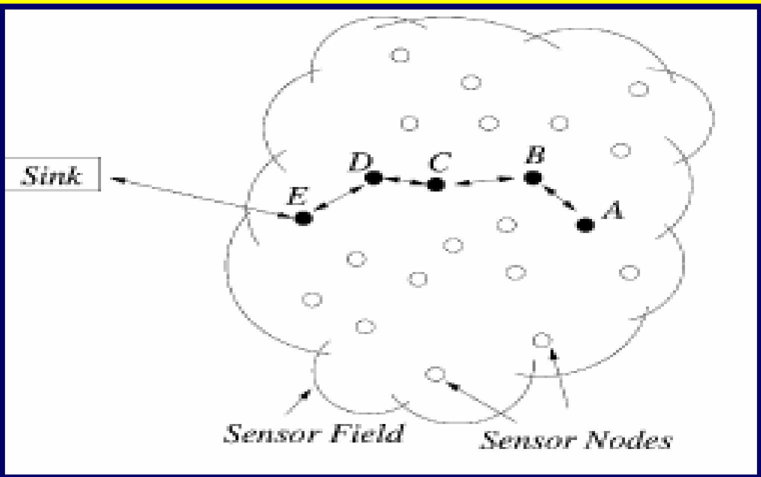
Approximate Aggregation Techniques for Sensor Databases

John Byers
Computer Science Department
Boston University

Joint work with Jeffrey Considine, George Kollios and Feifei Li

Sensor Network Model



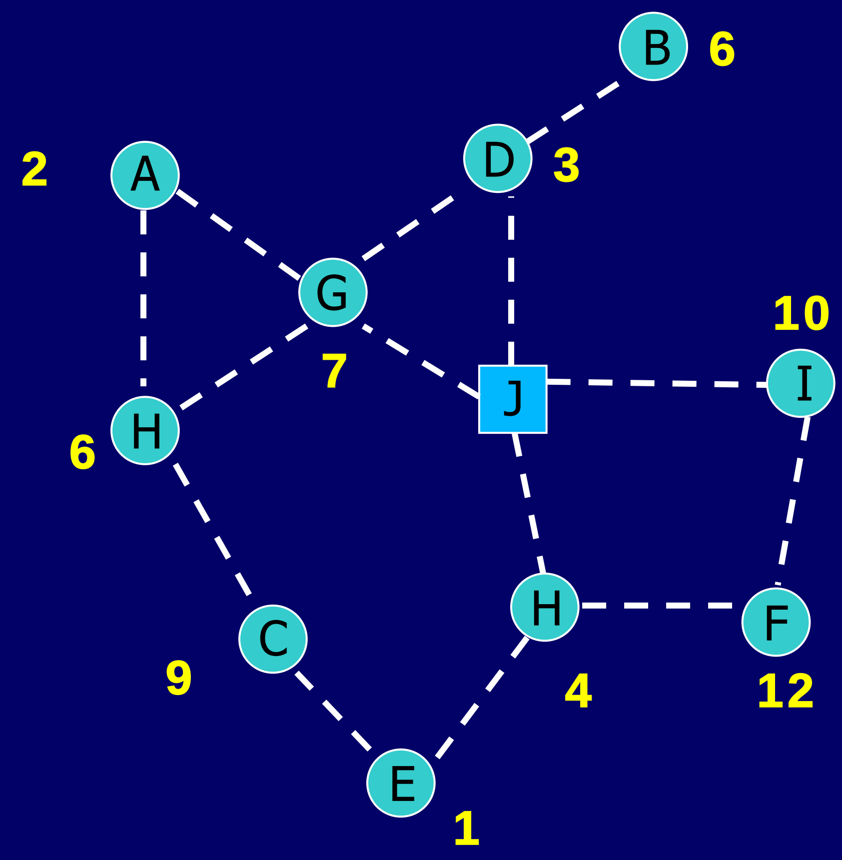
- Large set of sensors distributed in a sensor field.
- Communication via a wireless ad-hoc network.
- Node and links are failure-prone.
- Sensors are resource-constrained
  - Limited memory, battery-powered, messaging is costly.

Sensor Databases

- Useful abstraction:
  - Treat sensor field as a distributed database
    - But: data is gathered, not stored nor saved.
  - Express query in SQL-like language
    - COUNT, SUM, AVG, MIN, GROUP-BY
  - Query processor distributes query and aggregates responses
  - Exemplified by systems like TAG (Berkeley/MIT) and Cougar (Cornell)

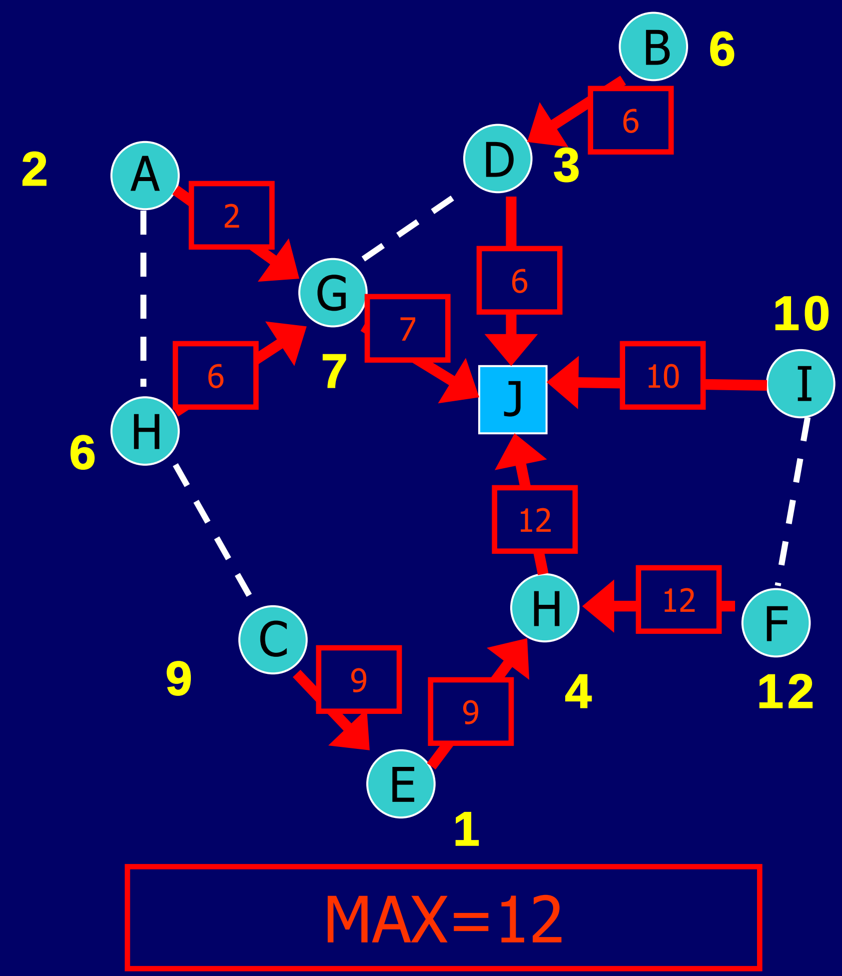
A Motivating Example

- Each sensor has a single sensed value.
- Sink initiates one-shot queries such as: What is the…
  - maximum value?
  - mean value?
- Continuous queries are a natural extension.



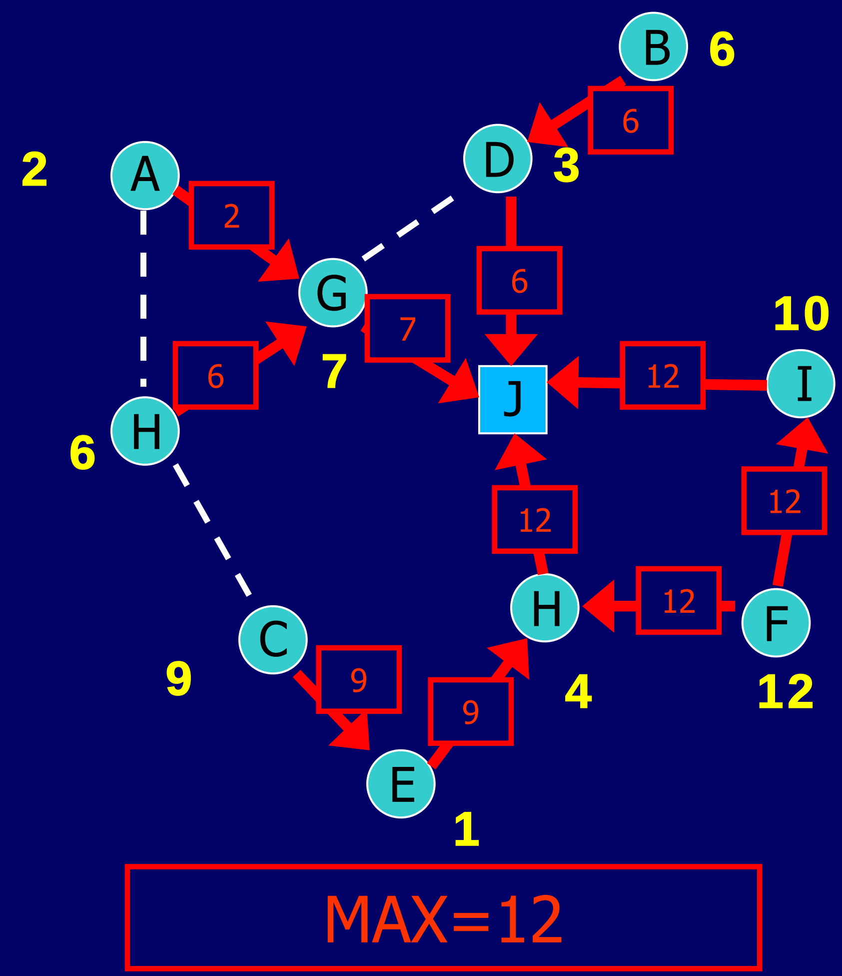
MAX Aggregation (no losses)

- Build spanning tree
- Aggregate in-network
  - Each node sends one summary packet
  - Summary has MAX of entire sub-tree
- One loss could lose MAX of many nodes
  - Neighbors of sink are particularly vulnerable



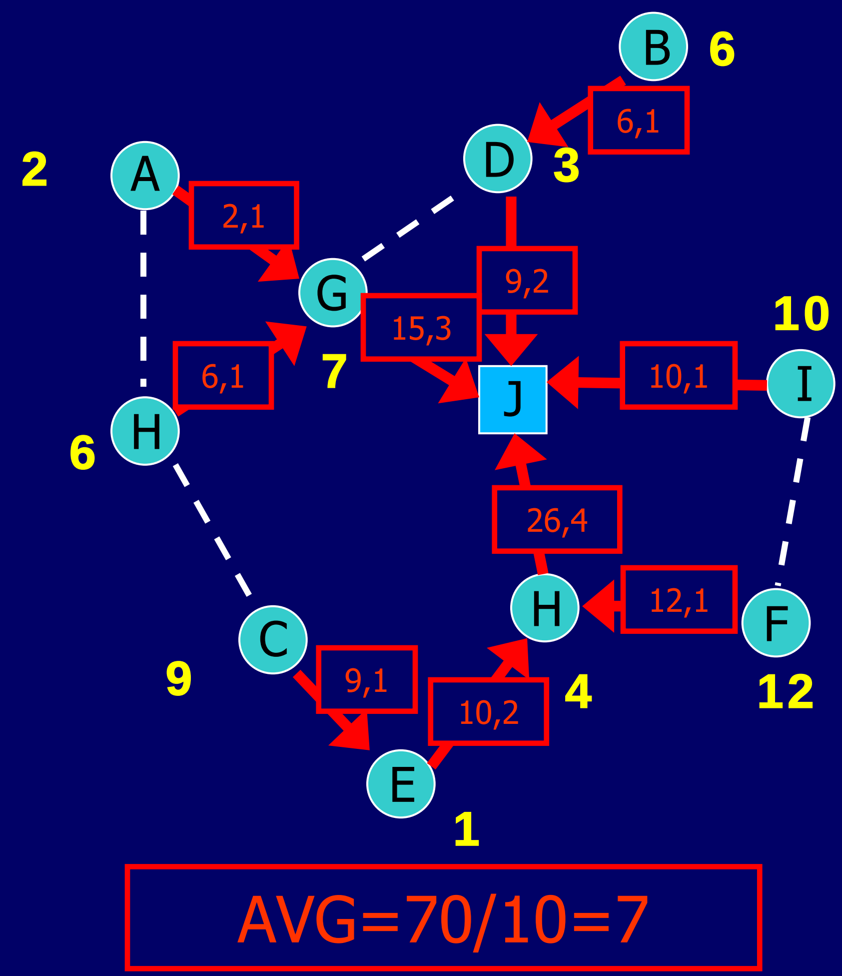
MAX Aggregation (with loss)

- Nodes send summaries over multiple paths
  - Free local broadcast
  - Always send MAX value observed
- MAX is “infectious”
  - Harder to lose
  - Just need one viable path to the sink
- Relies on duplicate-insensitivity of MAX



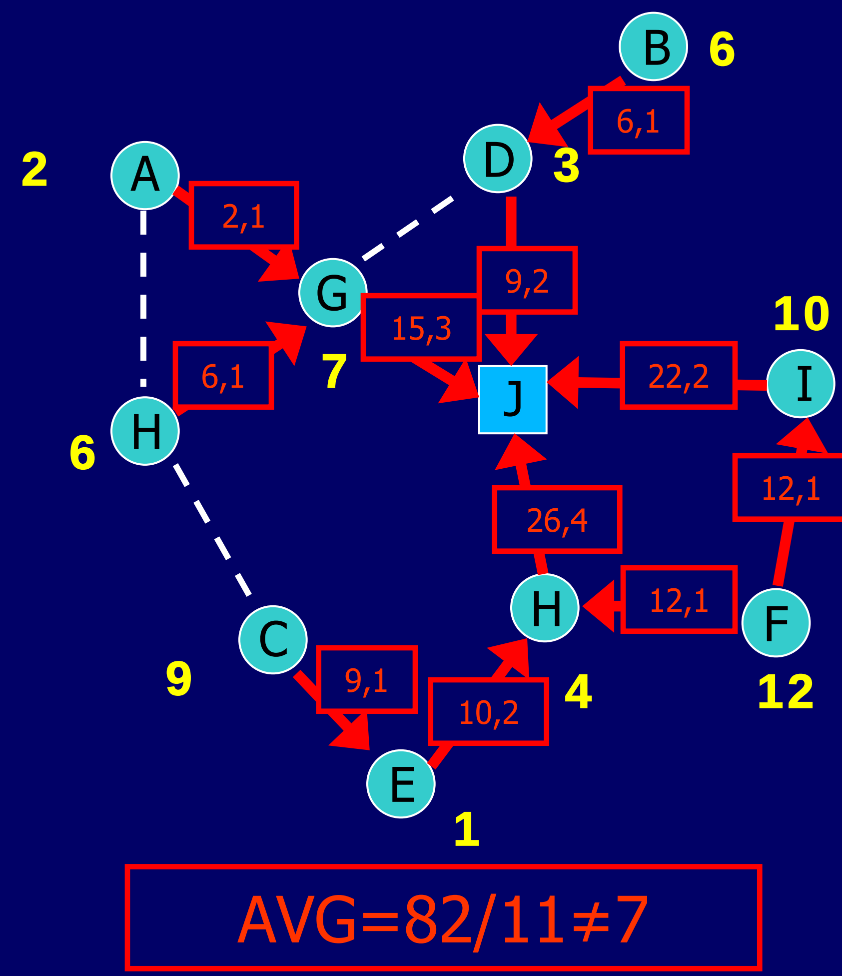
AVG Aggregation (no losses)

- Build spanning tree
- Aggregate in-network
  - Each node sends one summary packet
  - Summary has SUM and COUNT of sub-tree
- Same reliability problem as before



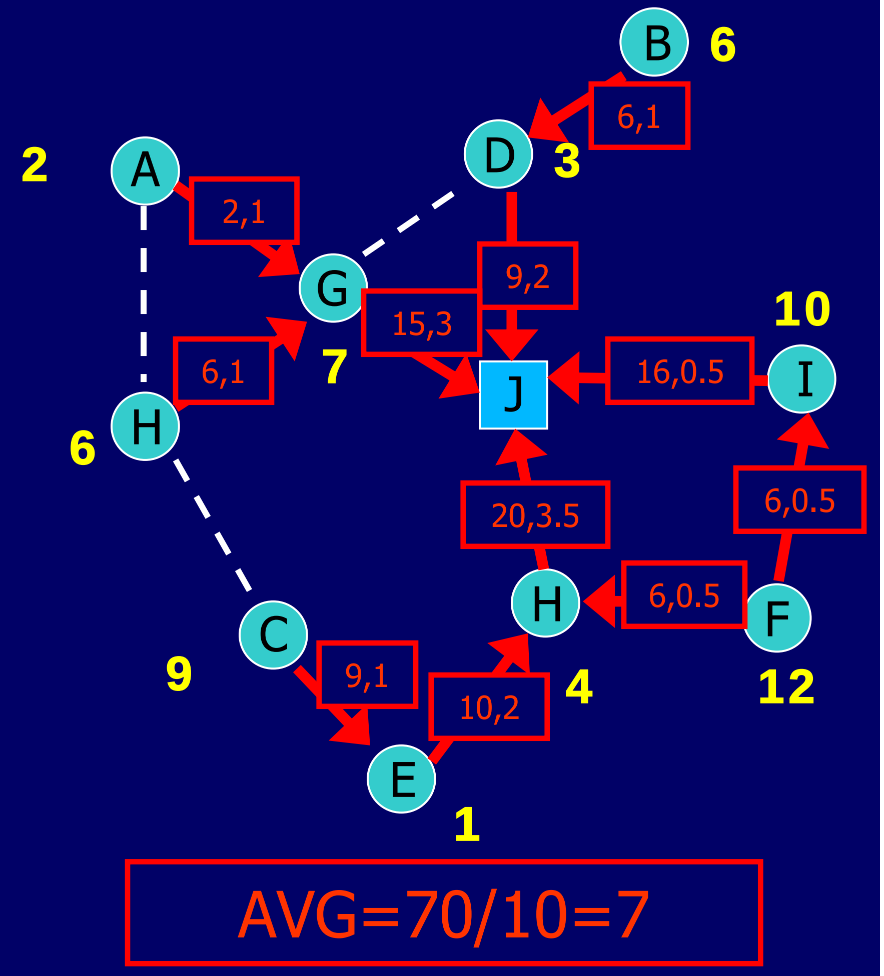
AVG Aggregation (naive)

- What if redundant copies of data are sent?
- AVG is duplicate-sensitive
  - Duplicating data changes aggregate
  - Increases weight of duplicated data



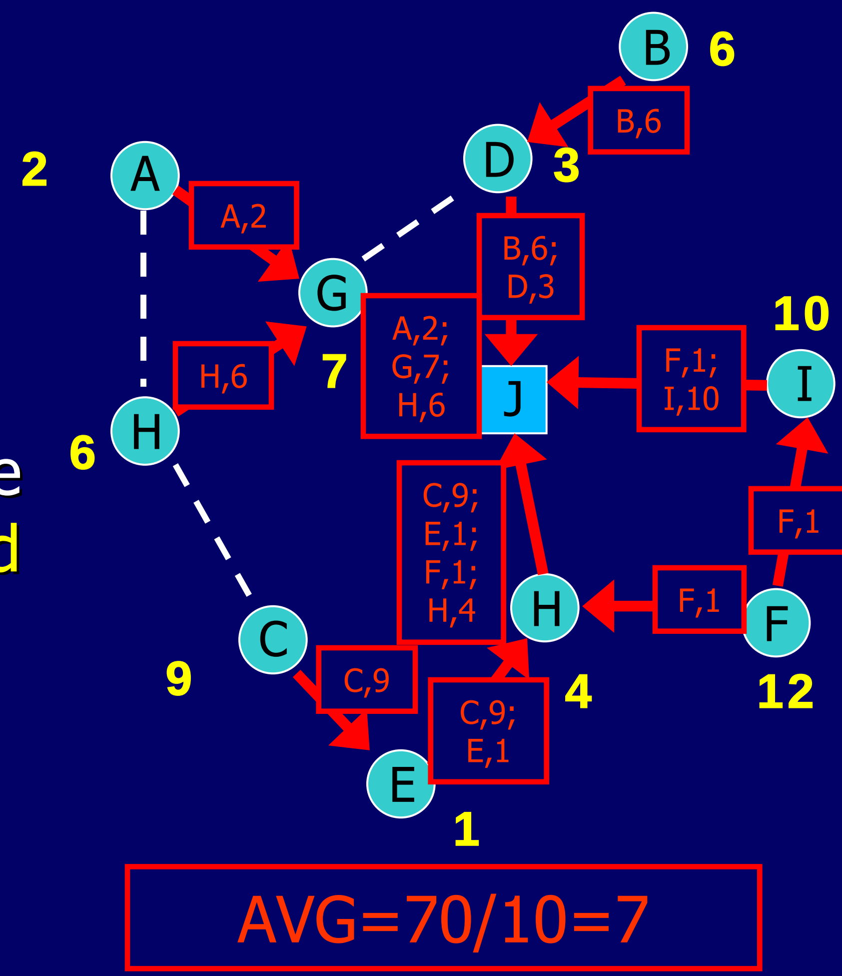
AVG Aggregation (TAG++)

- Can compensate for increased weight [MFHH'02]
  - Send halved SUM and COUNT instead
- Does not change expectation!
- Only reduces variance



AVG Aggregation (LIST)

- Can handle duplicates exactly with a list of <id, value> pairs
- Transmitting this list is expensive!
- Lower bound: linear space is necessary if we demand exact results.



Classification of Aggregates

- TAG classifies aggregates according to
  - Size of partial state
  - Monotonicity
  - Exemplary vs. summary
  - Duplicate-sensitivity
- MIN/MAX (cheap and easy)
  - Small state, monotone, exemplary, duplicate-insensitive
- COUNT/SUM/AVG (considerably harder)
  - Small state and monotone, BUT duplicate-sensitive
  - Cheap if aggregating over tree without losses
  - Expensive with multiple paths and losses

Design Objectives for Robust Aggregation

- Admit in-network aggregation of partial values.
- Let representation of aggregates be both *order-insensitive* and *duplicate-insensitive.*
- Be agnostic to routing protocol
  - Trust routing protocol to be best-effort.
  - Routing and aggregation can be logically decoupled [NG ’03].
  - Some routing algorithms better than others (multipath).
- Exact answers incur extremely high cost.
  - We argue that it is reasonable to use aggregation methods *that are themselves approximate*.

Outline

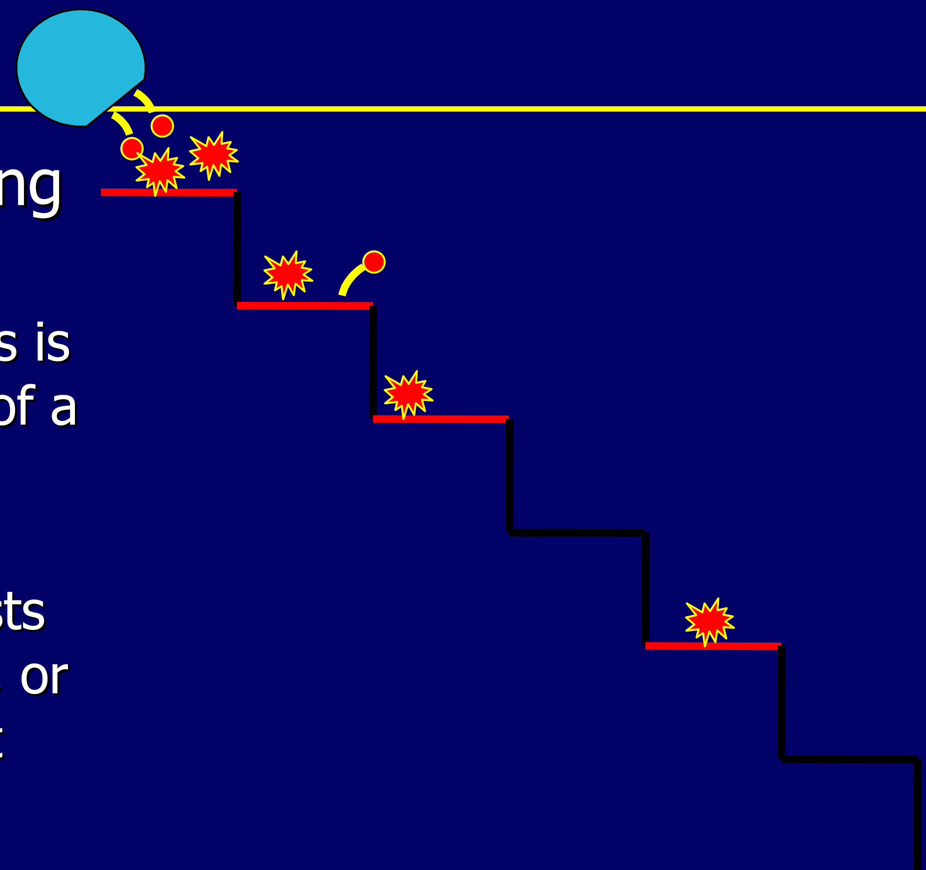
- Introduction
- Sketch Theory and Practice
  - COUNT sketches (old)
  - SUM sketches (new)
  - Practical realizations for sensor nets
- Experiments
- Conclusions

COUNT Sketches

- Problem: Estimate the number of distinct item IDs in a data set with only one pass.
- Constraints:
  - Small space relative to stream size.
  - Small per item processing overhead.
  - Union operator on sketch results.
- Exact COUNT is impossible without linear space.
- First approximate COUNT sketch in [FM'85].
  - O(log N) space, O(1) processing time per item.

Counting Paintballs

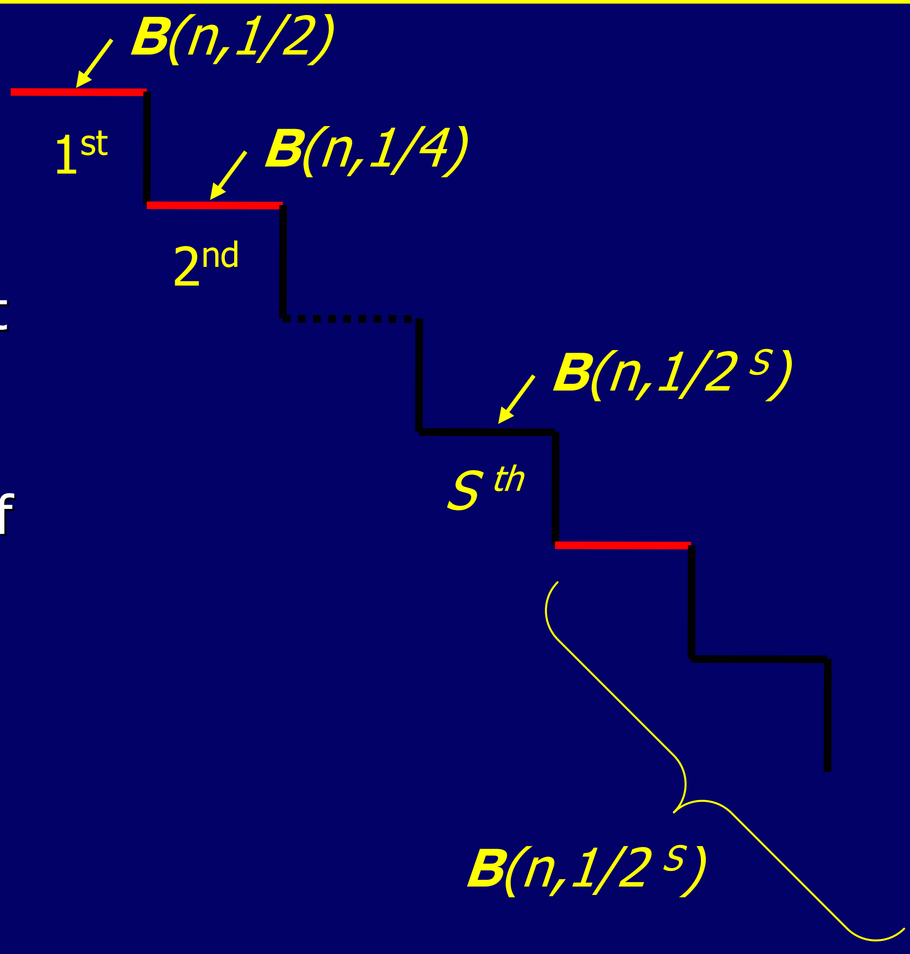
- Imagine the following scenario:
  - A bag of *n* paintballs is emptied at the top of a long stair-case.
  - At each step, each paintball either bursts and marks the step, or bounces to the next step. 50/50 chance either way.



Looking only at the pattern of marked steps, what was *n*?

Counting Paintballs (cont)

- What does the distribution of paintball bursts look like?
  - The number of bursts at each step follows a binomial distribution.
  - The expected number of bursts drops geometrically.
  - Few bursts after $log_2 n$ steps



Counting Paintballs (cont)

- Many different estimator ideas [FM'85,AMS'96,GGR'03,DF'03,...]
- Example: Let *pos* denote the position of the highest unmarked stair,

$$E(pos) \approx \log_2(0.775351\ n)$$

$$\sigma^2(pos) \approx 1.12127$$

- Standard variance reduction methods apply
- Either O(log n) or O(log log n) space

Back to COUNT Sketches

- The COUNT sketches of [FM'85] are equivalent to the paintball process.
  - Start with a bit-vector of all zeros.
  - For each item,
    - Use its ID and a hash function for coin flips.
    - Pick a bit-vector entry.
    - Set that bit to one.
- These sketches are duplicate-insensitive:

{x} →	1	0	0	0	0
{y} →	0	0	1	0	0
{x,y} →	1	0	1	0	0

$$\forall A,B \;\; (\mathit{Sketch}(A) \vee \mathit{Sketch}(B)) = \mathit{Sketch}(A \cup B)$$

Application to Sensornets

- Each sensor computes *k* independent sketches of itself using its unique sensor ID.
  - Coming next: sensor computes sketches of its *value*.
- Use a robust routing algorithm to route sketches up to the sink.
- Aggregate the *k* sketches via in-network XOR.
  - Union via XOR is duplicate-insensitive.
- The sink then estimates the count.

- Similar to gossip and epidemic protocols.

SUM Sketches

- Problem: Estimate the sum of values of distinct <*key, value*> pairs in a data stream with repetitions. (*value* ≥ 0, integral).

- Obvious start: Emulate *value* insertions into a COUNT sketch and use the same estimators.

  – For *<k,v>*, imagine inserting

  <k, v, 1>, <k, v, 2>, …, <k, v, v>

SUM Sketches (cont)

- But what if the value is 1,000,000?
- Main Idea (details on next slide):
  - Recall that all of the low-order bits will be set to 1 w.h.p. inserting such a value.
  - Just set these bits to one immediately.
  - Then set the high-order bits carefully.

Simulating a set of insertions

- Set all the low-order bits in the "safe" region.
  - First $S = \log v - 2 \log \log v$ bits are set to 1 w.h.p.
- Statistically estimate number of trials going beyond "safe" region
  - Probability of a trial doing so is simply 2^{-S}
  - Number of trials ~ $\boldsymbol{B}(v, 2^{-S})$. [Mean = $O(\log^2 v)$]
- For trials and bits outside "safe" region, set those bits manually.
  - Running time is O(1) for each outlying trial.

Expected running time:
$O(\log v)$ + time to draw from $\boldsymbol{B}(v, 2^{-S})$ + $O(\log^2 v)$

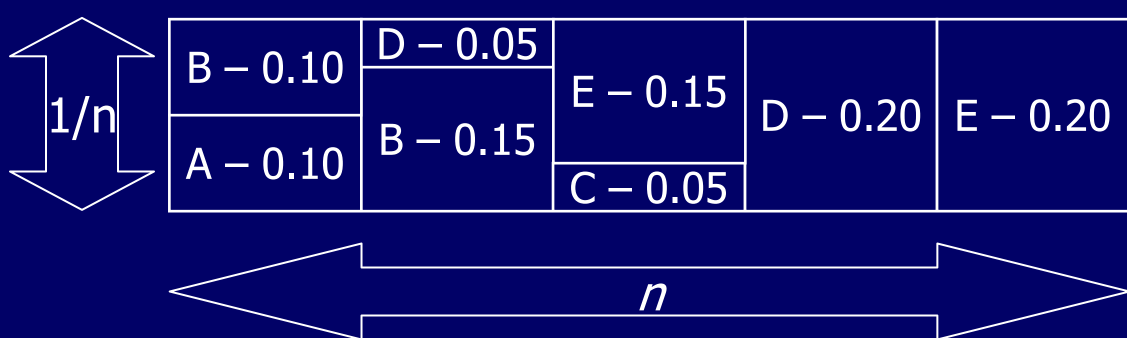
Sampling for Sensor Networks

- We need to generate samples from ***B*** (n, p).
  - With a slow CPU, very little RAM, no floating point hardware
- General problem: sampling from a discrete pdf.
- Assume can draw uniformly at random from [0,1].
- With an event space of size N:
  - O(log N) lookups are immediate.
    - Represent the cdf in an array of size N.
    - Draw from [0, 1] and do binary search.
  - Cleverer methods for O(log log N), O(log* N) time

Amazingly, this can be done in constant time!

Walker's Alias Method

- Theorem [Walker '77]: For any discrete pdf D over a sample space of size n, a table of size *O(n)* can be constructed in *O(n)* time that enables random variables to be drawn from D using at most two table lookups.



Binomial Sampling for Sensors

- Recall we want to sample from ***B****(v,2^{-S})* for various values of *v* and *S.*
  - First, reduce to sampling from ***G****(1 – 2^{-S}).*
  - Truncate distribution to make range finite (recursion to handle large values).
  - Construct tables of size *2^{S}* for each *S* of interest.
  - Can sample ***B****(v,2^{-S})* in *O(v · 2^{-S})* expected time.

The Bottom Line

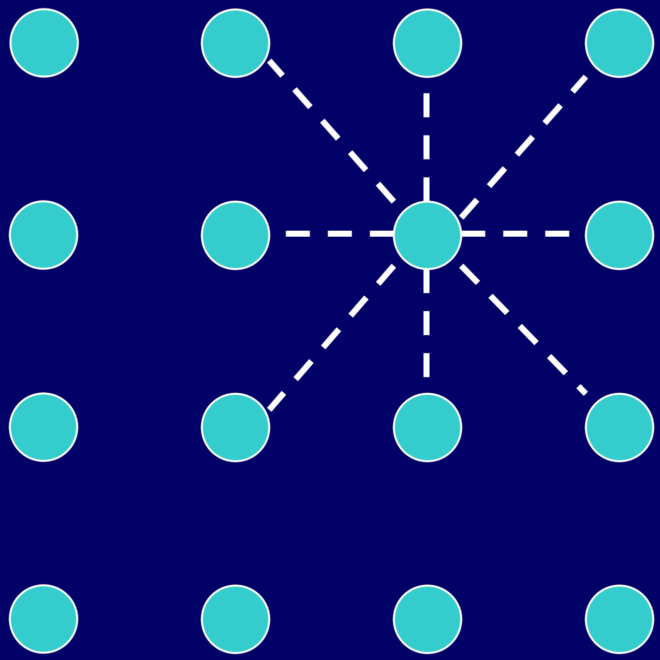
- SUM inserts in
  - $O(log^2(v))$ time with $O(v / log^2(v))$ space
  - $O(log(v))$ time with $O(v / log(v))$ space
  - $O(v)$ time with naïve method
- Using $O(log^2(v))$ method, 16 bit values ($S \leq 8$) and 64 bit probabilities
  - Resulting lookup tables are ~ 4.5KB
  - Recursive nature of $\mathbf{G}(1 - 2^{-S})$ lets us tune size further
- Can achieve $O(log\ v)$ time at the cost of bigger tables

Outline

- Introduction
- Sketch Theory and Practice
- Experiments
- Conclusions

Experimental Results

- Used TAG simulator
- Grid topology with sink in the middle
  - Grid size
    [default: 30 by 30]
  - Transmission radius
    [default: 8 neighbors on the grid]
  - Node, packet, or link loss
    [default: 5% link loss rate]
  - Number of bit vectors
    [default: 20 bit-vectors of 16 bits (compressed)].

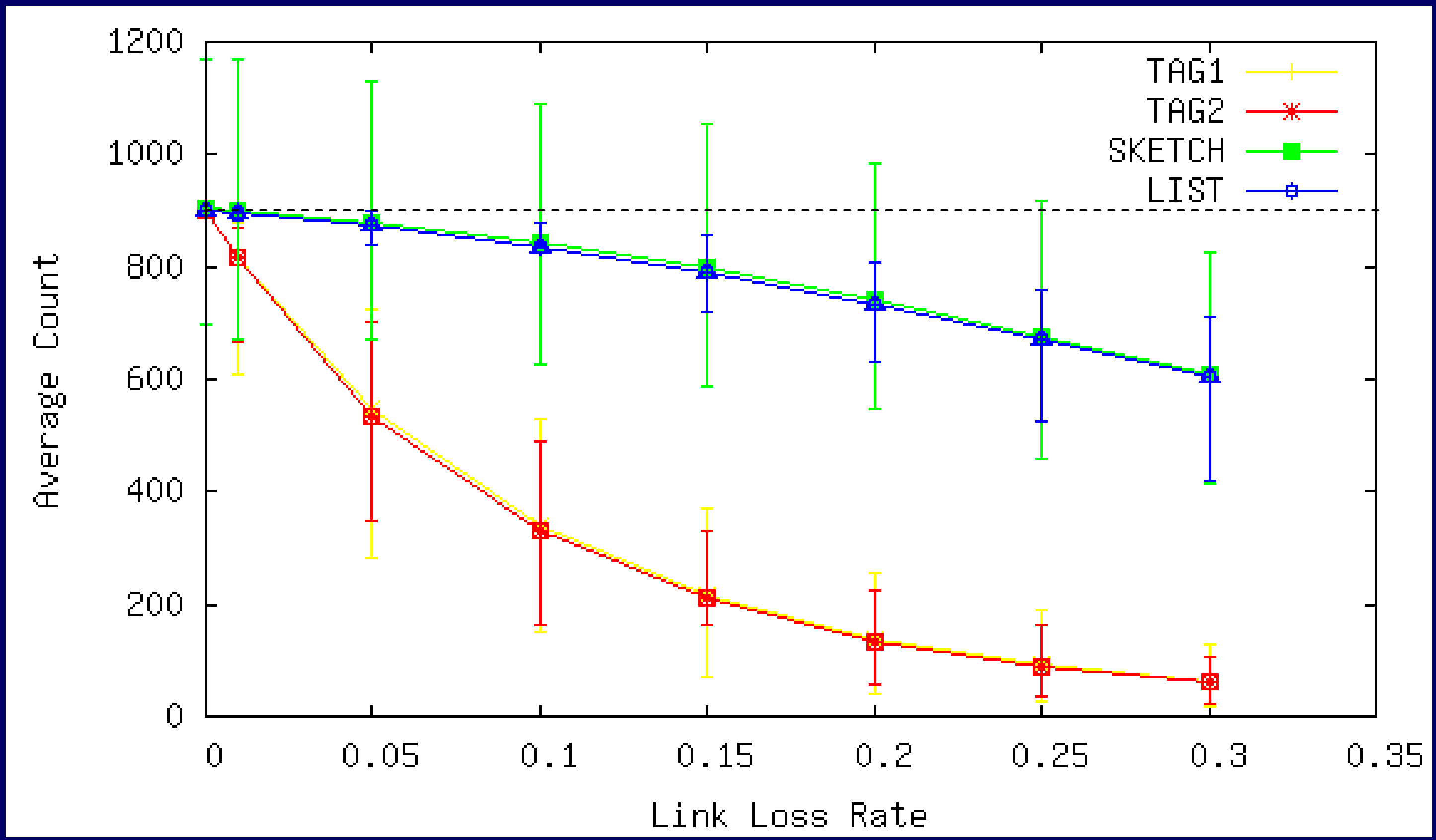


Experimental Results

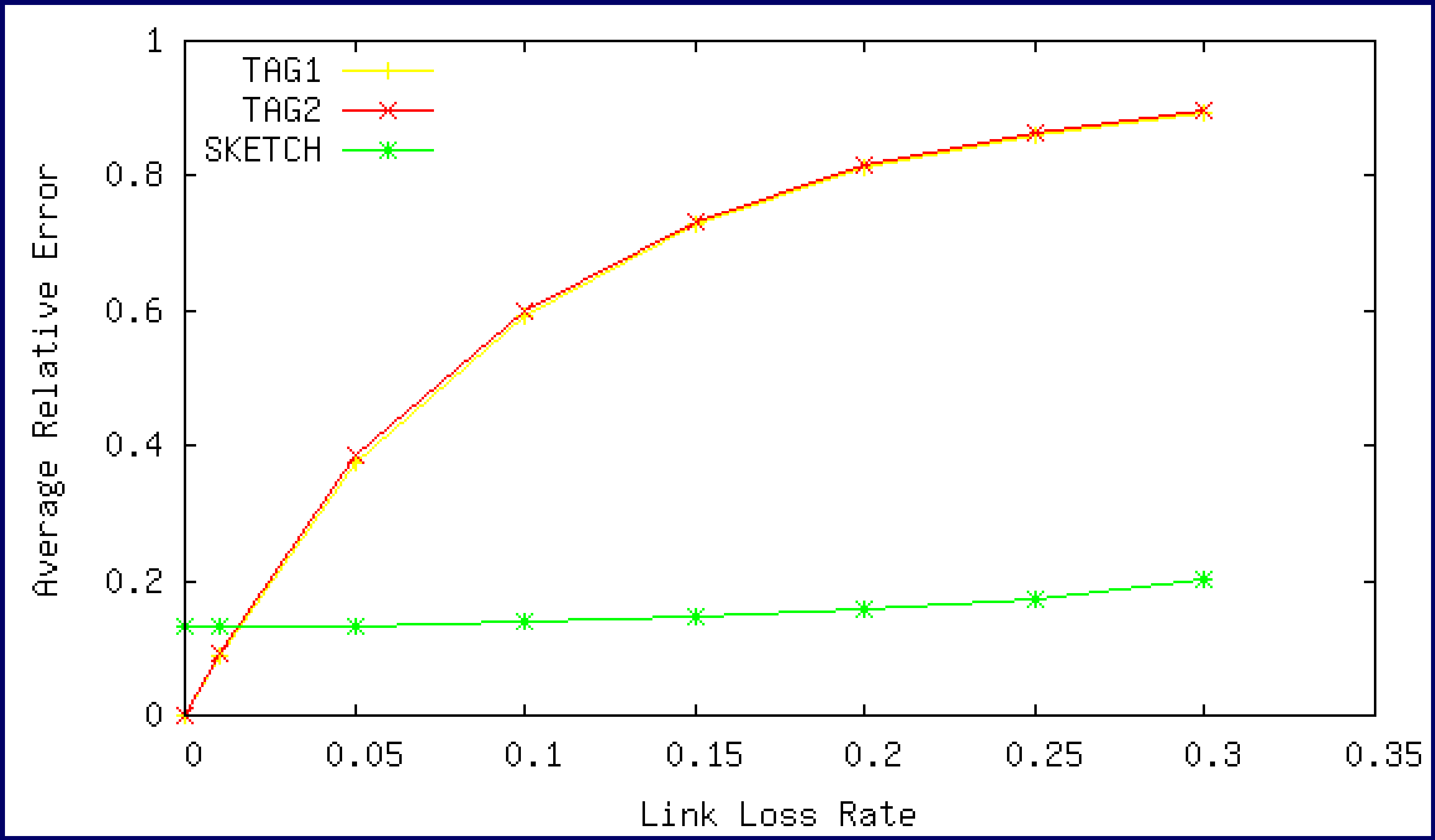
- We consider four main methods.
  - TAG1: transmit aggregates up a single spanning tree
  - TAG2: Send a $1/k$ fraction of the aggregated values to each of k parents.
  - SKETCH: broadcast an aggregated sketch to all neighbors at level $i-1$
  - LIST: explicitly enumerate all <key, value> pairs and broadcast to all neighbors at level $i - 1$.

- LIST vs. SKETCH measures the penalty associated with approximate values.

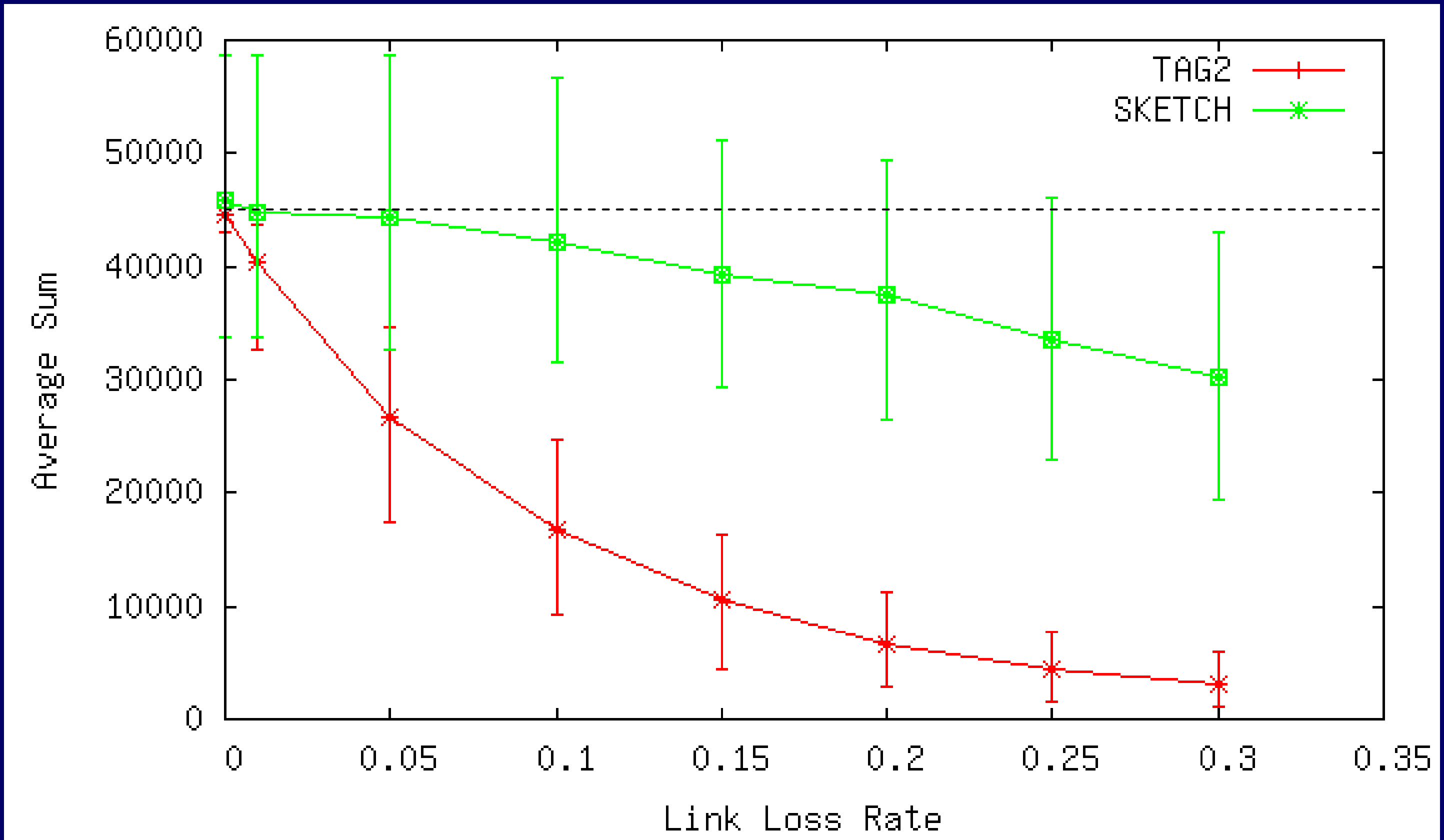
COUNT vs Link Loss (grid)



COUNT vs Link Loss (grid)



SUM vs Link Loss (grid)



Message Cost Comparison

Strategy	Total Data Bytes	Messages Sent	Messages Received
TAG1	1800	900	900
TAG2	1800	900	2468
SKETCH	10843	900	2468
LIST	170424	900	2468

Outline

- Introduction
- Sketch Theory and Practice
- Experiments
- Conclusions

Our Work in Context

- In parallel with our efforts,
  - Nath and Gibbons (Intel/CMU)
    - What are the essential properties of duplicate insensitivity?
    - What other aggregates can be sketched?
  - Bawa et al (Stanford)
    - What routing methods are necessary to guarantee the validity and semantics of aggregates?

Conclusions

- Duplicate-insensitive sketches fundamentally change how aggregation works
  - Routing becomes logically decoupled
  - Arbitrarily complex aggregation scenarios are allowable – cyclic topologies, multiple sinks, etc.
- Extended list of non-trivial aggregates
  - We added SUM, MEAN, VARIANCE, STDEV, …
- Resulting system performs better
  - Moderate cost (tunable) for large reliability boost

Ongoing Work

- What else can we sketch?
  - Clear need to extend expressiveness of sketches
  - Also: what are the limits of duplicate-insensitive ones?
- Distributed streaming model
  - Monitor and sketch streams of data
  - Collect sketches and estimate global properties
- Traffic monitoring
  - Identifying large flows, flows with large changes
  - Both already done with counting Bloom filters [KSGC'03,CM'04]
    - We can make those duplicate-insensitive!
- Aggregation via random sampling

Future Directions (cont)

Message Cost Comparison

Strategy	Total Data Bytes	Messages Sent	Messages Received
TAG1	1800	900	900
TAG2	1800	900	2468
SKETCH	10843	900	2468
LIST	170424	900	2468

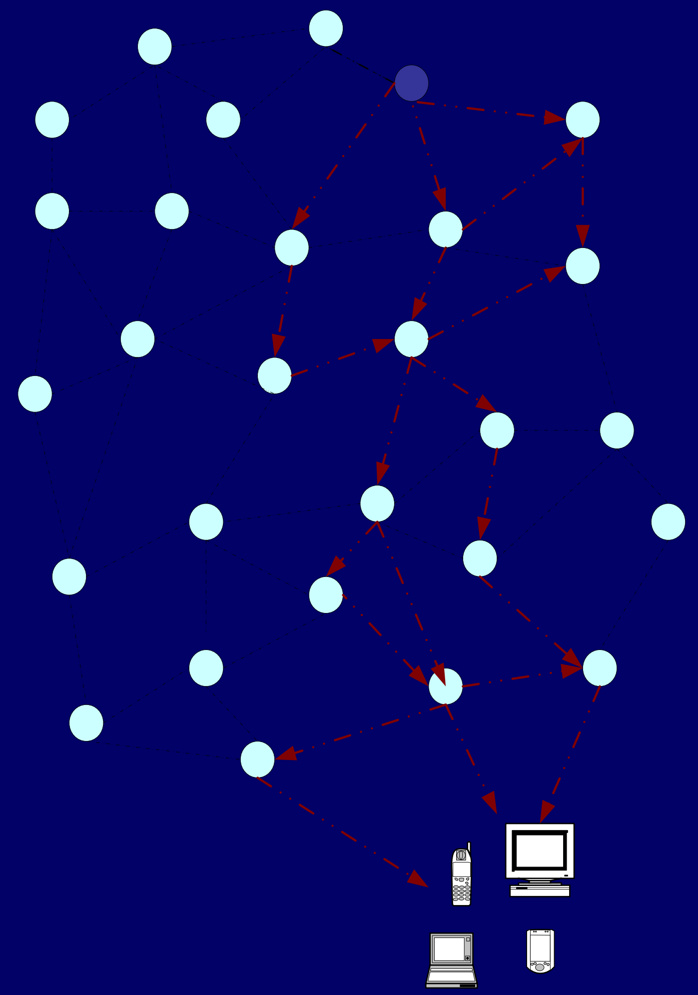
Thank you!

More questions?

Multipath Routing

Braided Paths:

Two paths from the source to the sink that differ in at least two nodes



Design Objectives (cont)

- Final aggregate is exact if at least one representative from each leaf survives to reach the sink.
- This won't happen in practice in sensornets without extremely high cost.
- It is reasonable to hope for approximate results.

- We argue that it is reasonable to use aggregation methods *that are themselves approximate*.

Goal of This Work

- So far, we've seen ideas of
  - In-network aggregation (low traffic per link)
  - Multi-path routing (reliability of individual items)
- These usually don't combine well
  - Only works for duplicate-insensitive aggregates such as MIN/MAX, AND/OR
- What about all the other aggregates?
  - We want them cheap, reliable, and correct

Contributions of This Work

- Propose duplicate-insensitive sketches to approximately aggregate data
  - Difficulty was noted [MFHH'02]
  - Approximation is necessary
- With duplicate-insensitive sketches, any best-effort routing method can be employed
- Design new duplicate-insensitive sketches
  - SUM => MEAN, VARIANCE, STDEV, ...

Routing Methodologies

- Considerable work on reliable delivery via multipath routing
  - Directed diffusion [IGE ’00]
  - “Braided” diffusion [GGSE ’01]
  - GRAdient Broadcast [YZLZ ’02]
    - Broadcast intermediate results along gradient back to source
    - Can dynamically control width of broadcast
    - Trade off fault tolerance and transmission costs
- Our approach similar to GRAB:
  - Broadcast. Grab if upstream, ignore if downstream

Common goal: try to get at least one copy to sink

SUM Sketches (cont)

- Remaining questions:
  - What should S be when inserting <k, v>?
    - When using analysis of [FM'85]
      - $S \approx log_2(v) - 2\ log_2\ log(v)$
      - Expected time = $O(log^2(v))$ + *sample time*
    - Can go farther keeping high probability…
      - $S \approx log_2(v) - log_2\ log(v)$
      - Expected time = $O(log(v))$ + *sample time*
  - How do we sample the binomial distribution?
    - Space requirements may affect choice of S

SUM Sketches (cont)

- Reduction to COUNT sketches:
  – Pick a prefix length *S*
    - The first *S* bits should be set with high probability.
  – Set the first *S* bits to one.
  – Sample from $\boldsymbol{B}(v, 2^{-S})$ to figure out how many items would pick bits after the first *S* bits.
  – Simulate the insertion of those items.

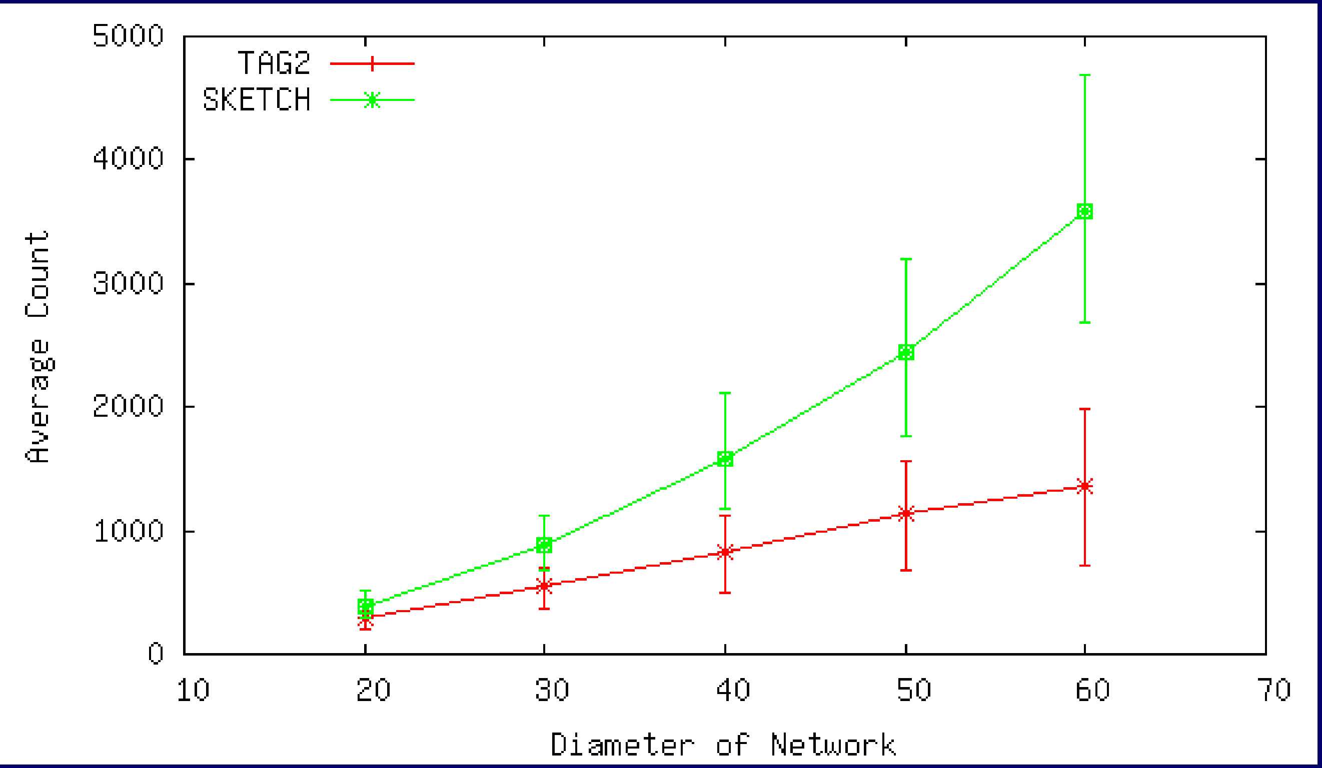
- Expected time =

$$O(S) + sample\ time + O(v \cdot 2^{-S})$$

Sampling Constraints

- Sensor motes have very limited resources
  - Slow CPU
  - Very little RAM
  - No floating point hardware
- Sampling from ***B****(n, p)* isn't easy normally
  - Obviously *O(log n)* time and *O(n)* space
  - *O(np)* expected time (and good FP hardware) with standard reduction to geometric distribution
- How hard is this sampling problem anyway?

COUNT vs Diameter (grid)



COUNT vs Link Loss (random)