Only 45% of messages from nodes at bottom of tree reached root.



Conclusion

- TAG is based on a declarative query interface
  - Uses aggregation extensively
  - Makes network tasking easier for the user who does **not** have to modify **low level code** or **worry about topology, routing and loss tolerance**
- TAG better than centralized approaches in most cases due to aggregation
- In network approach an order of magnitude reduction in bandwidth and power consumption