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

Samuel Madden, Michael Franklin, Joseph Hellerstein,
and Wei Hong (OSDI 2002)
UC Berkeley and Intel Research

Presented by Shameem Ahmed

Motivations

- Basic Problem
  - How to gather interesting data from thousands of motes in sensor network?
  - Data could be raw sensor readings or summaries/aggregations of many readings

- Prior Approach
  - Aggregation as application-specific mechanism

- TAG Approach
  - Aggregation as a core service rather than a set of extensible C APIs
  - TAG process aggregates in the network to save power
    - Sensor nodes are power constrained
    - Msg communication consumes a lot of power
      - Transmission of 1 bit = Execution of 800 instructions!!!

TAG

- Aggregates values in low power, distributed network
- Implemented on TinyOS Motes
- Simple, declarative interface for data collection and aggregation – SQL style
- Tree based methodology
  - Root node generates requests and dissipates down the children

Language: SQL style query syntax

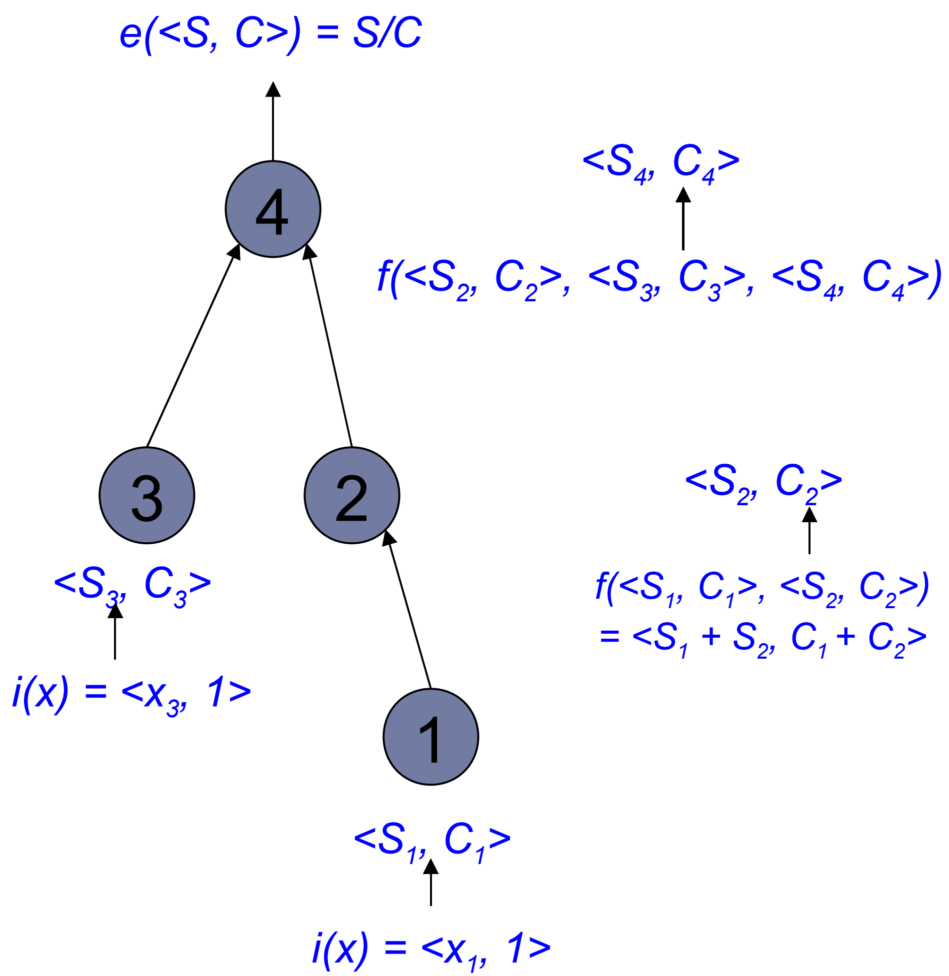
```
SELECT {agg(expr), attrs} FROM sensors
  WHERE {selPreds}
  GROUP BY {attrs}
  HAVING {havingPreds}
  EPOCH DURATION i
```

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

- SQL-like syntax
  - SELECT: specifies an arbitrary arithmetic expression over one or more aggregation values
  - expr: The name of a single attribute
  - agg: Aggregation 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 sensor readings
  - EPOCH DURATION: Time interval of aggr record computation

Aggregate Functions

- 3 components:
  - Merging function f
  - Initializer i
  - Evaluator e
- Example: AVERAGE
  - Partial State record: *<S,C>*
- Merging function
  - *f(<S1,C1>,<S2,C2>) = <S1+S2, C1+C2>*
- Initializer
  - *i(x)=<x,1> where x = sensor value*
- Evaluator
  - *e(<S,C>)=S/C*



* Example: Internet

TAG Taxonomy (1/2)

- Aggregates are classified according to 4 properties

  (1) Duplicate sensitivity
  - Insensitive aggr: unaffected by duplicate readings from same node (Max, Min)
  - Sensitive aggr: Affected by duplicate readings from same node (Count, Average)

  (2) Exemplary vs Summary
  - Exemplary returns one or more representative values of a set (Max, Min)
  - Summary returns some property over all values (Count, Average)

  (3) Monotonic aggregates
  - When 2 partial records s1 and s2 are combined via f, resulting state record s' will have

    either e(s') >= MAX (e(s1), e(s2)) or e(s') <= MIN (e(s1), e(s2))
  - Important when determining whether some predicates (e.g. HAVING) can be applied in network

TAG Taxonomy (2/2)

- Aggregates are classified according to 4 properties

  (4) Amount of state required for every partial state record

  Example: Partial AVERAGE record consists of pair of values, while partial COUNT record consists of a single value

  - Distributive: size of partial state records = size of final state record (MAX)
  - Algebraic: Partial states are of fixed size but differ from final state (AVERAGE)
  - Holistic: Partial states contain all sub-records (MEDIAN)
  - Unique: Similar to Holistic, but amount of state that must be propagated is proportional to # of distinct values in the partition (COUNT DISTINCT)
  - Content Sensitive: Size of partial records depend on content (HISTOGRAM)

TAG Operation

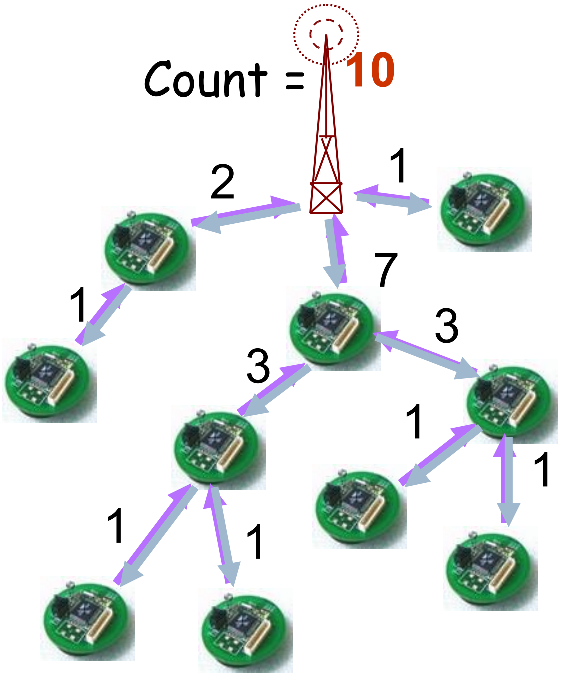
○ Users pose aggregation queries from a base station

SELECT COUNT (*) FROM sensors

○ Messages propagate from the base station to all nodes through routing tree rooted at base station

○ Divide time into epoch and in each epoch, children sends data back to parent using routing tree

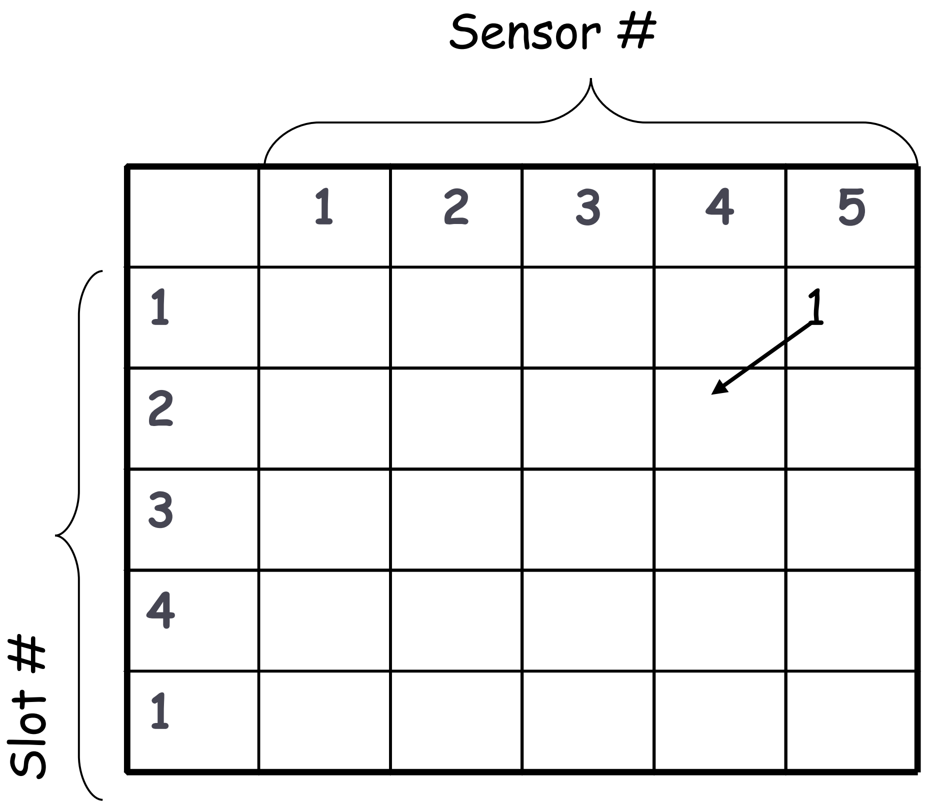
○ As data flows up the tree, it is aggregated according to aggregation function (here count)

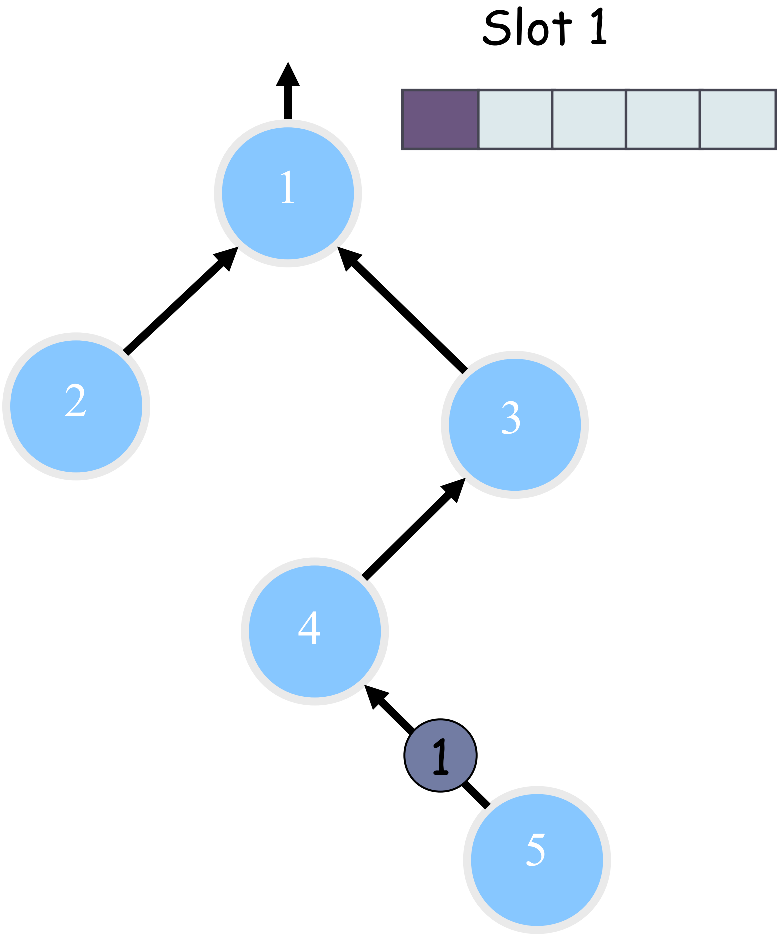


* Image source: Internet

Illustration: Aggregation

SELECT COUNT(*) FROM sensors





* Example: Author's slide

Illustration: Aggregation

SELECT COUNT(*) FROM sensors

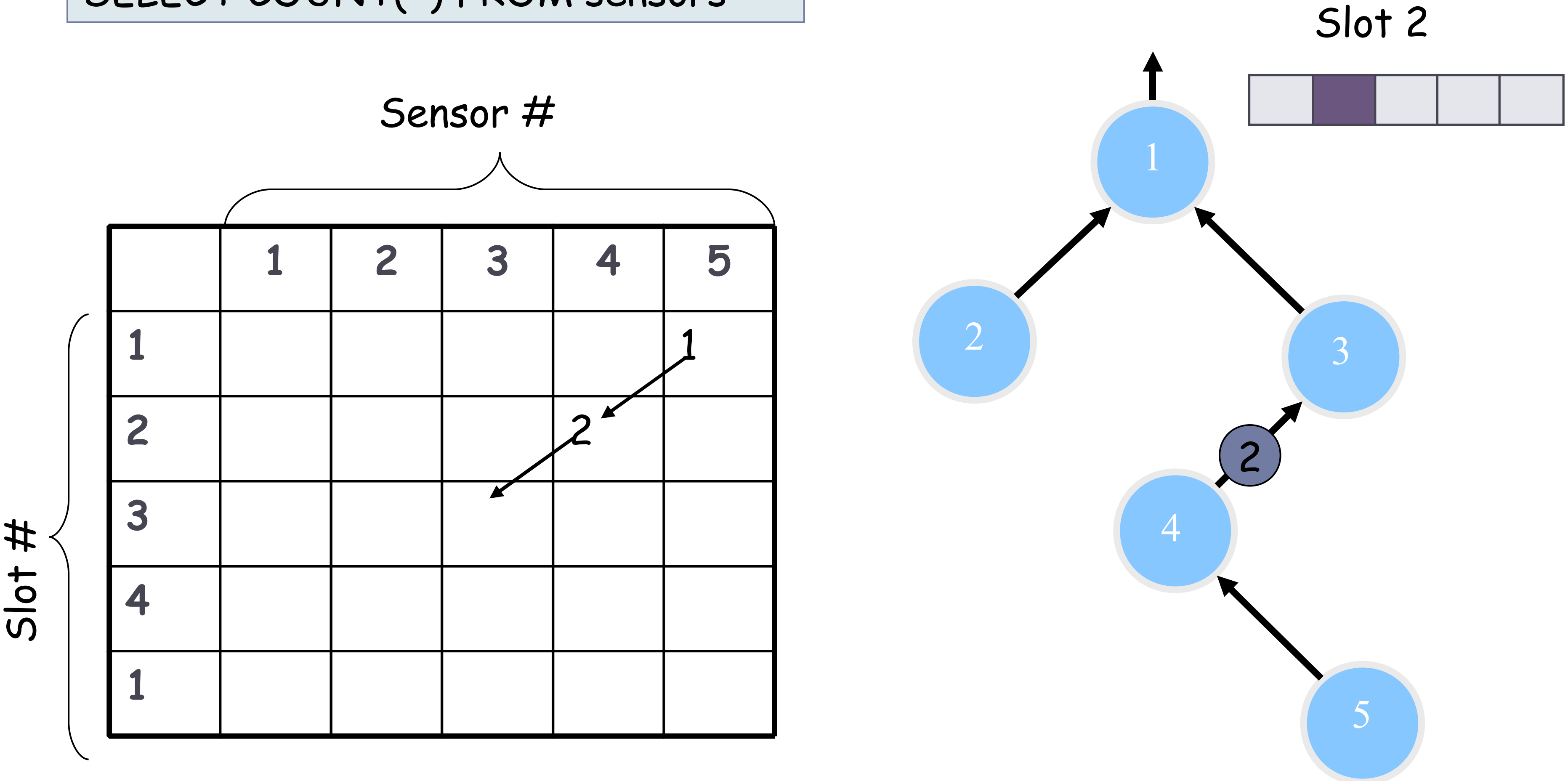
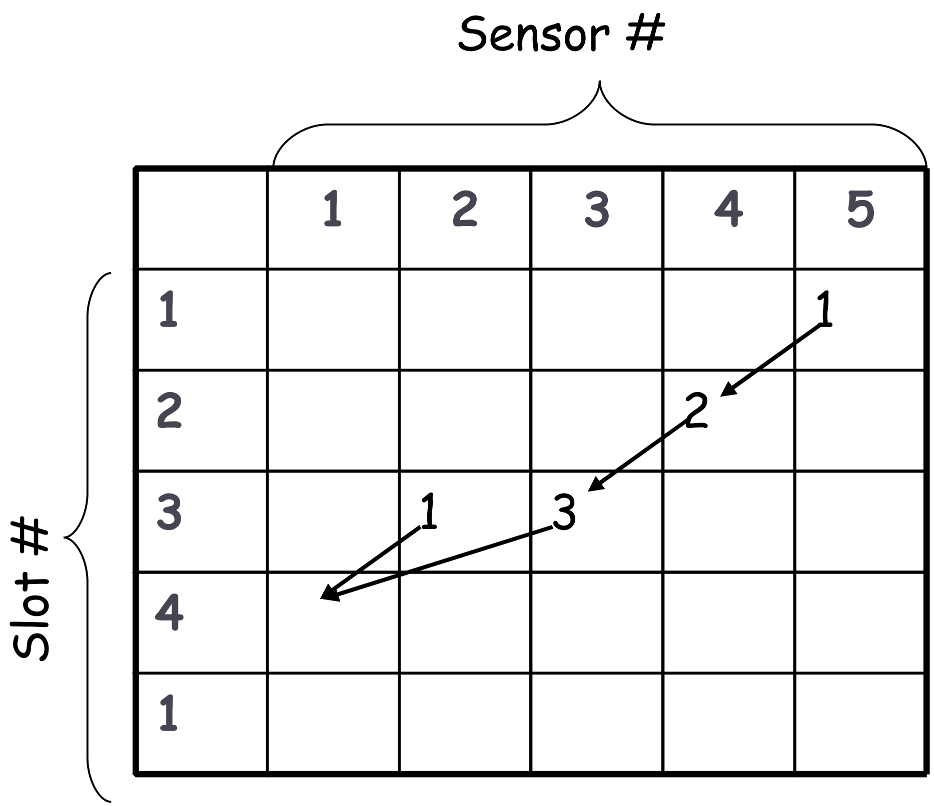


Illustration: Aggregation

SELECT COUNT(*) FROM sensors



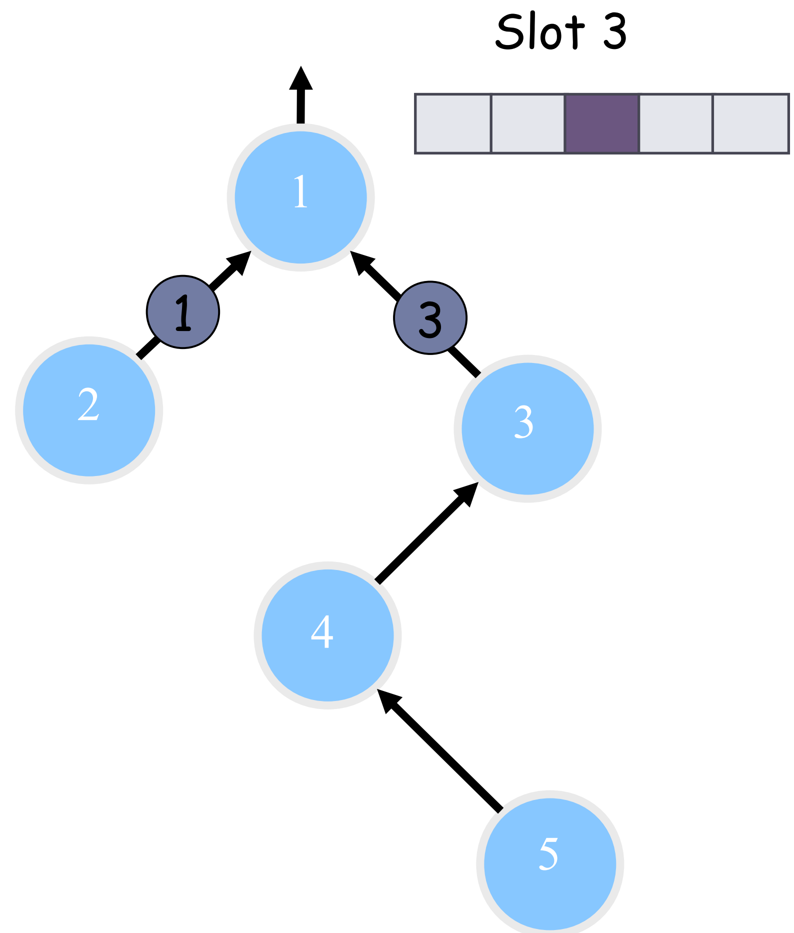


Illustration: Aggregation

SELECT COUNT(*) FROM sensors

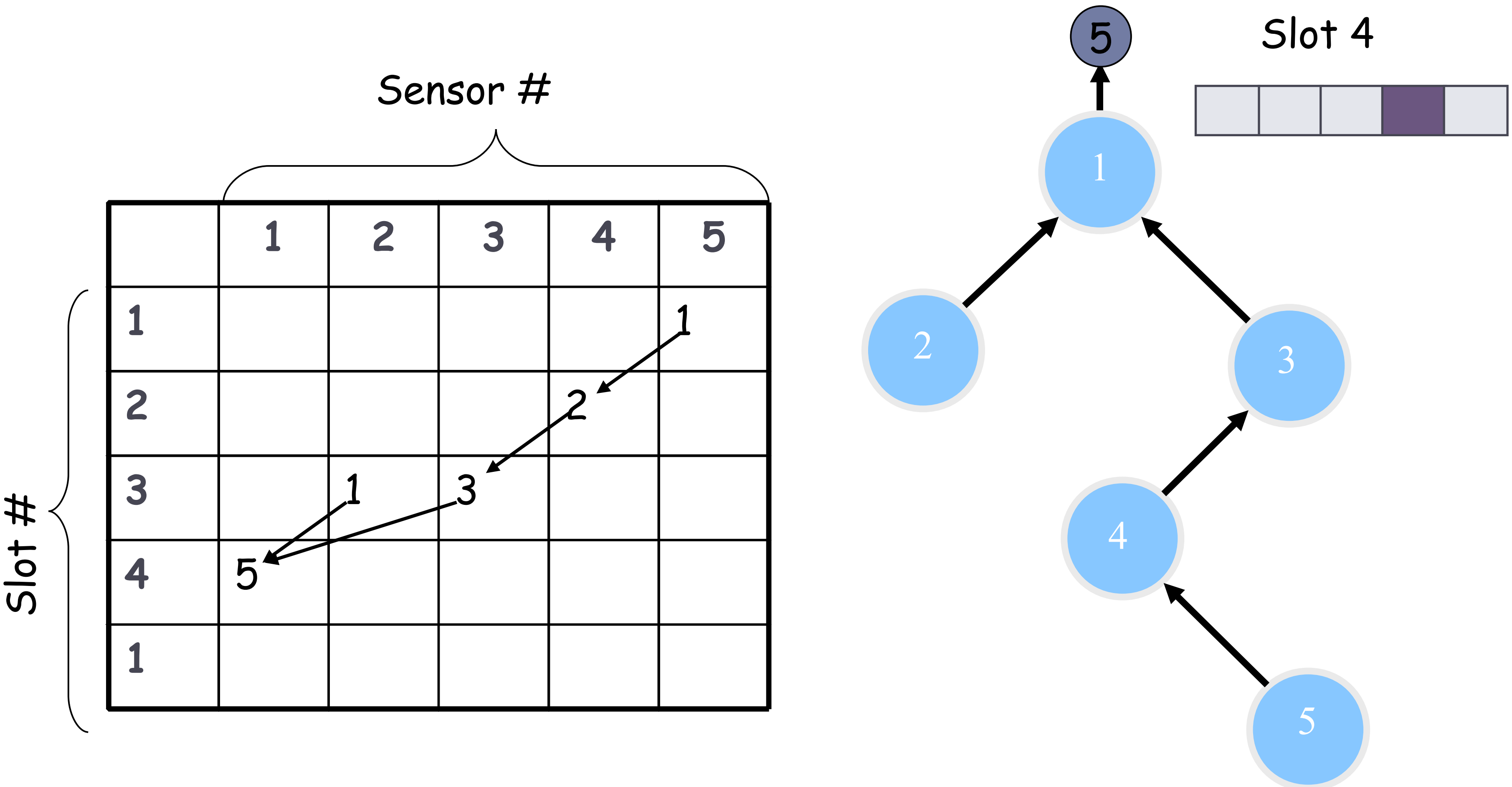


Illustration: Aggregation

SELECT COUNT(*) FROM sensors

Sensor #

Slot #

	1	2	3	4	5
1					1
2				2	
3		1	3		
4	5				
1					1

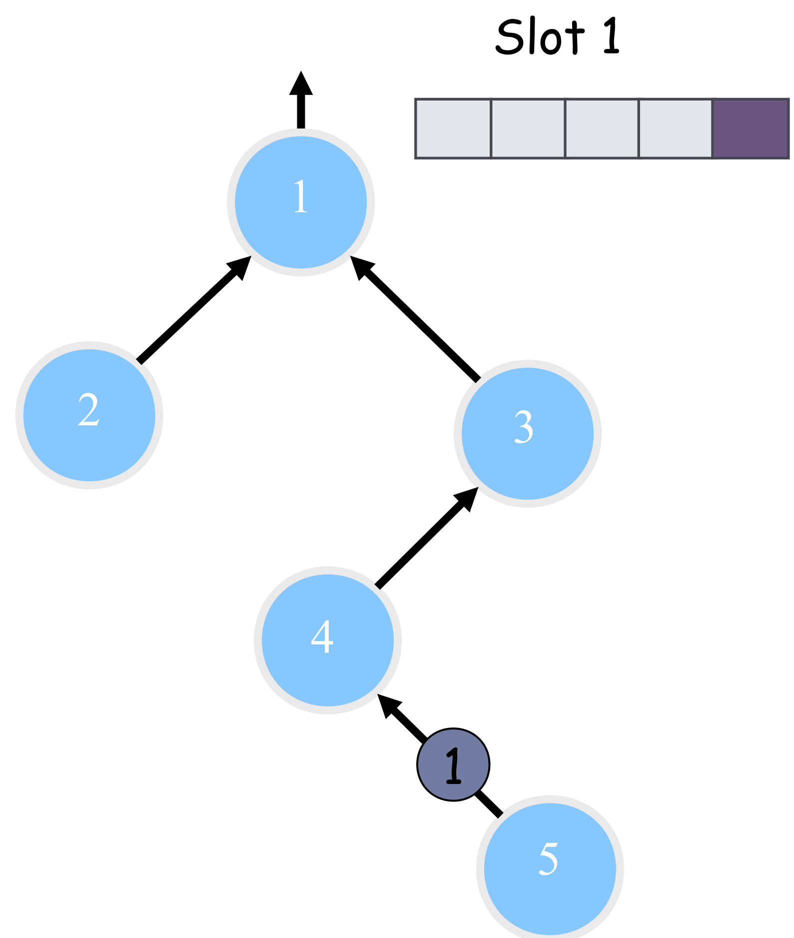
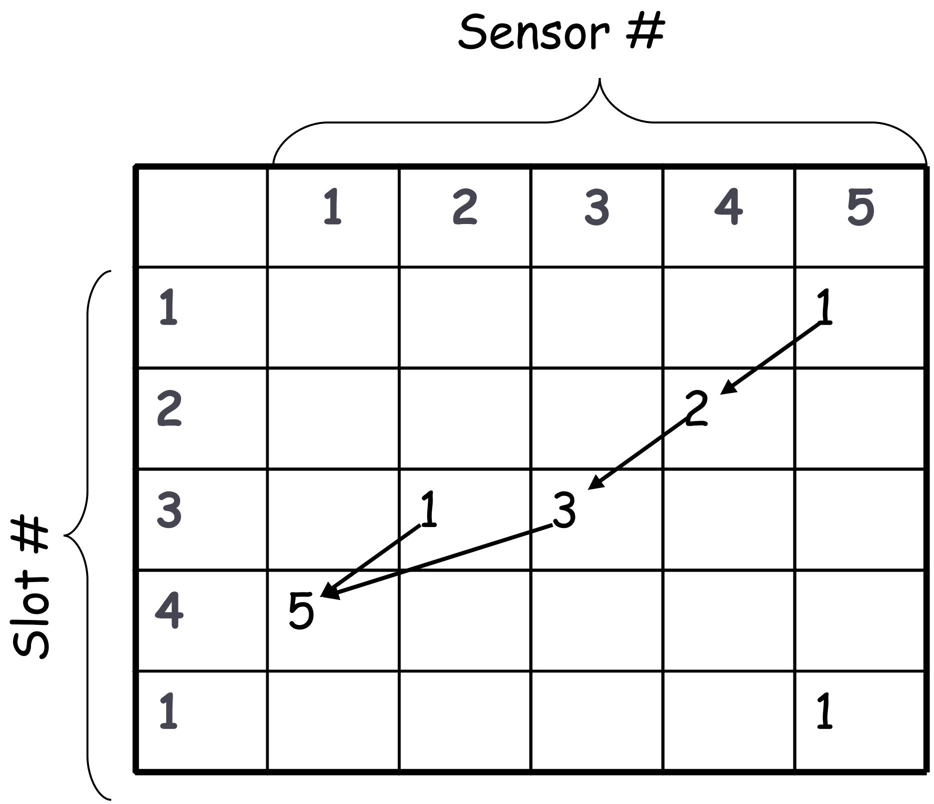
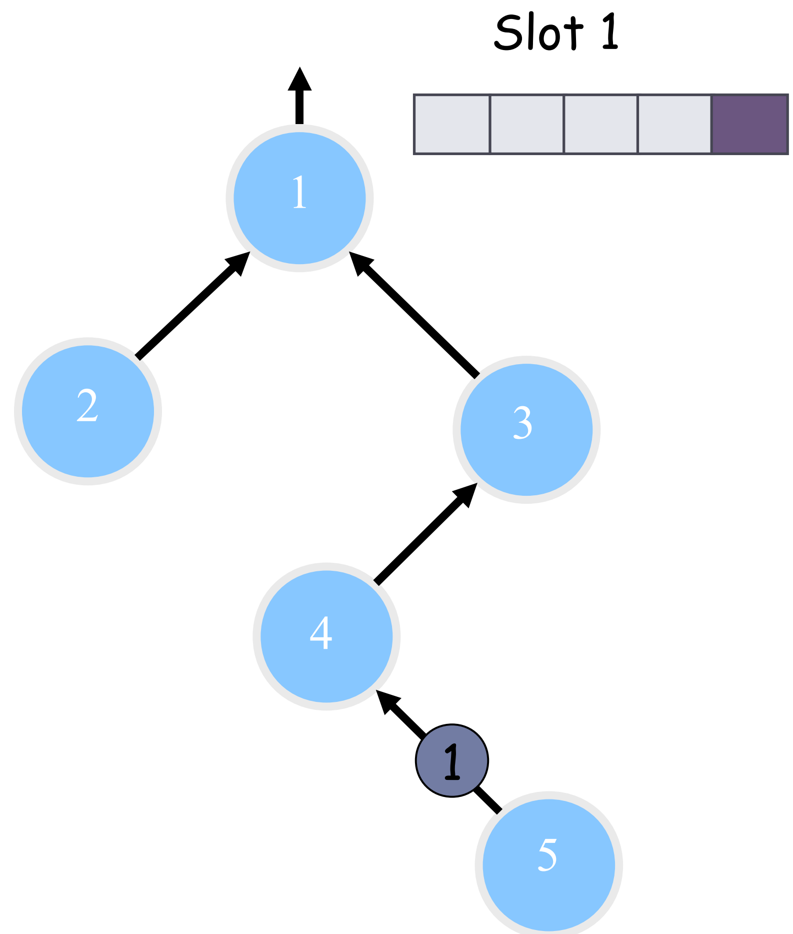


Illustration: Aggregation

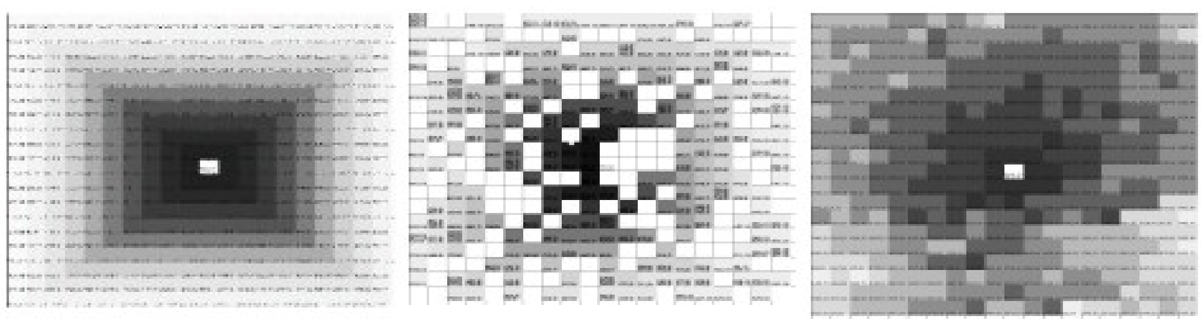
SELECT COUNT(*) FROM sensors





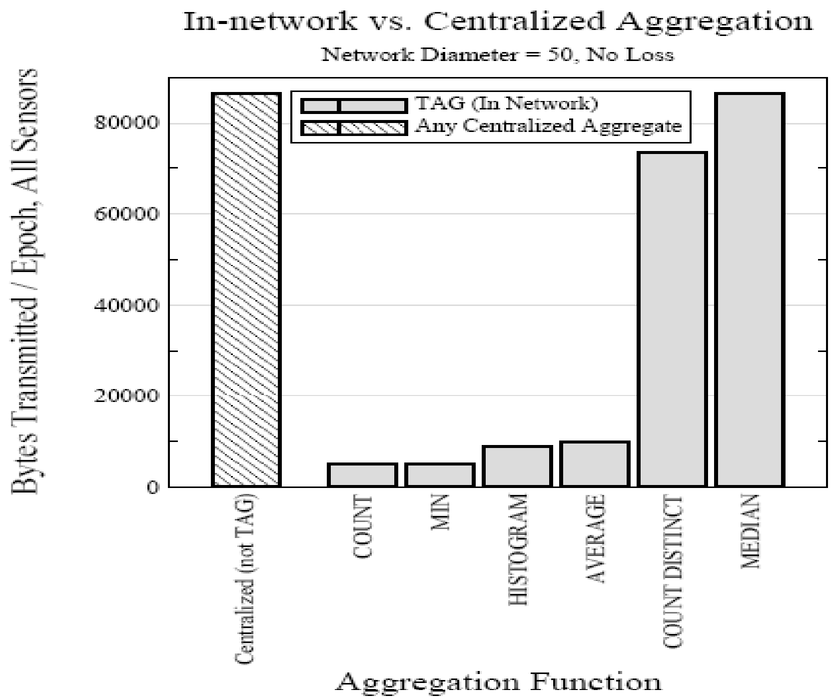
Simulation Based Evaluation (1/2)

- Implemented in Java
- 3 communication models
  - Simple: nodes have perfect lossless communication with regularly placed neighbors
  - Random: Nodes' placement is 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



(a) Simple (b) Random (c) Realistic

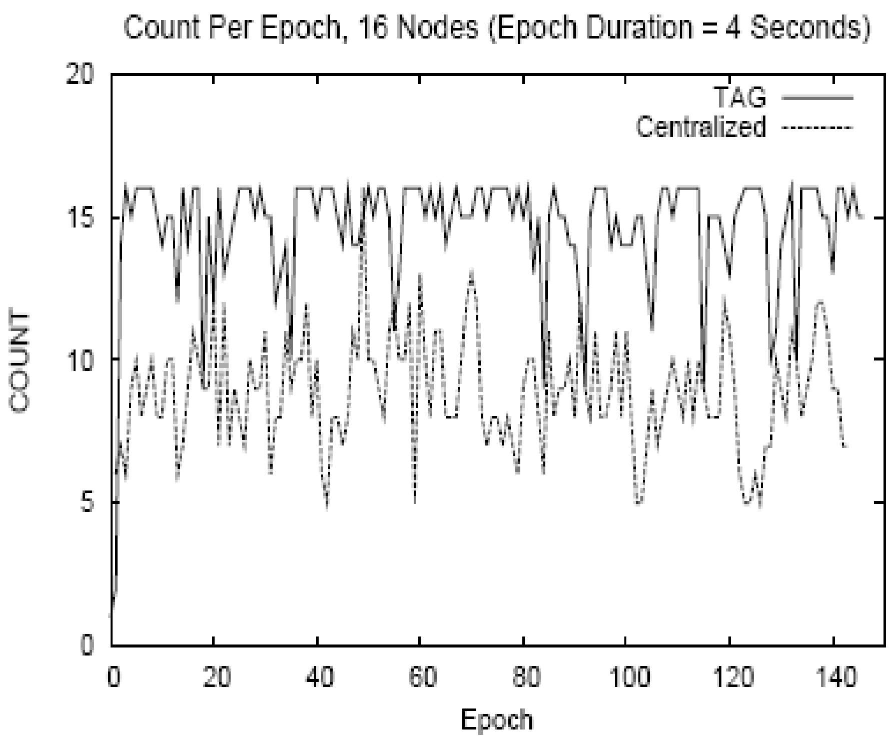
Simulation Based Evaluation (2/2)



- Min & Count: 1 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

TAG Performance in Real World

- 16 nodes, depth 4 tree, COUNT aggregate, 150 4-sec epoch (10 min run)
- No optimization
- Lossy environment
- # of messages (Centralized: 4685, TAG: 2330, 50% comm. Reduction)



TAG Optimizations

- Channel sharing
  - If node misses initial request to aggregate, it can snoop network traffic and “catch up” and include itself
  - In case of MAX, do not broadcast if peer has transmitted a higher value
- Hypothesis Testing
  - Root can provide information that will suppress readings that cannot affect the final aggregate value.
  - Example: MIN must be < 50; nodes with value ≥ 50 need not participate
- 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 (child messages are lost)

Limitations

- TAG is not robust against node or link failure
- Cached results during node failure or disconnections may affect accuracy of the result
- TAG might not perform well if rate of queries is high, as it follows the flood-respond approach
- Message transmission consumes higher power; however power consumption also depends on node density and node layout which was ignored in evaluation
- Single message per node per epoch
  - Message size might increase at higher level nodes
  - Root gets overload
- Trade-off between aggregation and security/privacy
  - In case of privacy, data needs to be encrypted. Aggregation makes each node to do encryption and decryption for each message, which will consume energy

Discussions

- Besides tree topology, what other topology can be considered?
- Correctness issue: How does the user know which nodes are and are not included in an aggregate?
- How to incorporate nested queries?
  - Example: MAX(AVG(1000 readings) @ each node)

Synopsis Diffusion for Robust Aggregation in Sensor Networks

Authors: Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary Anderson

Presented by: Nathan Dautenhahn and Shameem Ahmed

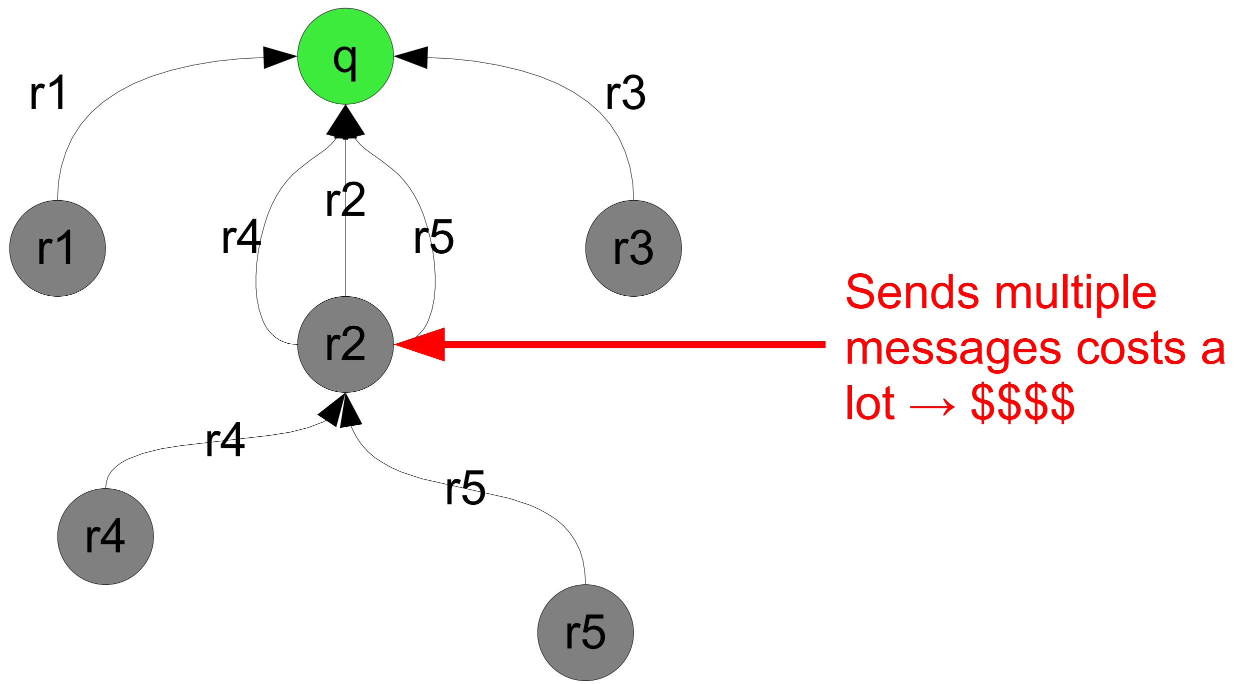
CS 525 Distributed Systems

March 9, 2010

Outline

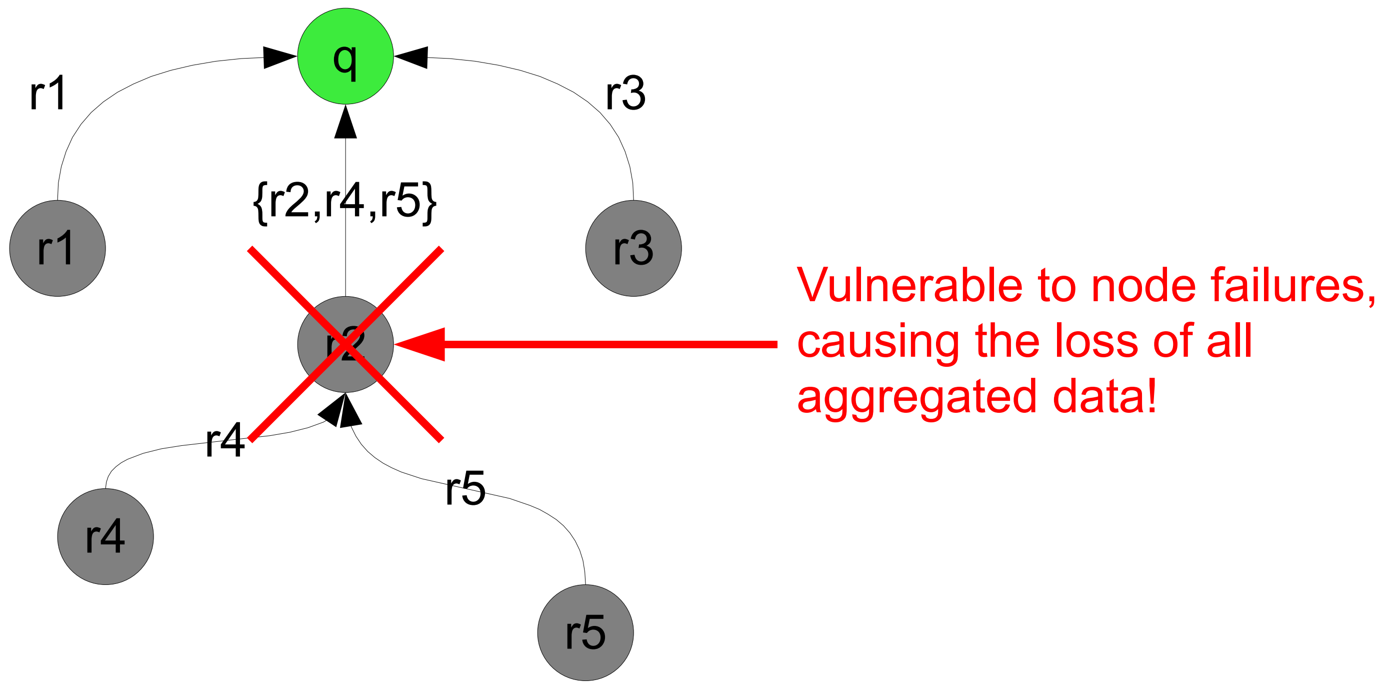
- Problem Definition and Motivation
- 10,000 Foot View of Synopsis Diffusion
  - General Algorithm
  - Concrete Descriptions of Examples
    - Rings
- Formal Framework of ODI Correctness
- Aggregation Algorithms
- Topology Changing – Adaptive Ring
- Evaluation

How do massive wireless sensor networks answer data queries?



Direct routing of answers to query node

An alternative solution is to perform in-network aggregation of results during routing.



In-Network aggregation of data using spanning tree topology and unreliable communications

Two main solutions have been attempted: better topologies and the use of reliable communications

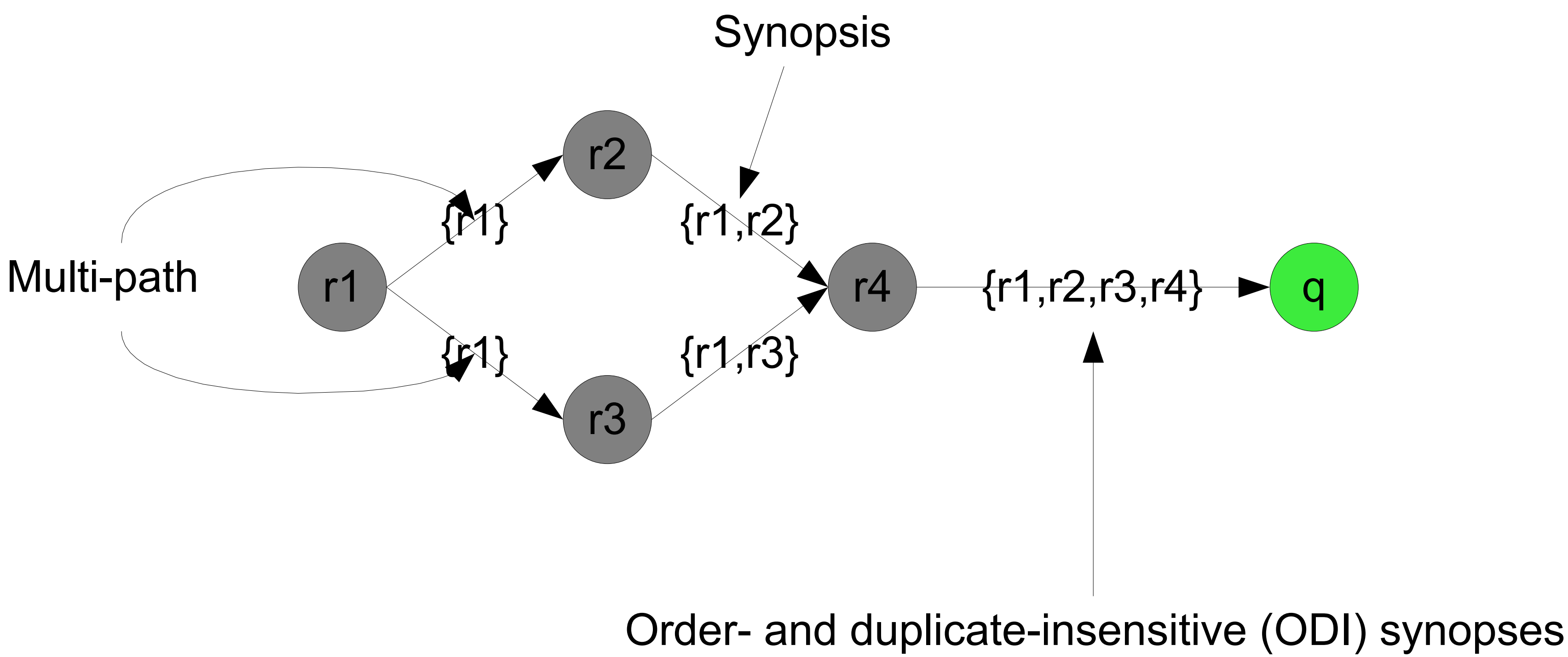
	Reliable Communication	Unreliable Communication
Tree topology	Robust, not energy-efficient For example, Reliable Directed Diffusion [Stann and Heidemann 2003]	Energy-efficient, not robust For example, TAG [Madden et al. 2002a], Directed Diffusion [Intanagonwiwat et al. 2000]
More robust topology	Robust, not energy-efficient For example, Gossip [Kempe et al. 2003]	Energy-efficient and robust For example, Synopsis diffusion (this article)

Gupta et al. as well

Multipath Routing Fails Too → Duplicate Aggregate Answers

[Image: Nath et al. Synopsis Diffusion 08]

Synopsis Diffusion combines multi-path routing with order and duplicate-insensitive synopses.

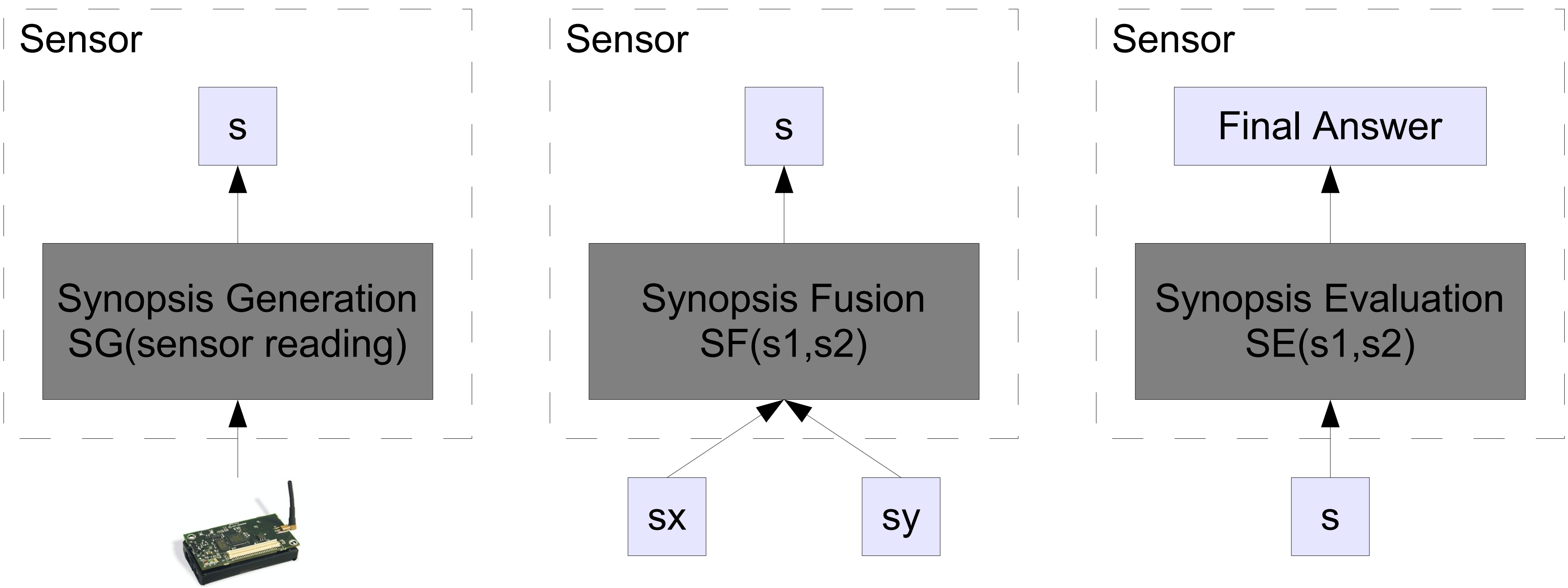


Only 1 message sent per node as broadcast

There are three generic functions on synopses:

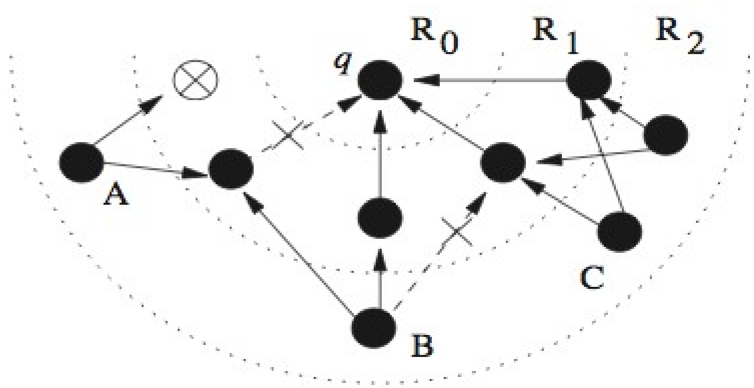
All sensor nodes perform SG() for their local reading

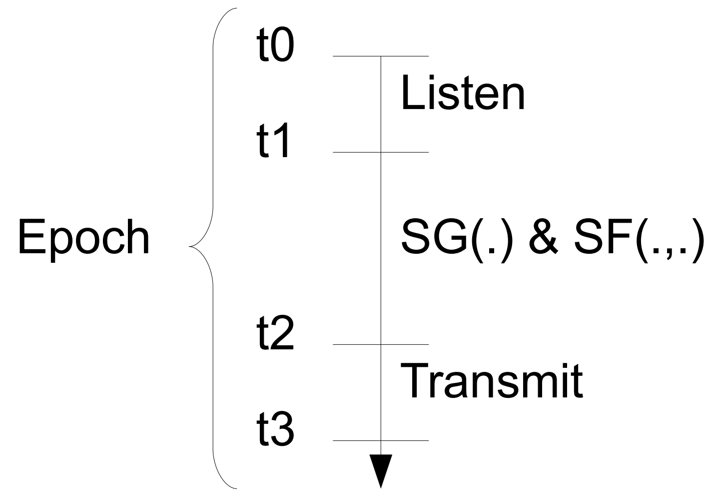
The synopses sx and sy can be any combination provided by an output of SG(.) or SF(.,.)



The functions SG(.), SF(.,.), and SE(.) depend upon the specific aggregation scheme.

Synopsis Diffusion on a rings overlay network provides a more concrete description.





[Image: Nath et al. Synopsis Diffusion 08]

The count algorithm approximates the total number of live sensor nodes in a sensor network.

Derived from the Flajolet and Martin's algorithm (FM) for counting distinct elements in a multiset.

Synopsis: $\begin{pmatrix} b1 \\ b2 \\ b3 \\ . \\ . \\ bk \end{pmatrix}$

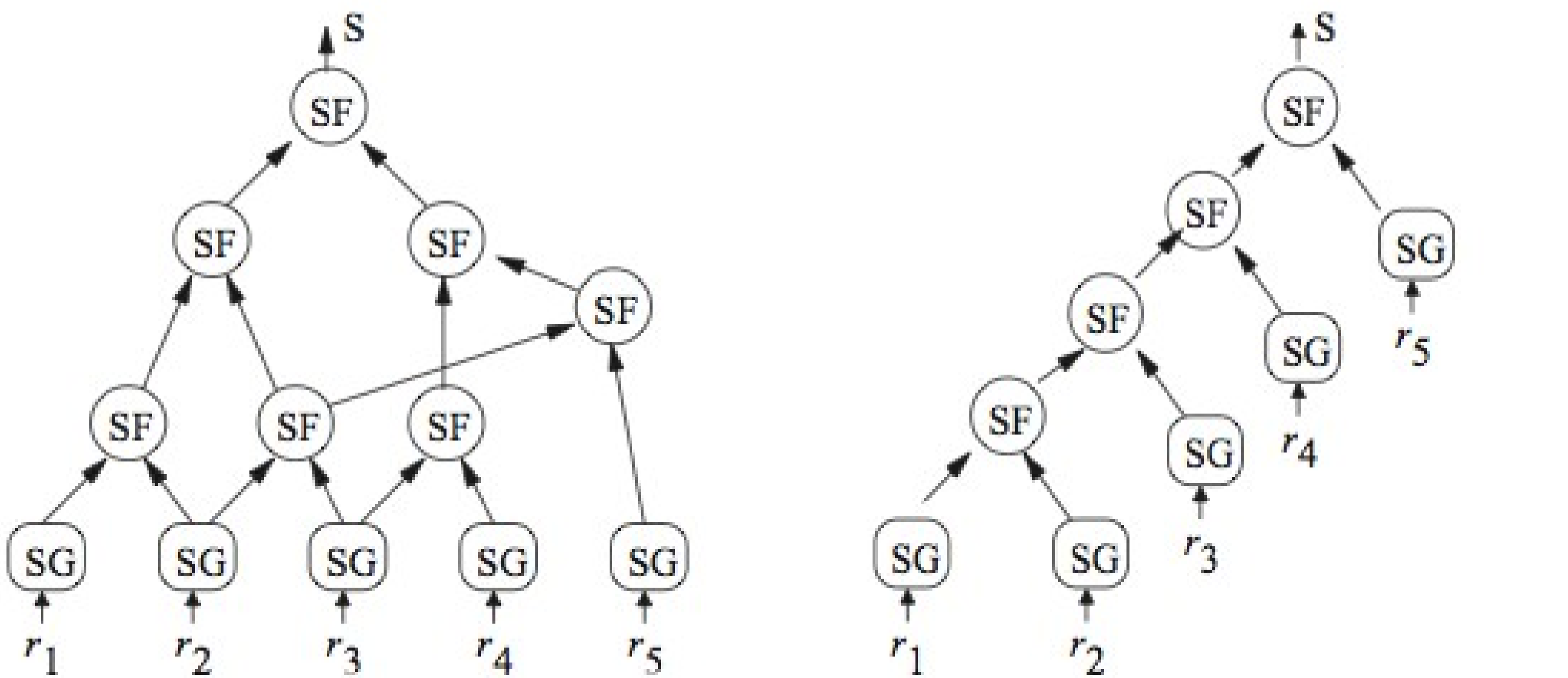
SG(): output a bit vector s of length k, with CT(k)th bit set.

SF(s,s'): Output a bitwise OR of s and s'

SE(s): Return 2^(i-1)/0.77351, where i = lowest order bit not set

ODI-Correctness exists when all potential synopsis combinations produce the same result.

Definitions: sensor reading, synopsis computation, aggregation DAG, Edge e, synopsis label function, projection operator



(a) Aggregation DAG (b) Canonical left-deep tree

Proof is unbounded!

[Image: Nath et al. Synopsis Diffusion 08]

In order to prove ODI-correctness one must only prove the following four properties hold.

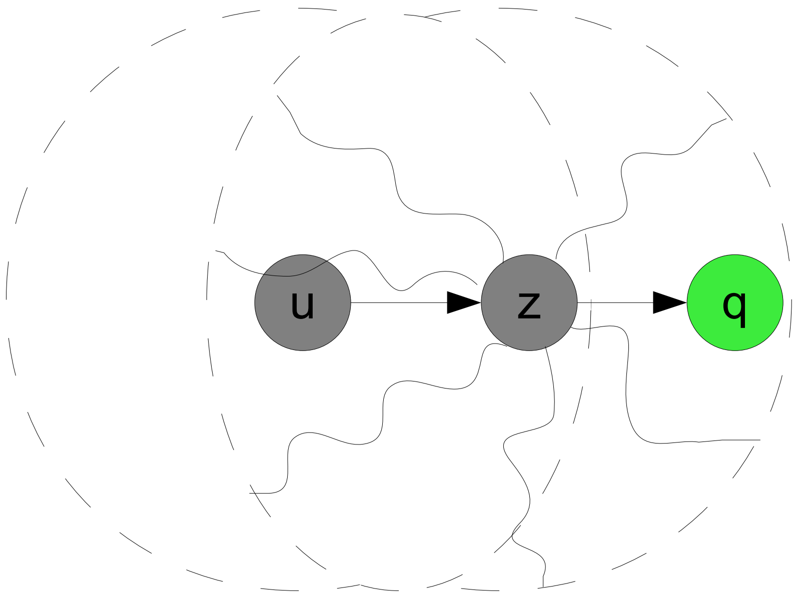
1. SG() preserves duplicates
2. SF() is communicative
3. SF() is associative
4. SF() is same synopsis indempotent

Much easier than proving the unbounded DAG problem!

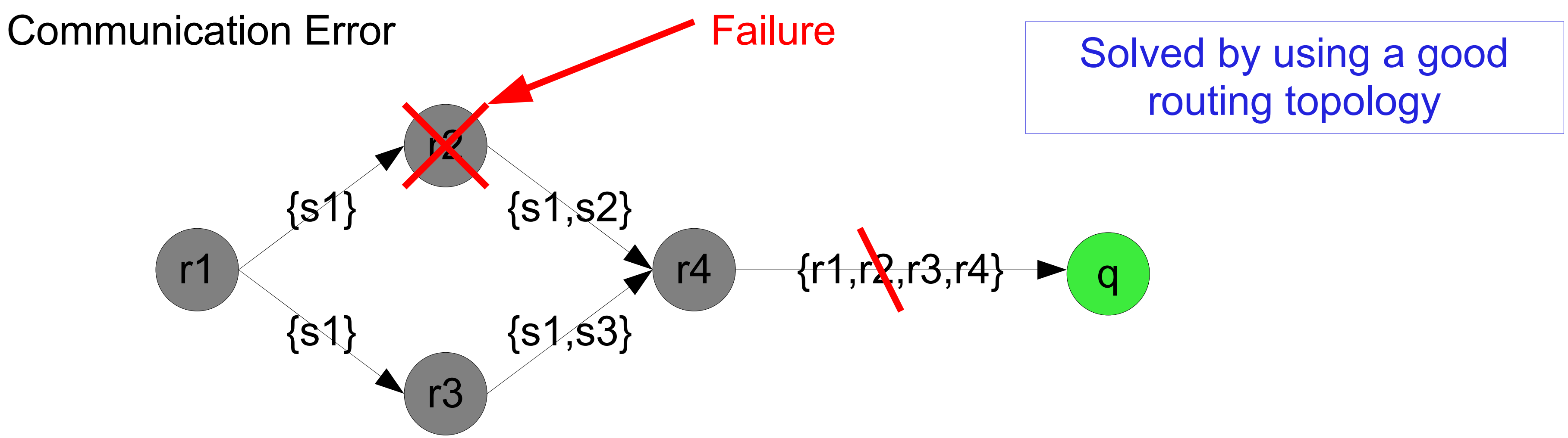
An ODI-Correct synopsis diffusion algorithm results in a semi-lattice structure.

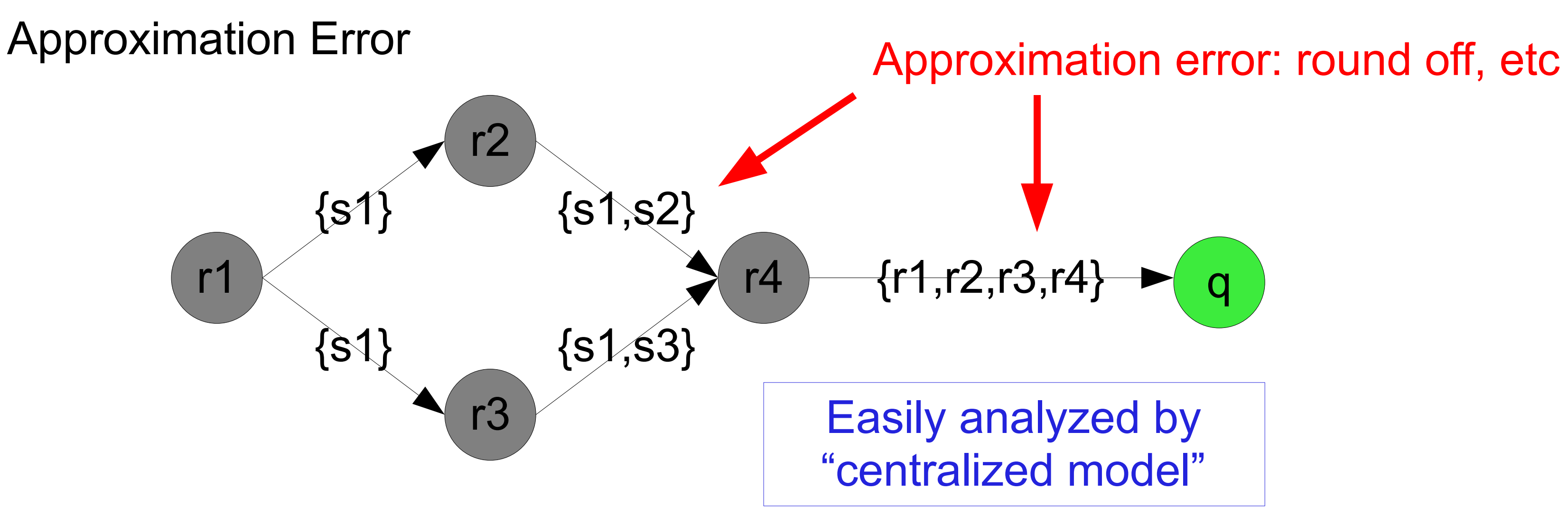
$$\text{if } z = SF(x, y)$$
$$\text{then } SF(x, z) = z,$$
$$SF(y, z) = z$$

Implies that the use of ODI synopsis provides an implicit acknowledgement of message success.



There are two types of errors that can occur when using a synopsis diffusion algorithm.



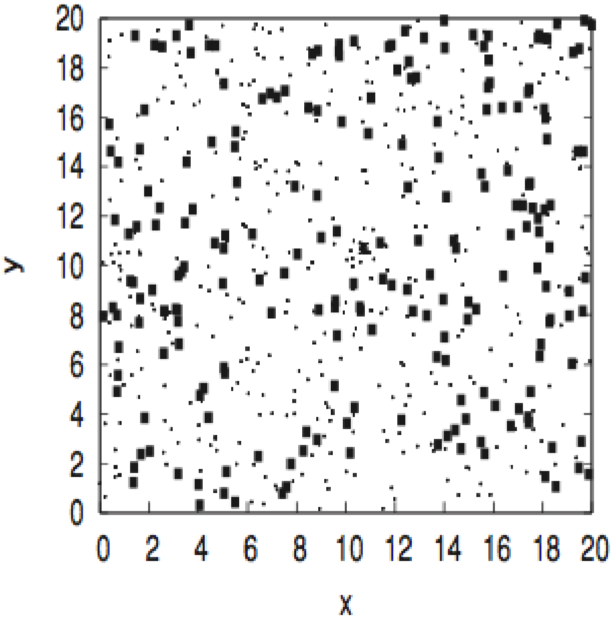


The authors provide several ODI-Correct synopsis diffusion algorithms for different types of aggregation.

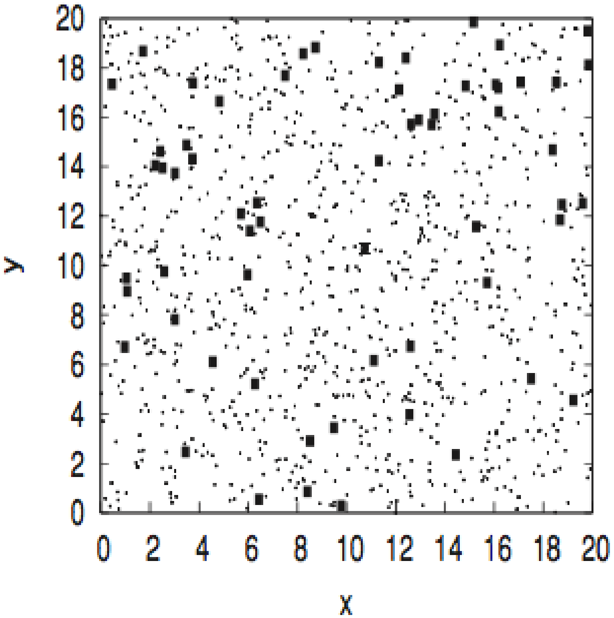
Aggregate
Maximum, Minimum
Count, Count Distinct
Sum, Average, Standard Deviation, Second Moment
Uniform Sample
Mean, *k*th Statistical Moments
Medium, Quantiles
Frequent Items
Range Aggregates, Inner Product Queries

[Image: Nath et al. Synopsis Diffusion 08]

Implicit acknowledgements allow for the automated adaptation of the routing topology.



(a) Rings



(b) Adaptive Rings

[Image: Nath et al. Synopsis Diffusion 08]

Evaluation methodology is to simulate and evaluate performance.

- Topology
  - Querying node at center of grid
- Aggregation Schemes
  - 3 separate for the first experiment, and only one for subsequent experiments
- Message size: 48-byte
- Transmission Model:
  - TAG simulator based on empirical data
- Accuracy: Root Mean Square (RMS)
- Power Consumption: Only include communications

Does this affect the performance of the other algorithms such as gossip base?

Limits the breadth of their evaluation

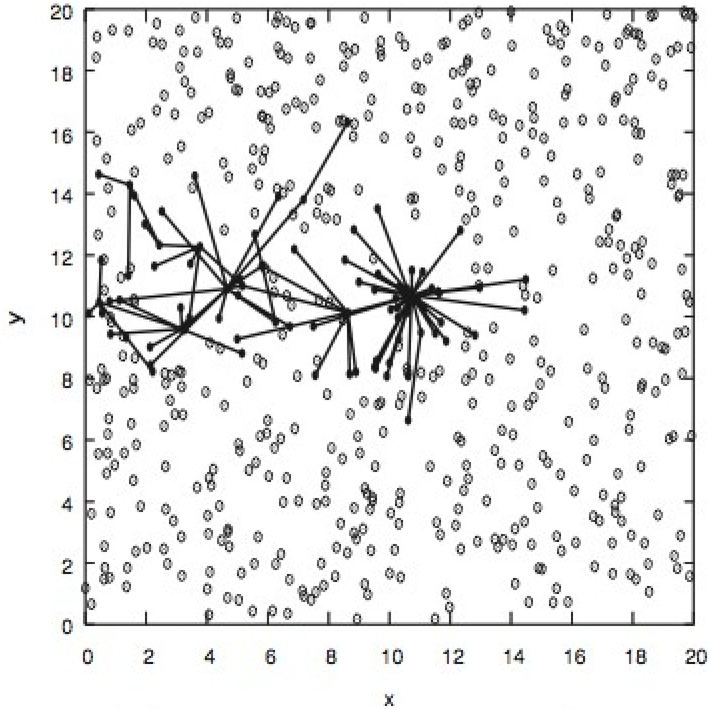
Is a theoretical description enough to not include computation issues?

Authors use an algorithm for sum from a primary competing paper by Considine et al.

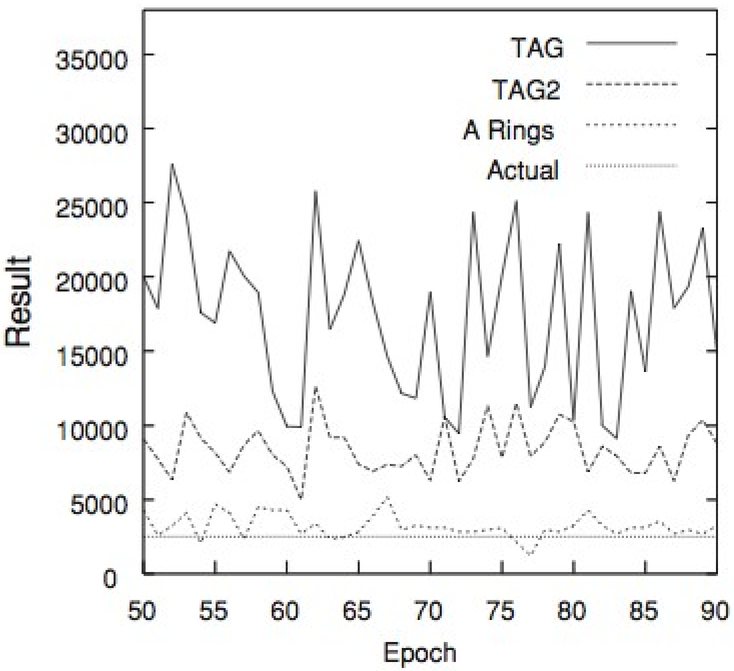
- How does this effect the results?
- Is it bad that they use this algorithm?
- Is it bad that they only mention this in one line of the related works section, and not in the evaluation?

Evaluation: comparison of aggregation schemes.

Scheme	% Nodes	Error (uniform)	Error (skewed)	Error (Gaussian)
TAG	< 15%	0.87	0.99	0.94
TAG2	N/A	0.85	0.98	0.92
GOSSIP	N/A	0.91	0.99	0.93
RINGS	65%	0.33	0.19	0.21
ADAPT. RINGS	95%	0.15	0.16	0.15
FLOOD	≈ 100%	0.13	0.13	0.13



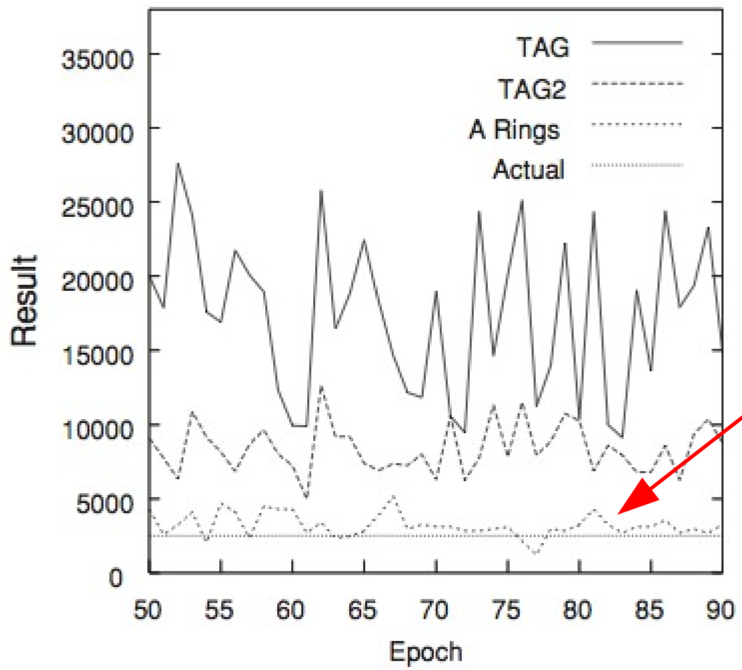
(a)Nodes accounted for in TAG



(b)Avg computed by different schemes

[Image: Nath et al. Synopsis Diffusion 08]

Evaluation: comparison of aggregation schemes.



(b)Avg computed by different schemes

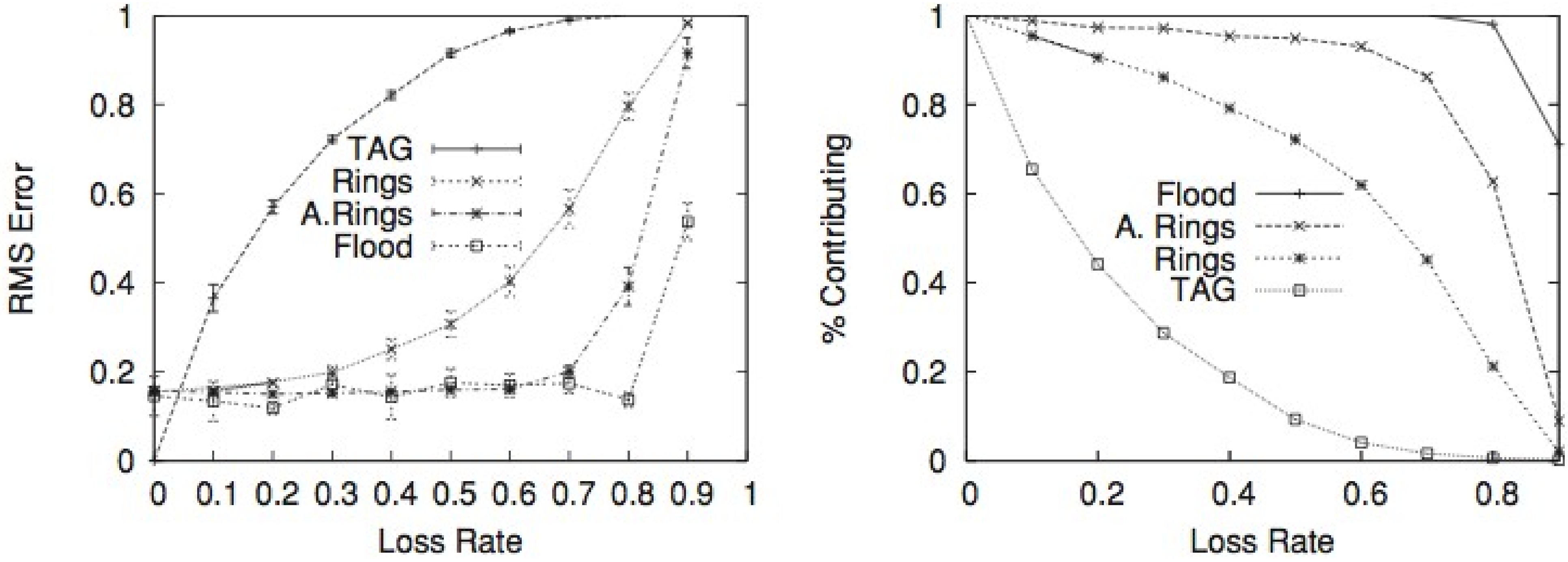
If they are using ODI and are duplicate insensitive: Is it okay to be using approximation algorithms?

Notice that they are still 200% off in some cases.

Is 200% realistic?

[Image: Nath et al. Synopsis Diffusion 08]

Evaluation: effect of communication/packet loss.

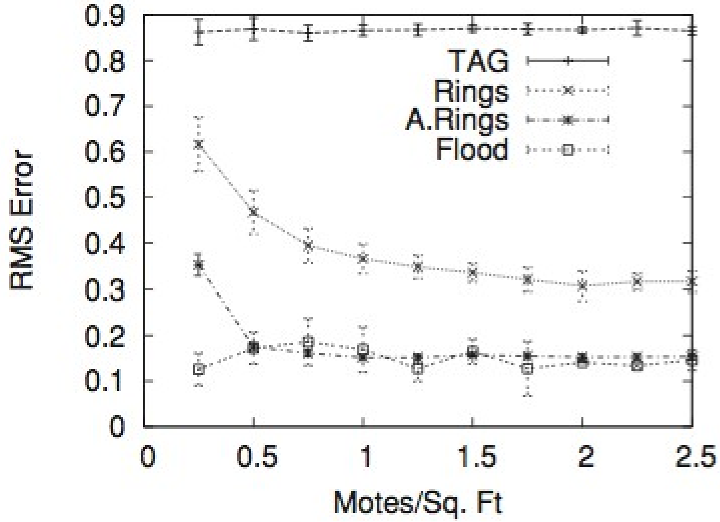


(a) RMS error

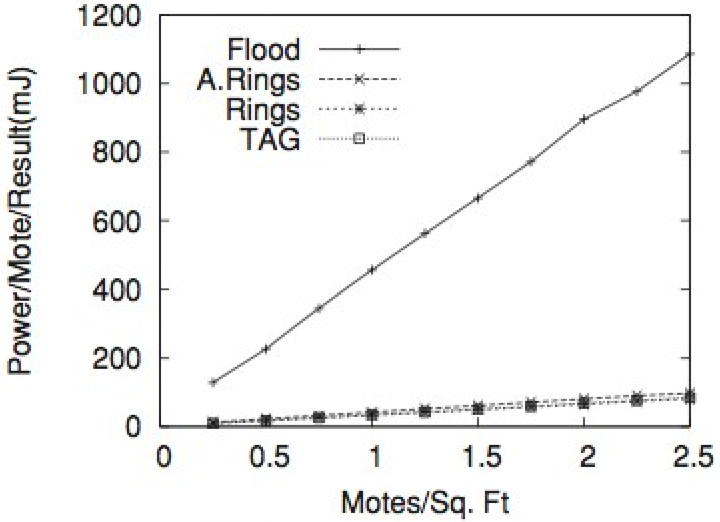
(b) % values included

[Image: Nath et al. Synopsis Diffusion 08]

Evaluation: effect of deployment density.



(a) RSM error



(b) Power consumption

[Image: Nath et al. Synopsis Diffusion 08]

This paper has the following major contributions:

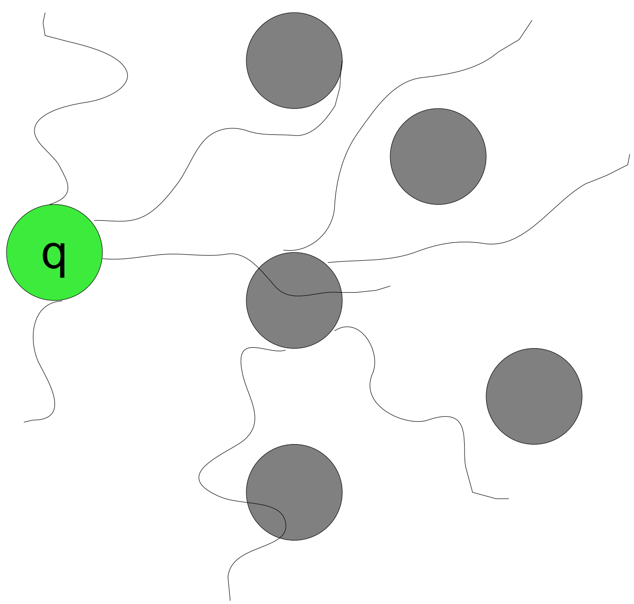
- A general framework to perform synopsis diffusion algorithm development and evaluation.
- Rings overlay topology, and subsequently adaptive rings topology
- Showed that they can develop algorithms for several aggregation schemes, which is better than related works.
- Successful separation of routing and aggregation mechanisms.

Discussion Questions

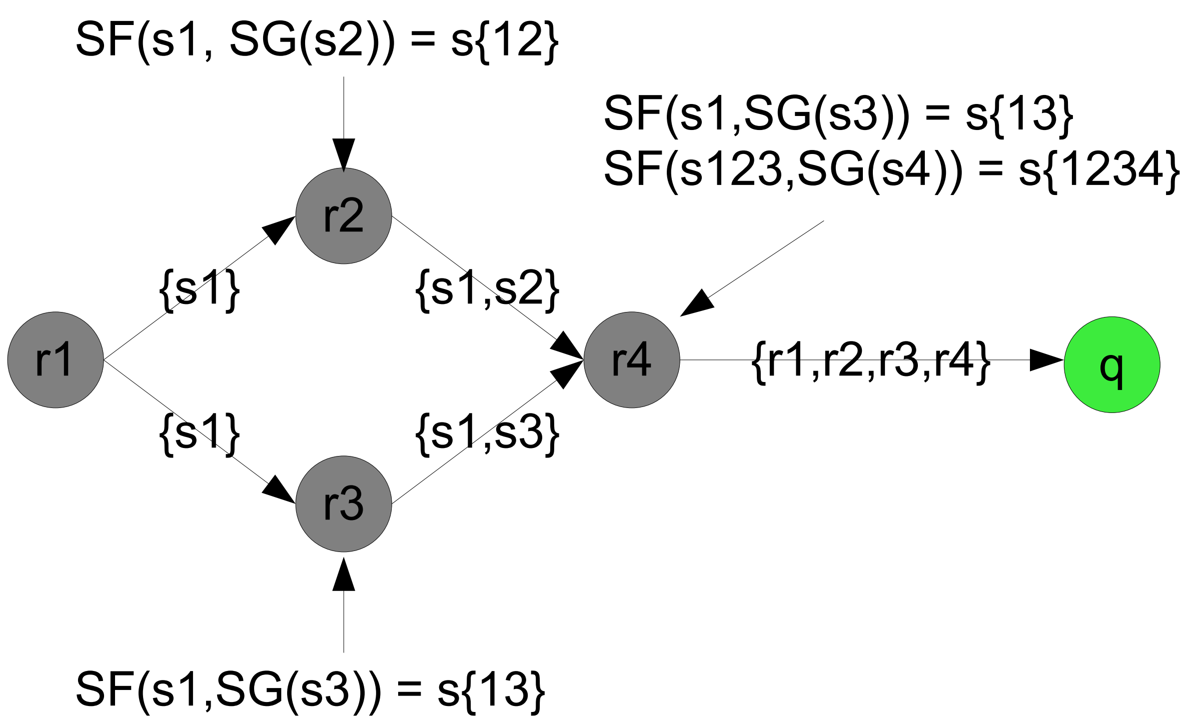
- Synopsis Diffusion requires a specific implementation for each routing scheme and aggregation mechanism
  - What are the limitations of this approach with respect to scalability and flexibility?
- How much of this work is practical? Intuitively it seems as though there should be higher costs in computation.
- Is a 20% loss rate okay for a real application?

There are two phases in a synopsis diffusion algorithm: distribution and aggregation phases

Distribution



Aggregation



Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks

Authors: Philip Levis, Neil Patel, David Culler, Scott Shenker

Presented by: Nathan Dautenhahn and Shameem Ahmed

CS 525 Distributed Systems

March 9, 2010

Trickle is a solution to the problem of how to perform efficient code updates in a wireless sensor network.

- Primary motivations:
  - Large scale, must minimize transmission costs
  - Application specific transfer protocol
  - High transient loss patterns
  - Instability of motes
  - Cost of propagating code as well as the maintenance for performing propagation is costly
  - Maintenance costs exceed code propagation cost

The properties of an efficient sensor network are as follows:

- Low maintenance costs
- Rapid propagation
- Scalability

Trickle uses a "polite gossip" protocol to exchange code metadata for low cost maintenance.

- Periodically transmits code metadata if it has heard no such meta data within a given time period
- All messages are sent via broadcast
- Guaranteed code propagation if every mote:
  - Receives or transmits data periodically
  - Some motes communicate at a threshold minimum "communication rate"
- In a lossless single hop network of size n, the communication rate is 1/n

The Trickle code propagation routing algorithm:

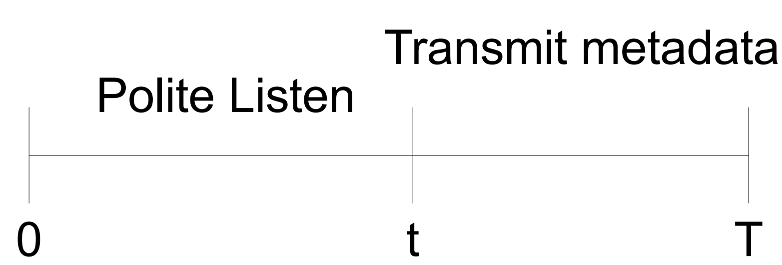
```
--- Polite Listening and Response ---

If motex receives metadata == motex_metadata:
      c++
If motex has update for code_x-y:
      Broadcast code_x
If motex needs code_x+y:
      Broadcast code_x metadata to receive update from y

---
Init: c = 0; k = 1 or 2; t = [0, T]

If c < k at time t:
      Broadcast motex_metadata

If t==T:
      c = 0
      T = rand(0,T)
```



Overcoming basic assumptions:

No Packet Loss

Grows with density of network at O(log(n))

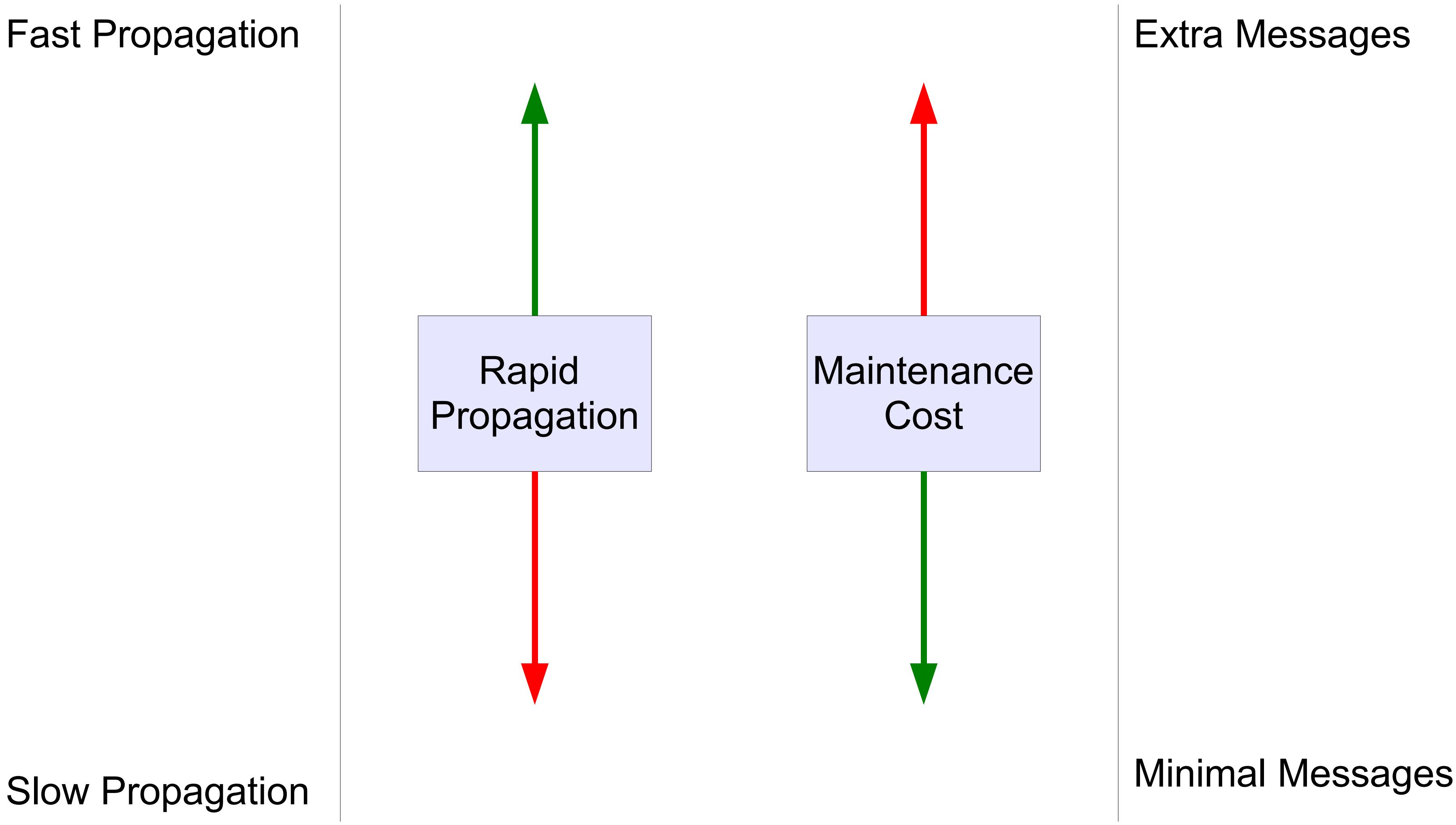
Perfect Time Synchronization

O(sqrt(n))

- Short listen problem
- Listen only period

Single-hop Network

Automated variation of T parameter to allow for rapid propagation with minimal maintenance cost



Discussion

- No code propagation: Will this skew the results at all? Will they just scale up?
- To send code requires a broadcast message: How do we deal with 100 motes responding with updates?
- Trickle scales to approximately 1000 motes, is this enough?
- Is the simplicity and success of Trickle worth the broadcast costs?