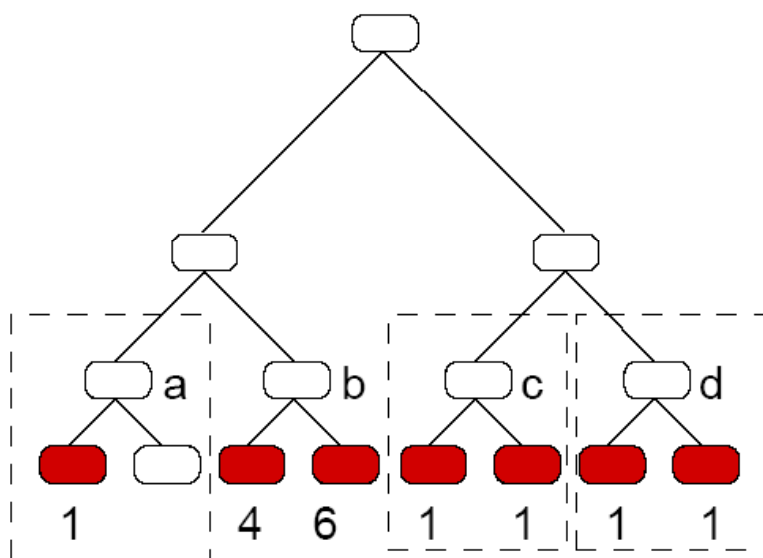

Data Collection and Aggregation

Q-digest

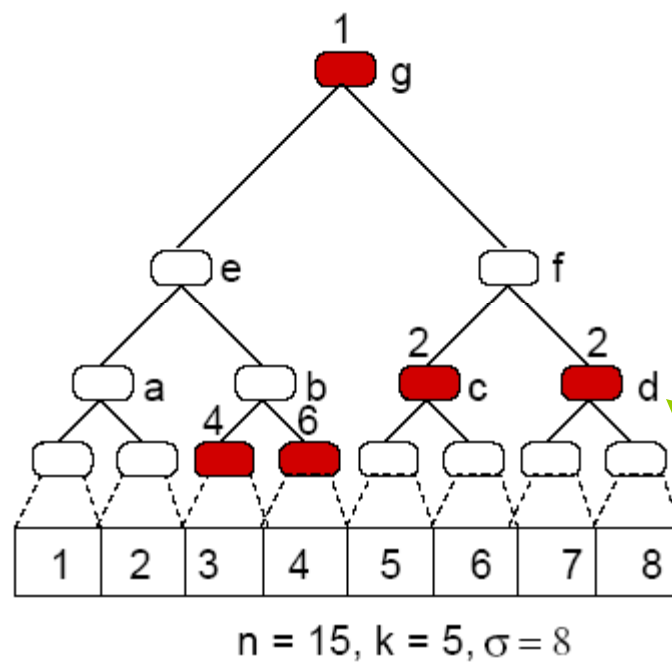
- Input data: frequency of data value $\{f_1, f_2, \dots, f_\sigma\}$.
- Compress the data:
 - detailed information concerning frequent data are preserved;
 - less frequently occurring values are lumped into larger buckets resulting in information loss.
- Buckets: the nodes in a **binary partition** of the range $[1, \sigma]$. Each bucket v has range $[v.min, v.max]$.
- Only store non-zero buckets.

Example

Input data bucketed




Q-digest



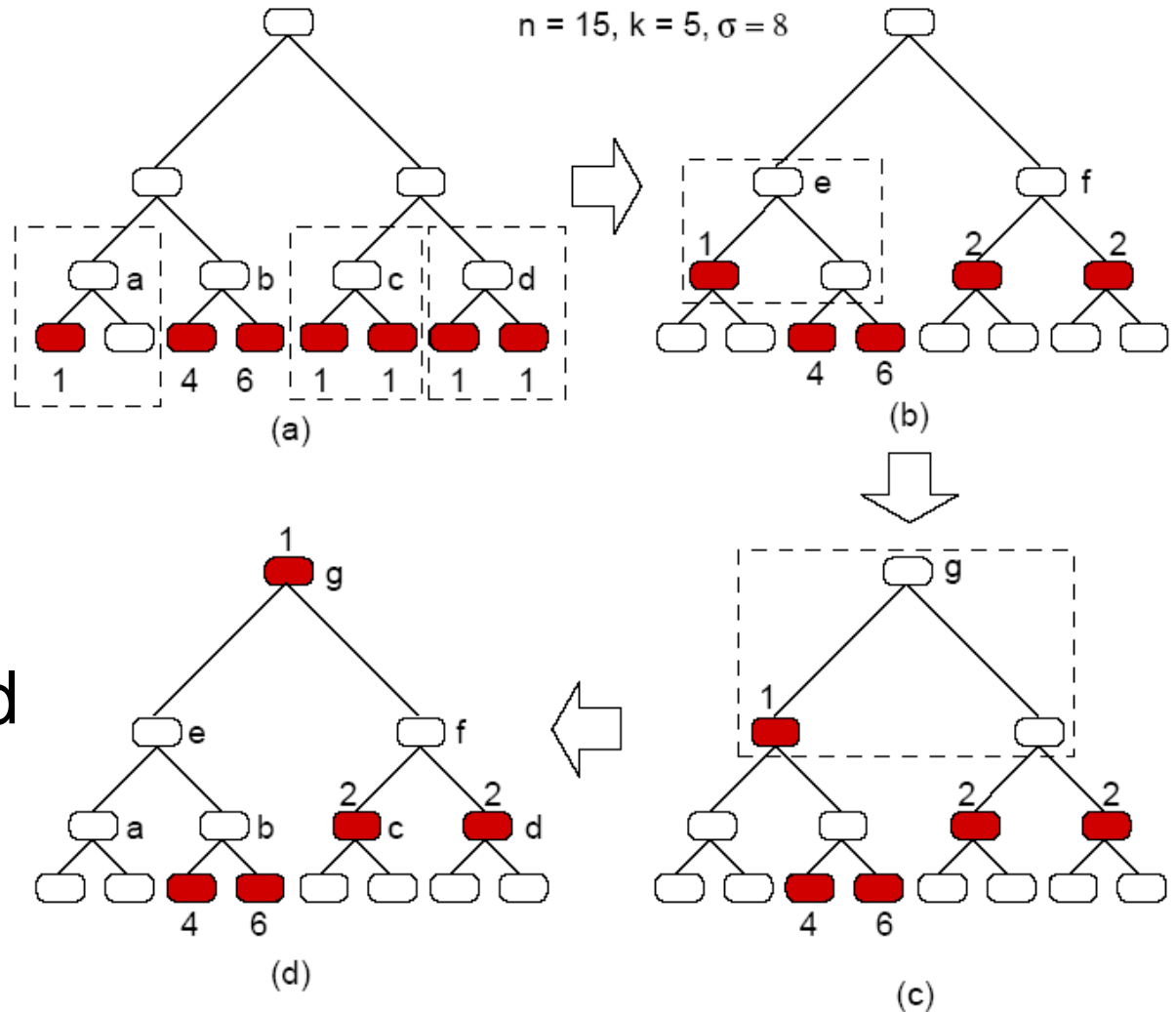
Information loss

Q-digest properties

- Store values in buckets.
 1. $\text{Count}(v) \leq n/k$. (except leaf)
 - Control information loss.
 2. $\text{Count}(v) + \text{Count}(p) + \text{Count}(s) > n/k$.
(except root)
 - Ensure sufficient compression.
 - K : compression parameter.
- 

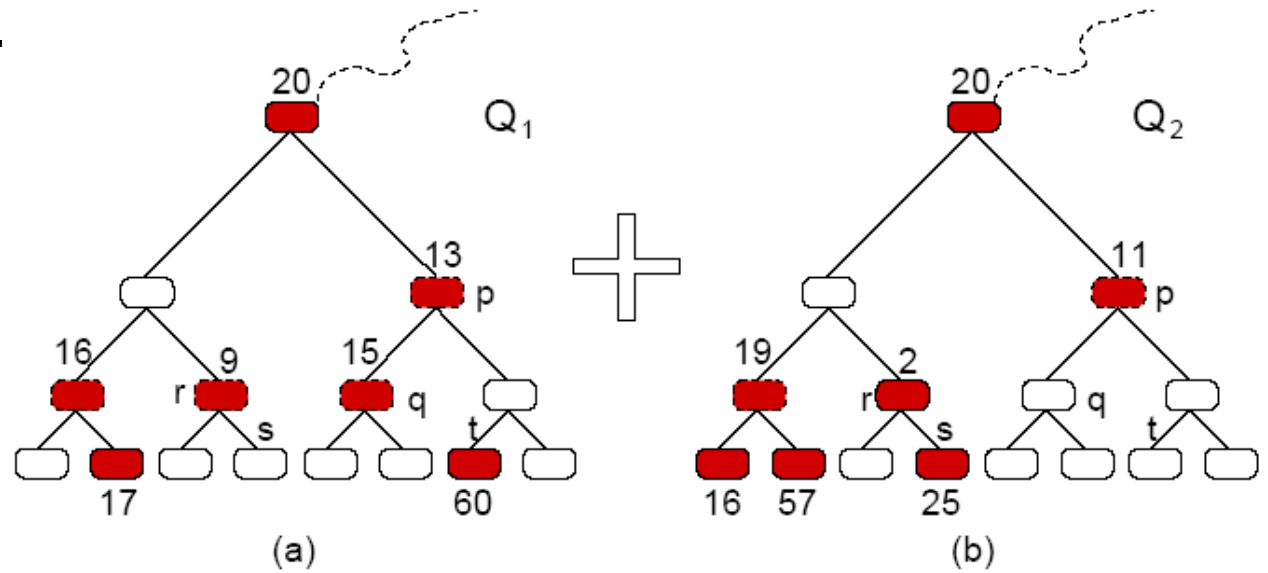
Construct a q-digest

- Each sensor constructs a q-digest based on its value.
- Check the digest property bottom up: two “small” children’s count are added up and moved to the parent.

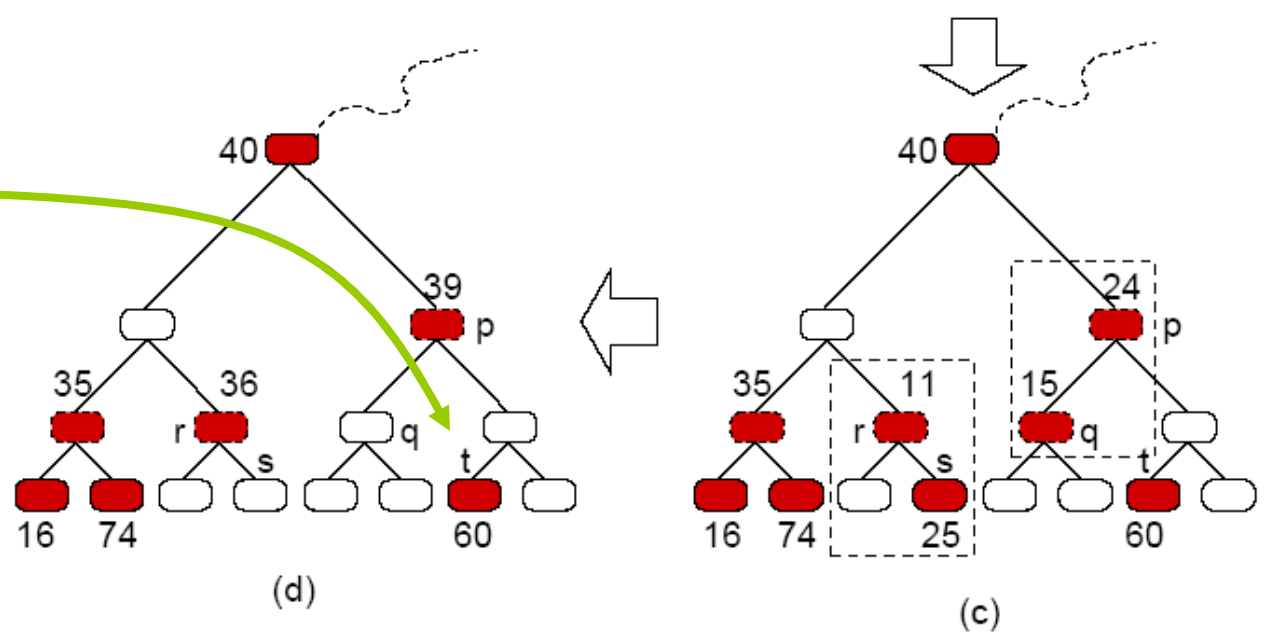


Merging two q-digests

- Merge q-digests from two children
- Add up the values in buckets
- Re-evaluate the digest property bottom up.



Information loss: t undercounts since some of its value appears on ancestors.



Space complexity

Claim: A q-digest with compression parameter k has at most 3k buckets.

- By property 2, for all buckets v in Q ,
 - $\sum_{v \in Q} [\text{Count}(v) + \text{Count}(p) + \text{Count}(s)] > |Q| n/k.$
 - $\sum_{v \in Q} [\text{Count}(v) + \text{Count}(p) + \text{Count}(s)] \leq 3$
 $[\sum_{v \in Q} \text{Count}(v)] = 3n.$
 - $|Q| < 3k.$

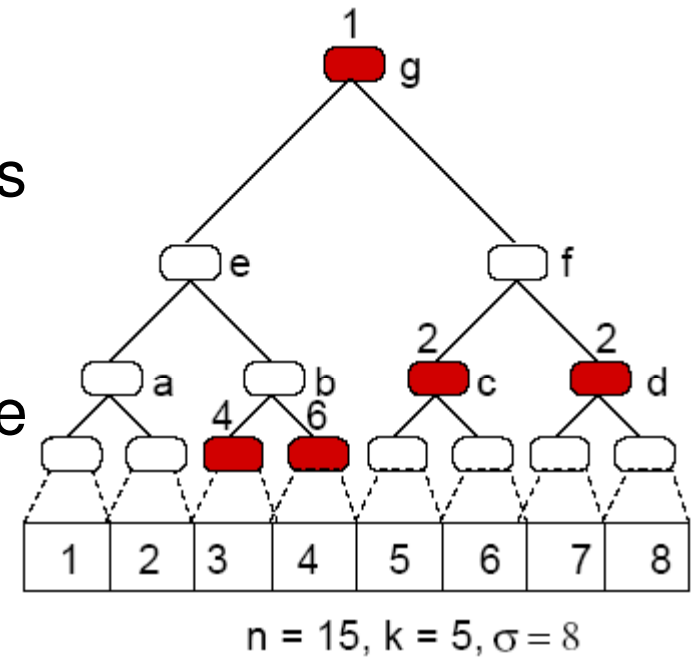
Error bound

Claim: Any value that should be counted in v can be present in one of the ancestors.

1. $\text{Count}(v)$ has max error $\log\sigma \cdot n/k$.
 - $\text{Error}(v) \leq \sum_{\text{ancestor } p} \text{Count}(p) \leq \sum_{\text{ancestor } p} n/k \leq \log\sigma \cdot n/k$.
2. MERGE maintains the same relative error.
 - $\text{Error}(v) \leq \sum_i \text{Error}(v_i) \leq \sum_i \log\sigma \cdot n_i/k \leq \log\sigma \cdot n/k$.

Median and quantile query

- Given $q \in (0, 1)$, find the value whose rank is qn .
- Relative error $\epsilon = |r - qn|/n$, where r is the true rank.
- Post-order traversal on Q , sum the counts of all nodes visited before a node v , which is $\leq \#$ of values less than $v.\max$. Report it when it is first time larger than qn .
- Error bound: $\epsilon = \log \sigma / k$.

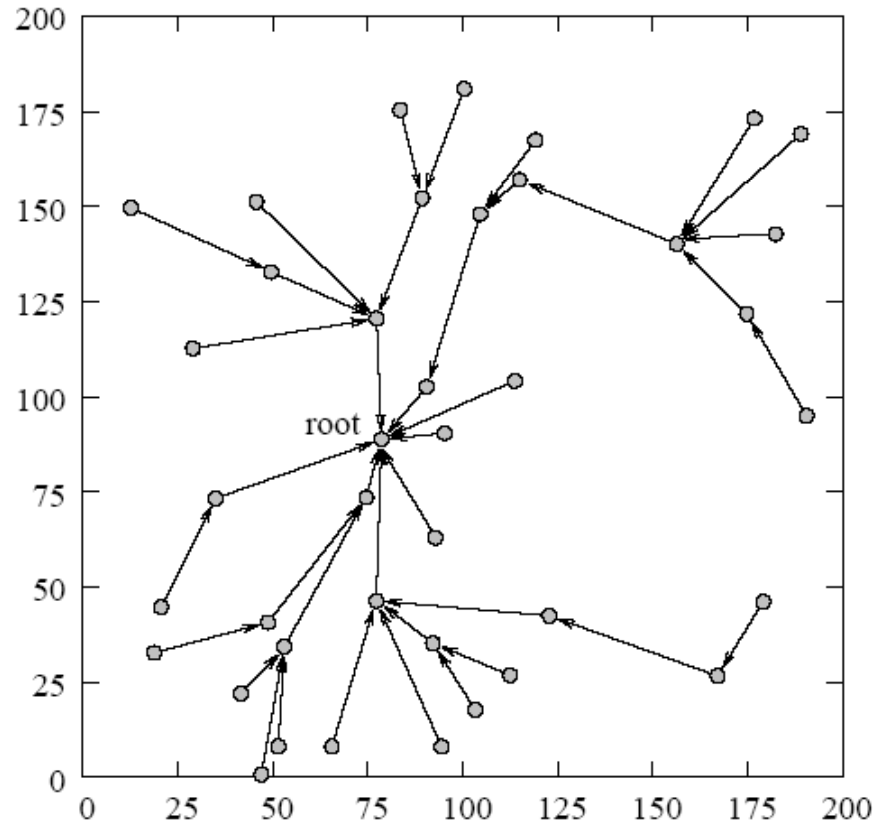


Other queries

- **Inverse quantile**: given a value, determine its rank.
 - Traverse the tree in post-order, report the sum of counts v for which $x > v$, which is within $[\text{rank}(x), \text{rank}(x) + \epsilon n]$
- **Range query**: find # values in range $[l, h]$.
 - Perform two inverse quantile queries and take the difference. Error bound is $2\epsilon n$.
- **Frequent items**: given $s \in (0, 1)$, find all values reported by more than sn sensors.
 - Count the leaf buckets whose counts are more than $(s - \epsilon)n$.
 - Small false positive: values with count between $(s - \epsilon)n$ and sn may also be reported as frequent.

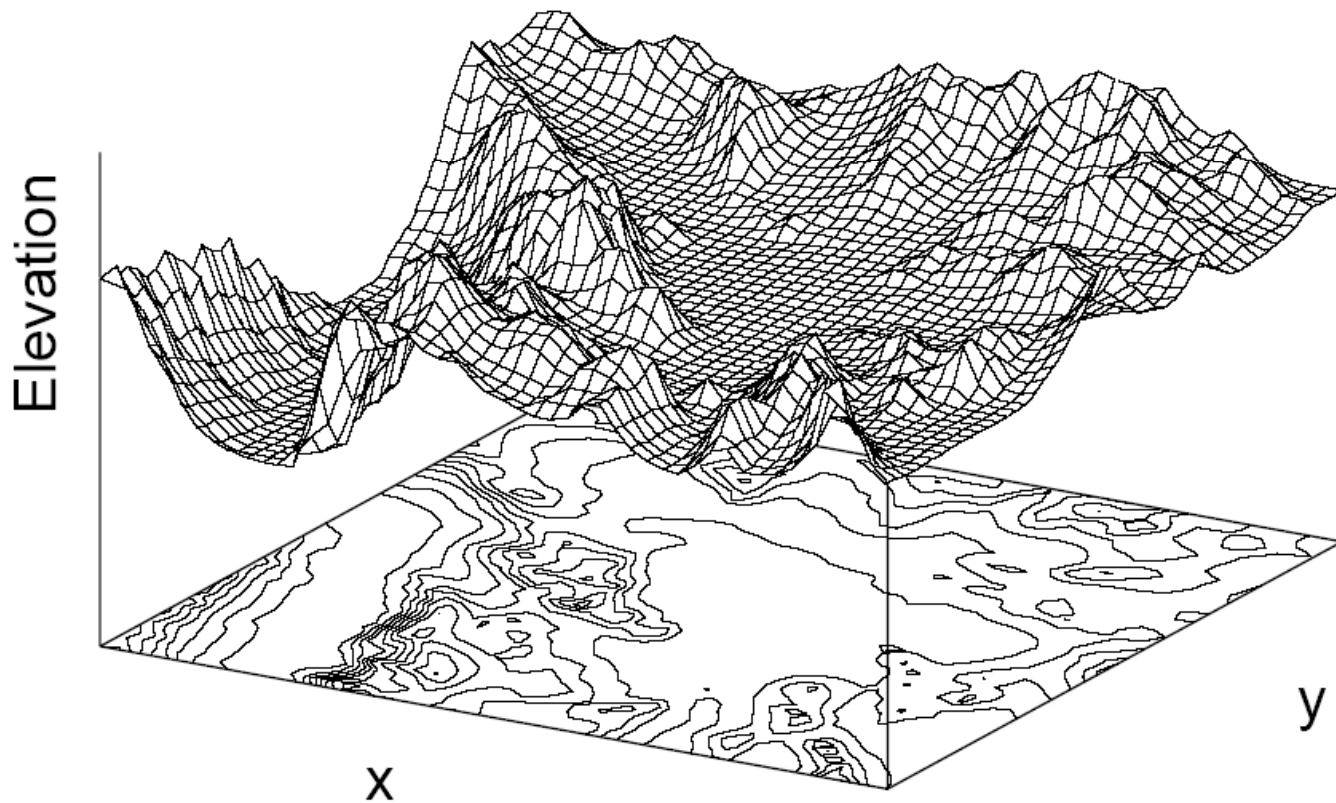
Simulation setup

- A typical aggregation tree (BFS tree) on 40 nodes in a 200 by 200 area. In the simulation they use 4000~8000 nodes.



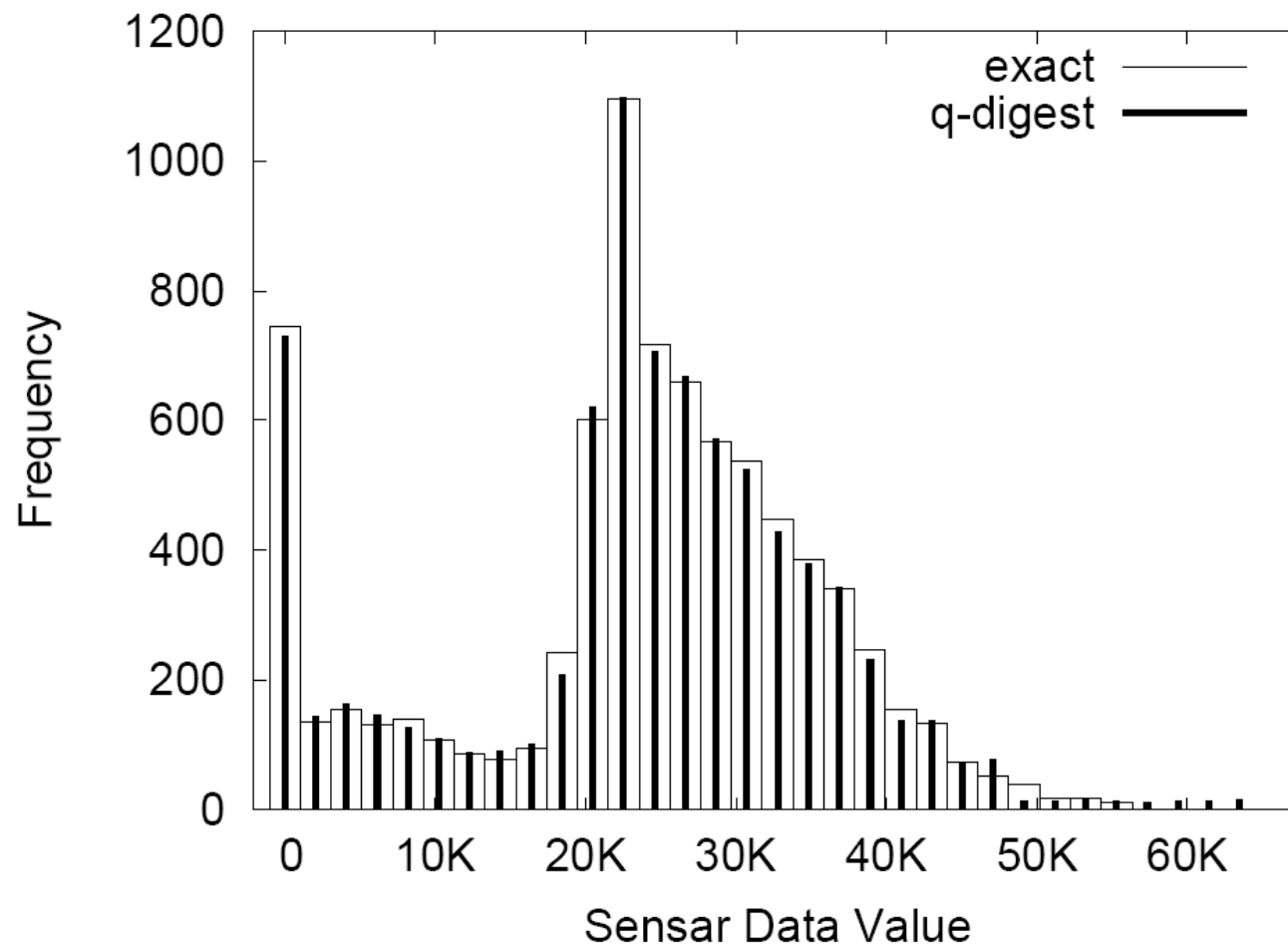
Simulation setup

- Random data;
- Correlated data: 3D elevation value from Death Valley.

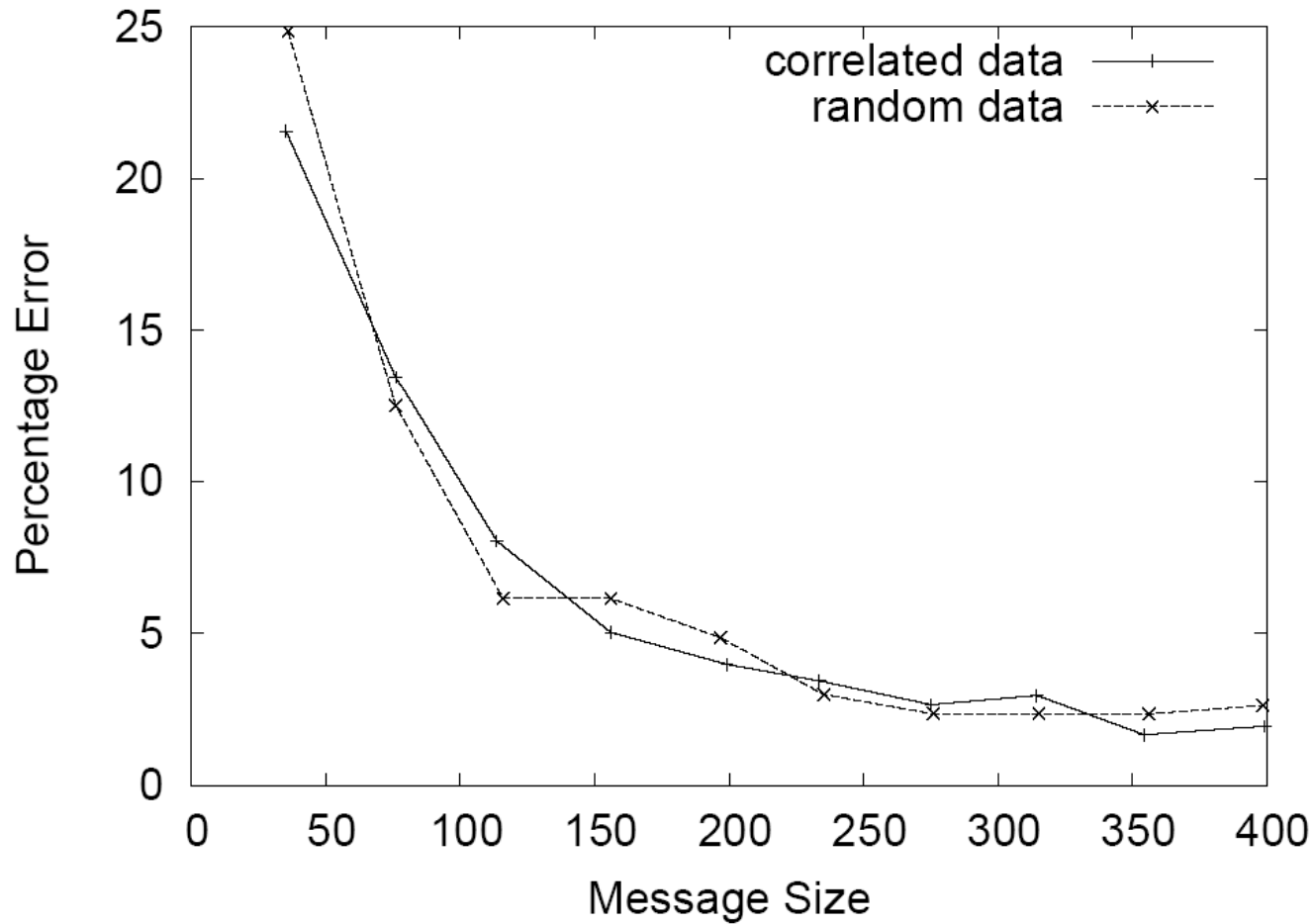


Histogram v.s. q-digest

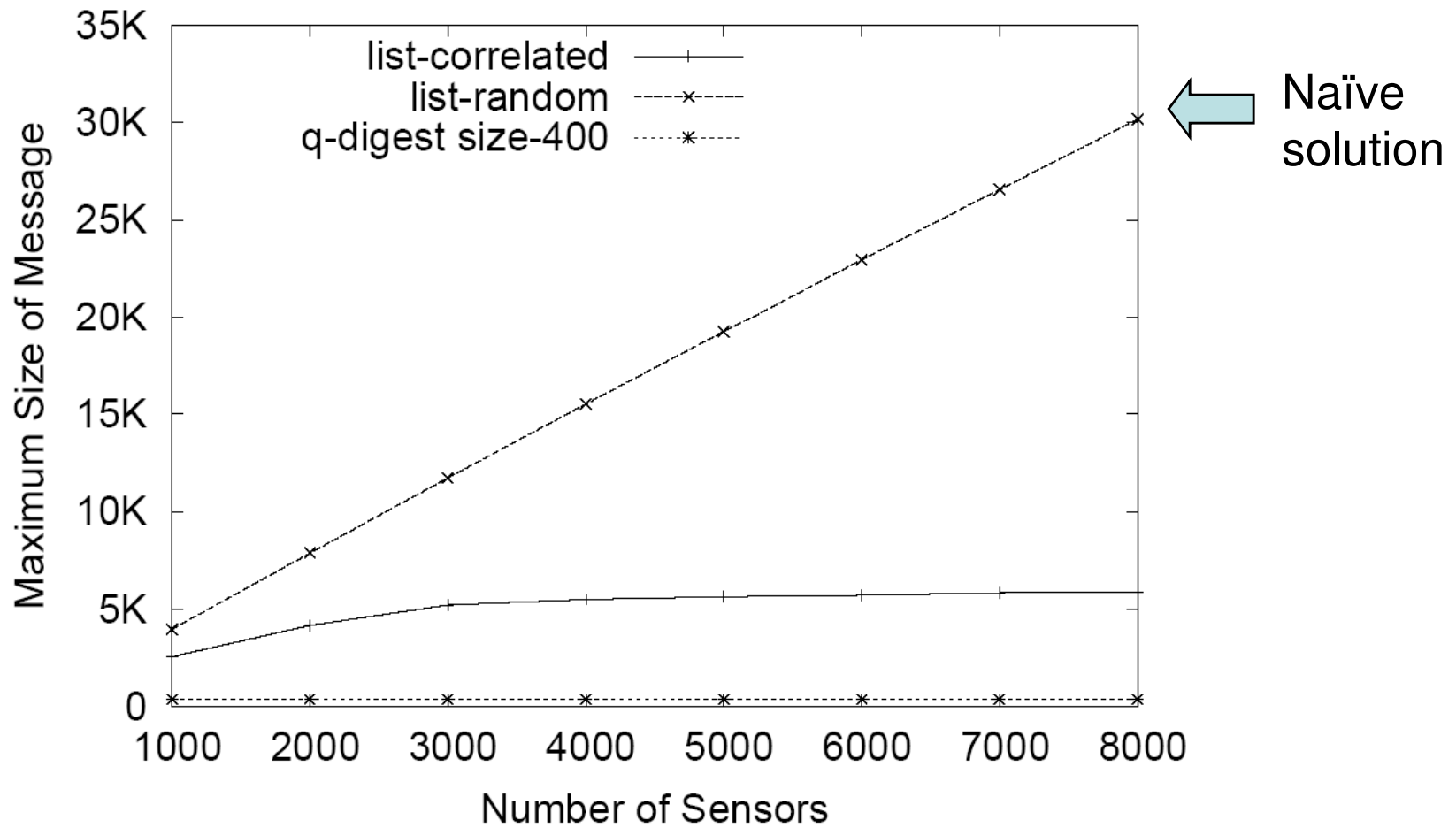
- Comparison of histogram and q-digest.



Tradeoff between error and msg size

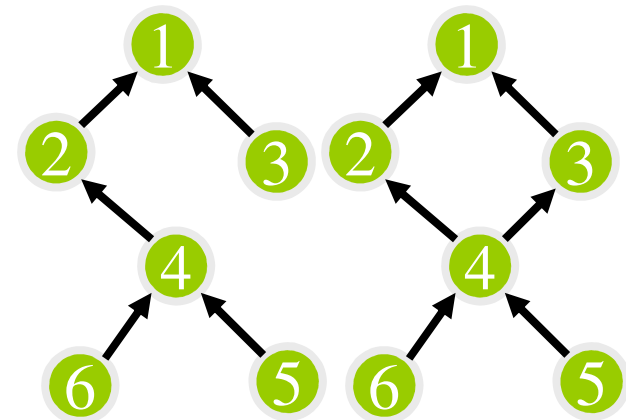


Saving on message size

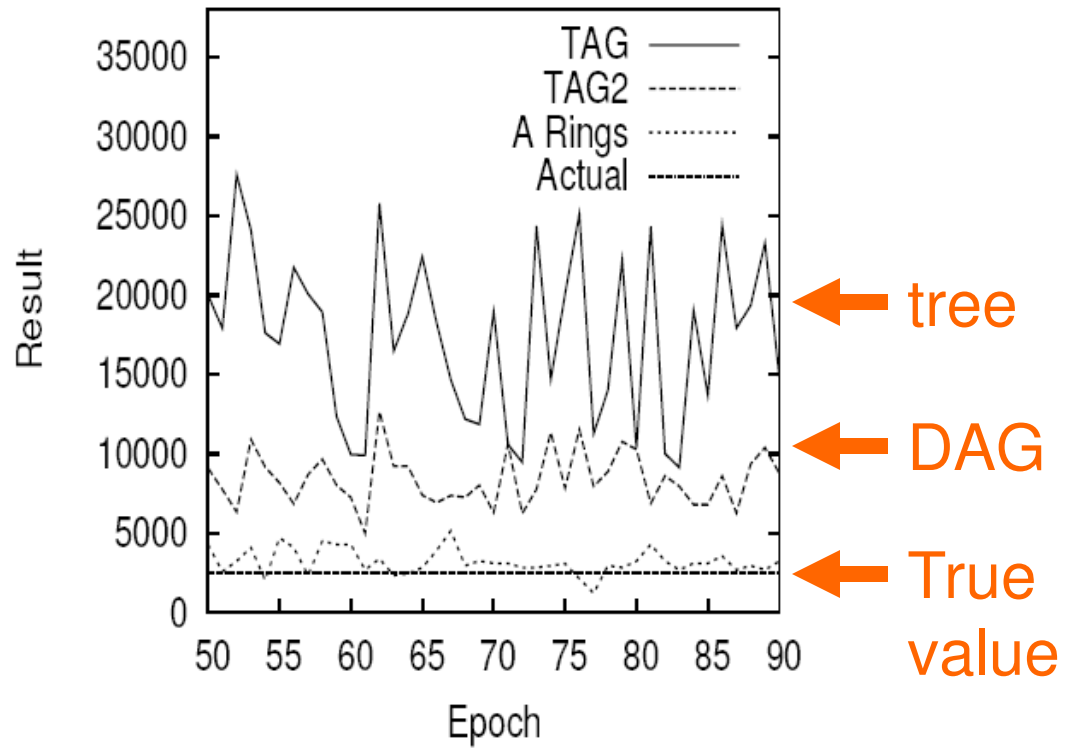
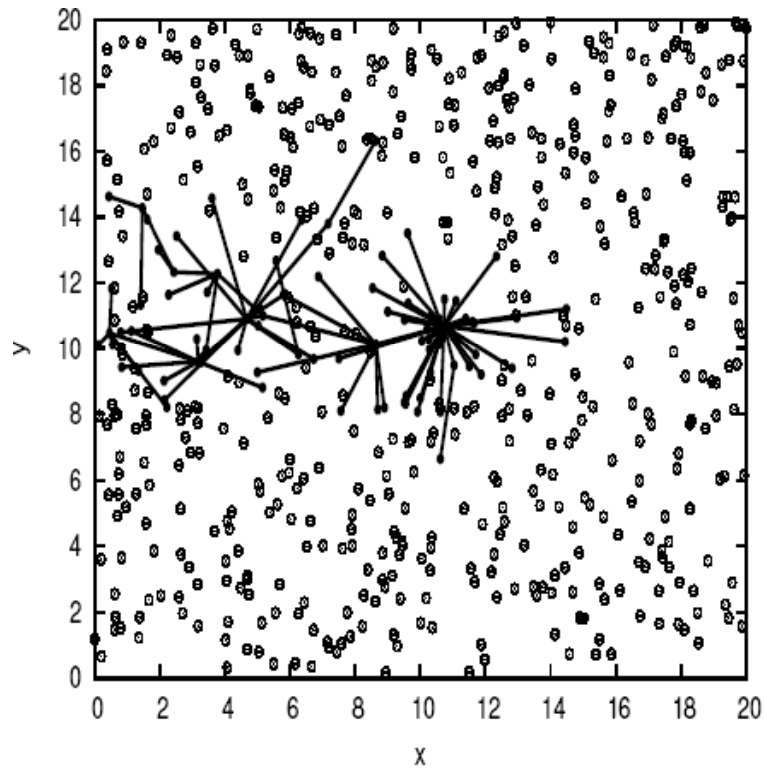


2nd problem: Aggregation tree in practice

- Tree is a fragile structure.
 - If a link fails, the data from the entire subtree is lost.
- Fix #1: use a DAG instead of a tree.
 - Send $1/k$ data to each of the k upstream nodes (parents).
 - A link failure lost $1/k$ data



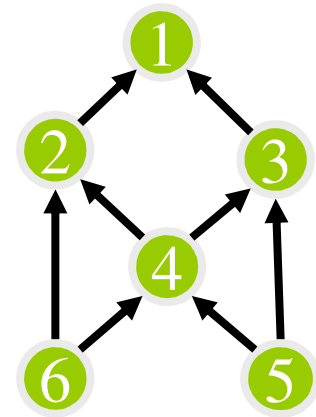
Aggregation tree in practice



(a) Nodes counted in TAG (b) Computing Avg with TAG

Fundamental problem

- Aggregation and routing are coupled
- Improve routing robustness by multi-path routing?
 - Same data might be delivered multiple times.
 - Message over-counting.
- Decouple routing & aggregation
 - Work on the robustness of each separately



Order and duplicate insensitive (ODI) synopsis

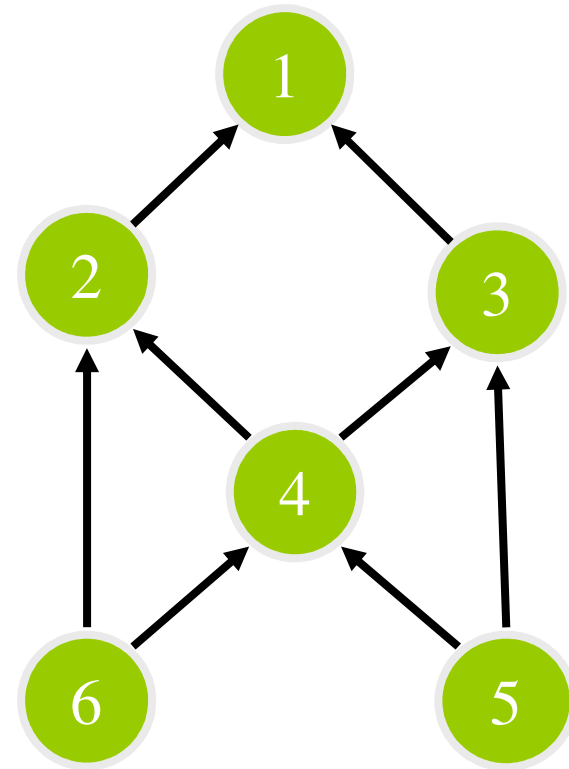
- Aggregated value is **insensitive** to the **sequence** or **duplication** of input data.
- Small-sizes digests such that any particular sensor reading is accounted for only once.
 - Example: MIN, MAX.
 - Challenge: how about COUNT, SUM?

Aggregation framework

- Solution for robustness aggregation:
 - Robust routing (e.g., multi-hop) + ODI synopsis.
- Leaf nodes: [Synopsis generation](#): $SG(\cdot)$.
- Internal nodes: [Synopsis fusion](#): $SF(\cdot)$ takes two synopsis and generate a new synopsis of the union of input data.
- Root node: [Synopsis evaluation](#): $SE(\cdot)$ translates the synopsis to the final answer.

An easy example: ODI synopsis for MAX/MIN

- Synopsis generation: $SG(\cdot)$.
 - Output the value itself.
- Synopsis fusion: $SF(\cdot)$
 - Take the MAX/MIN of the two input values.
- Synopsis evaluation: $SE(\cdot)$.
 - Output the synopsis.



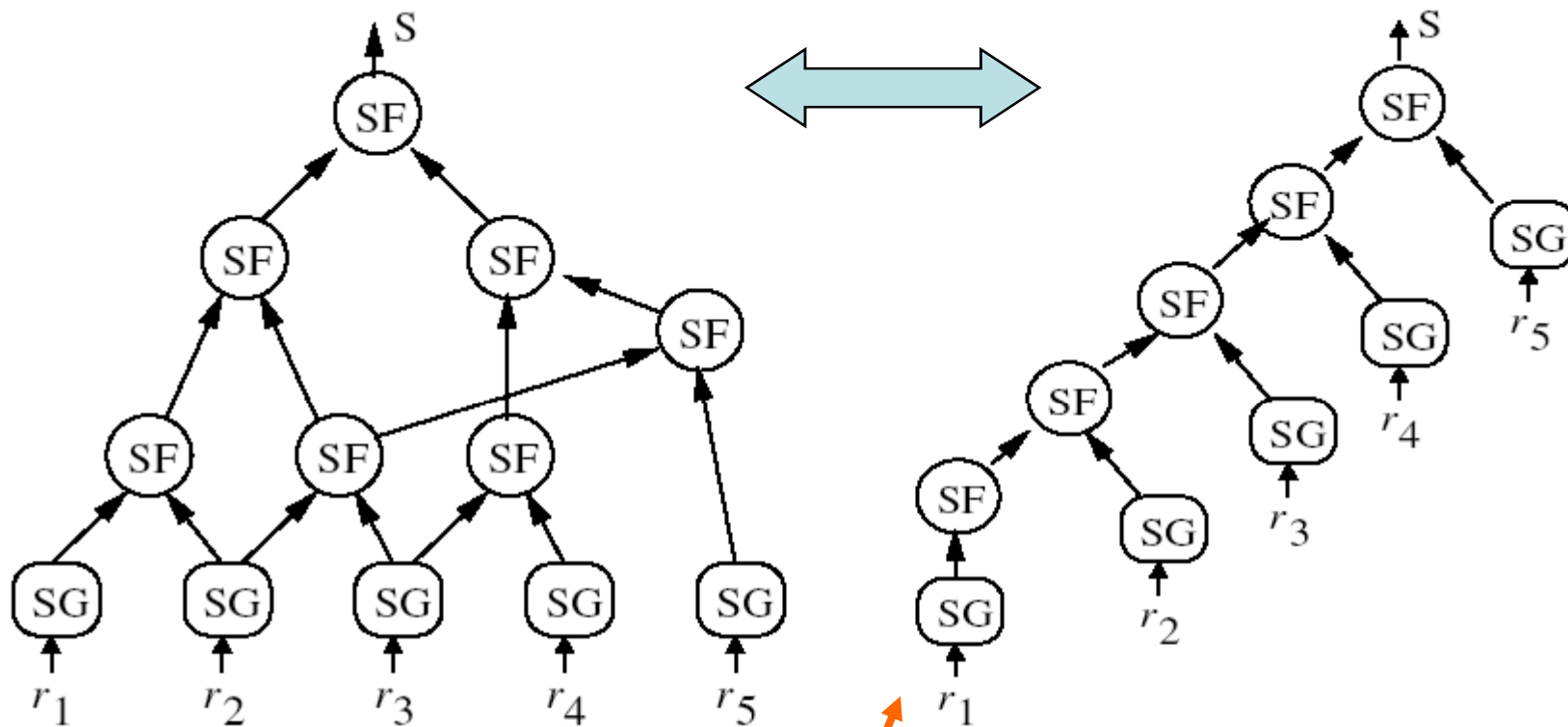
Three questions

- What do we mean by ODI, rigorously?
- Robust routing + ODI
- How to design ODI synopsis?
 - COUNT
 - SUM
 - Sampling
 - Most popular k items
 - Set membership – Bloom filter

Definition of ODI correctness

- A synopsis diffusion algorithm is ODI-correct if SF() and SG() are order and duplicate insensitive functions.
- Or, if for **any** aggregation DAG, the resulting synopsis is **identical** to the synopsis produced by the canonical left-deep tree.
- The final result is independent of the **underlying routing topology**.
 - Any evaluation order.
 - Any data duplication.

Definition of ODI correctness



(a) Aggregation DAG

(b) Canonical left-deep tree

Connection to streaming model: data item comes 1 by 1.

Test for ODI correctness

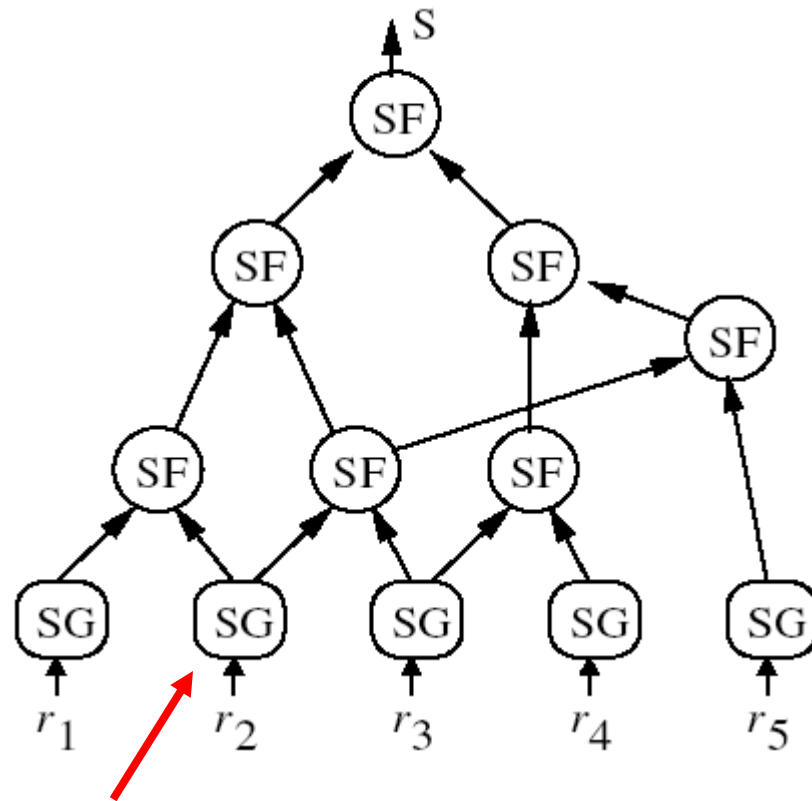
1. $SG()$ **preserves duplicates**: if two readings are duplicates (e.g., two nodes with same temperature readings), then the same synopsis is generated.
2. $SF()$ is **commutative**.
3. $SF()$ is **associative**.
4. $SF()$ is **same-synopsis idempotent**, $SF(s, s)=s$.

Theorem: The above properties are **sufficient and necessary** properties for ODI-correctness.

Proof idea: transfer an aggregation DAG to a left-deep tree with the same output by using these properties.

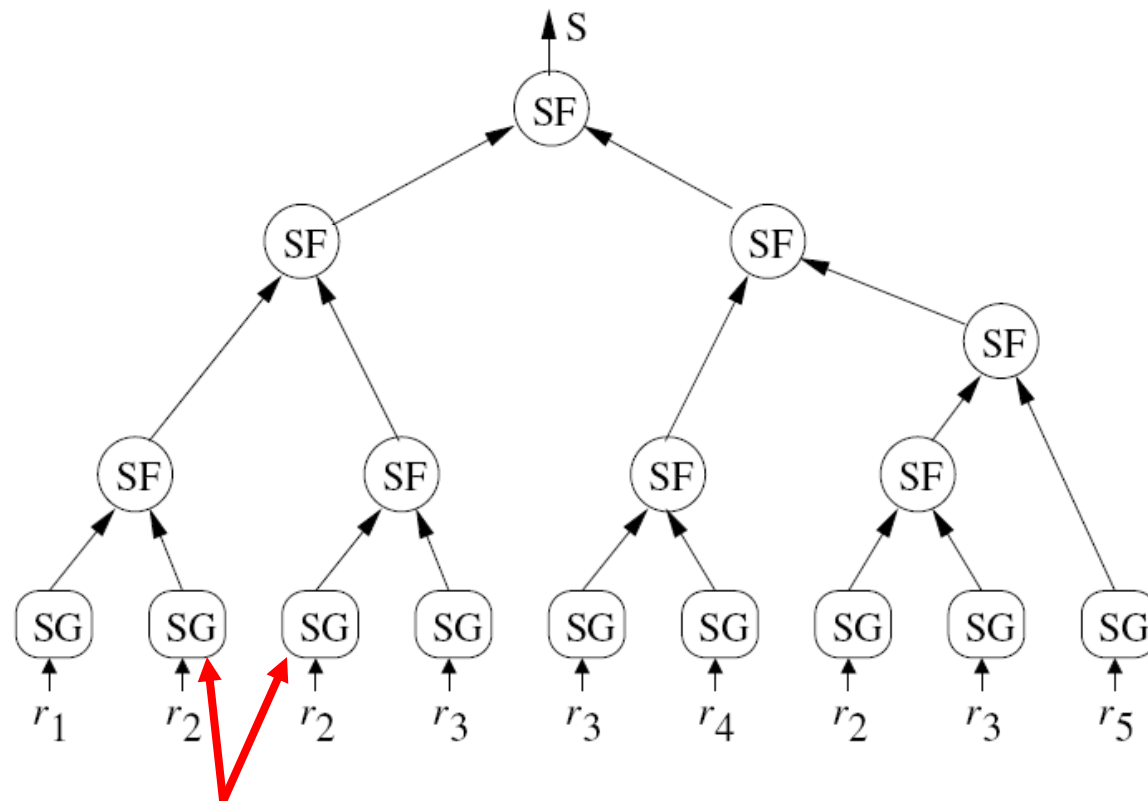
Proof of ODI correctness

1. Start from the DAG. Duplicate a node with out-degree k to k nodes, each with out degree 1. ← duplicates preserving.



Proof of ODI correctness

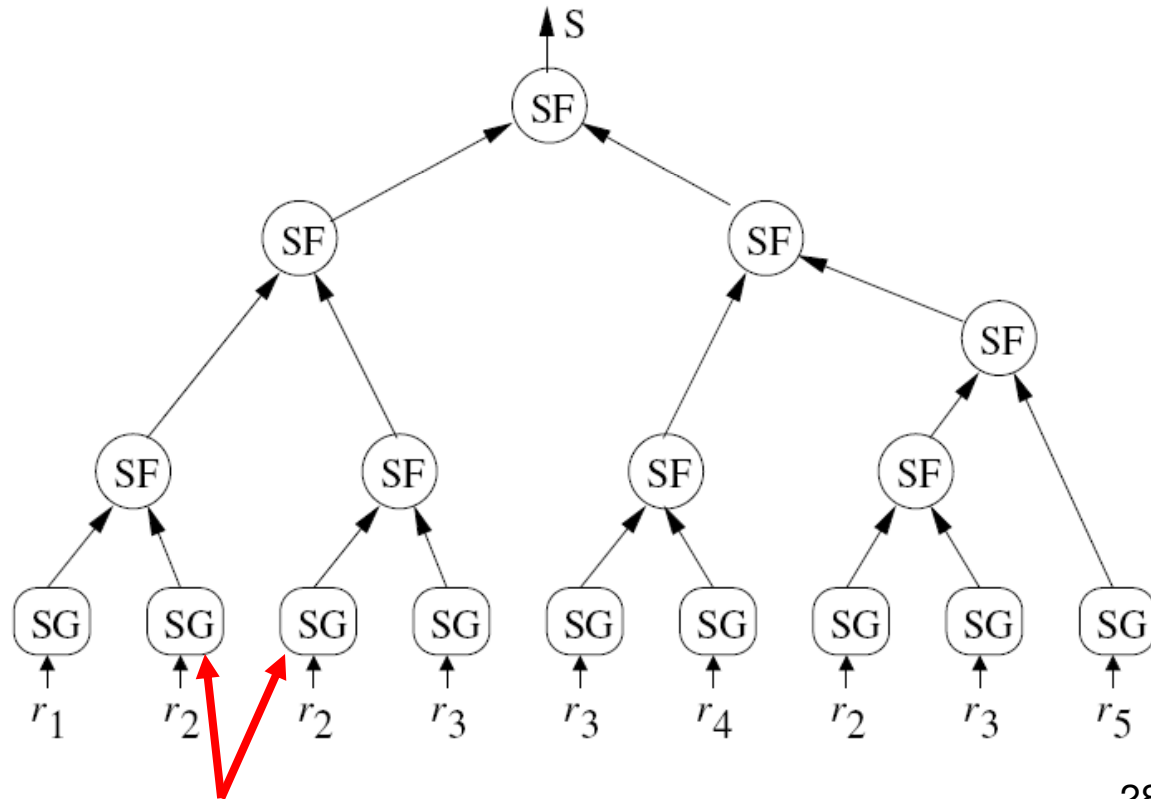
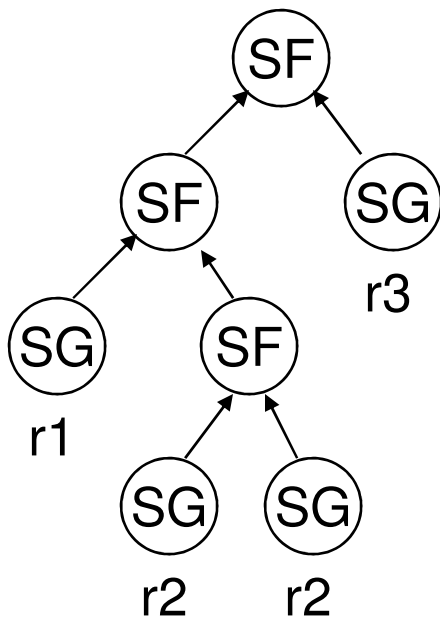
2. Re-order the leaf nodes by the increasing value of the synopsis. ← **Commutative**.



Proof of ODI correctness

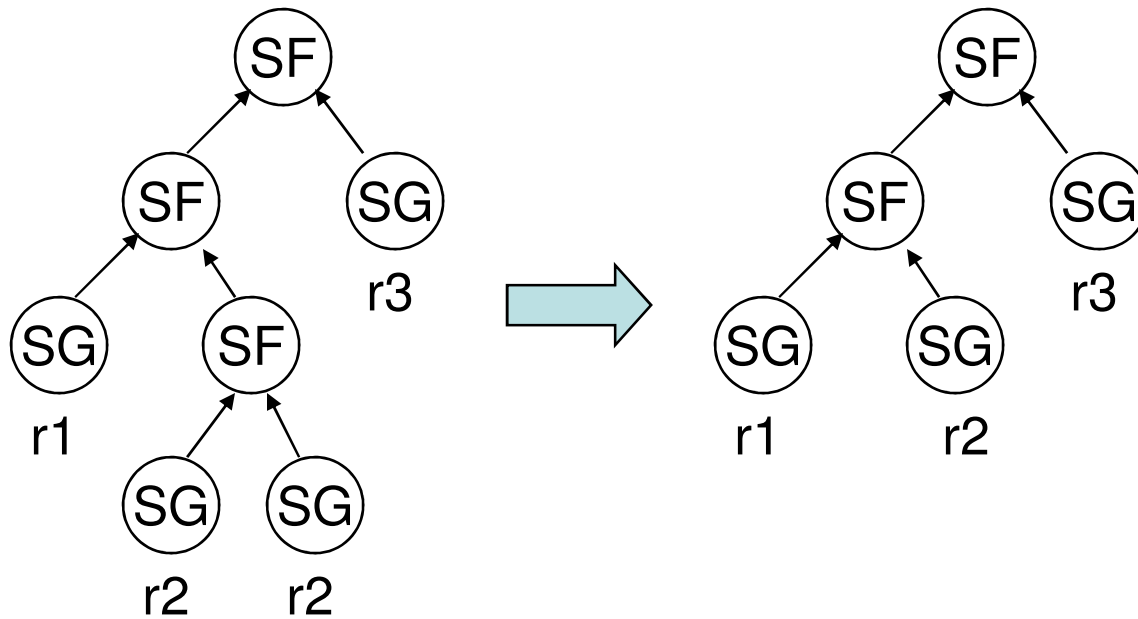
3. Re-organize the tree s.t. adjacent leaves with the same value are input to a SF function. ←

Associative.



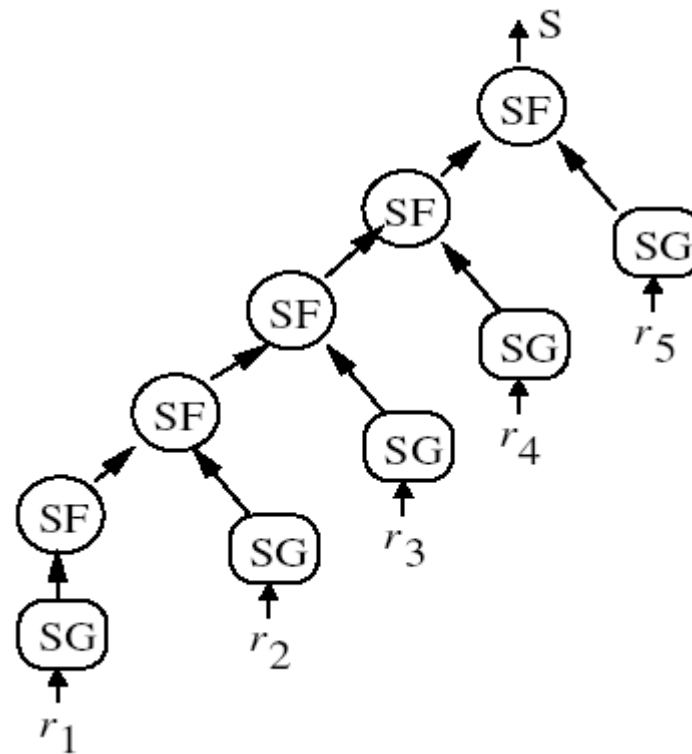
Proof of ODI correctness

4. Replace $SF(s, s)$ by s . ← same-synopsis idempotent.



Proof of ODI correctness

5. Re-order the leaf nodes by the increasing canonical order. ← **Commutative**.
6. QED.



Design ODI synopsis

- Recall that MAX/MIN are ODI.
- Translate all the other aggregates (COUNT, SUM, etc.) by using MAX.
- Let's first do COUNT.
- Idea: use probabilistic counting.
- Counting distinct element in a multi-set. (Flajolet and Martin 1985).

Counting distinct elements

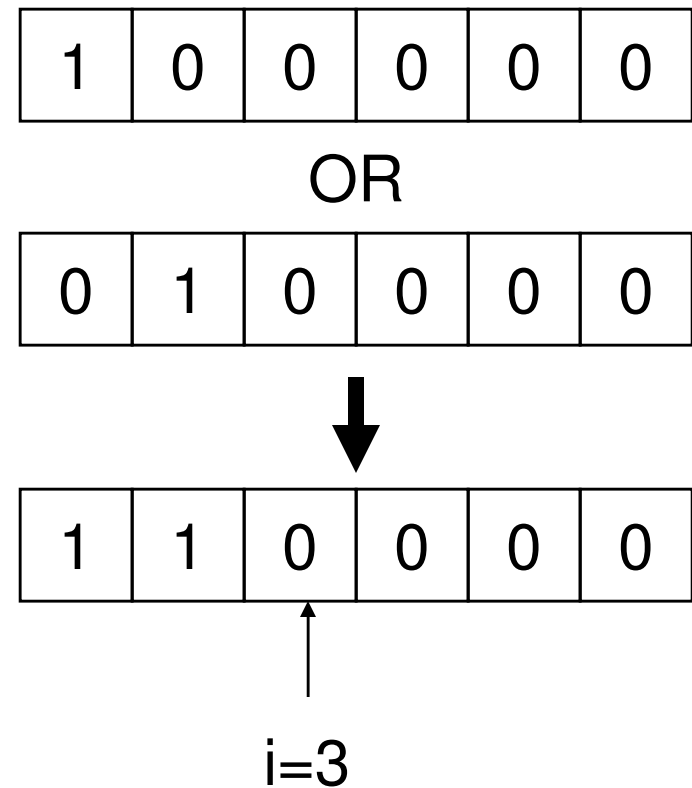
- Each sensor generates a sensor reading. Count the total number of different readings.
- Counting distinct element in a multi-set. (Flajolet and Martin 1985).
- Each element chooses a random number $i \in [1, k]$.
- $\Pr\{\text{CT}(x)=i\} = 2^{-i}$, for $1 \leq i \leq k-1$. $\Pr\{\text{CT}(x)=k\} = 2^{-(k-1)}$.
- Use a pseudo-random generator so that $\text{CT}(x)$ is a hash function (deterministic).

1	0	0	0	0	0
---	---	---	---	---	---

$\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{16}$

Counting distinct elements

- Synopsis: a bit vector of length $k > \log n$.
- $SG()$: output a bit vector s of length k with $CT(k)$'s bit set.
- $SF()$: **bit-wise boolean OR** of input s and s' .
- $SE()$: if i is the lowest index that is still 0, output $2^{i-1}/0.77351$.
- Intuition: i -th position will be 1 if there are 2^i nodes, each trying to set it with probability $1/2^i$



Distinct value counter analysis

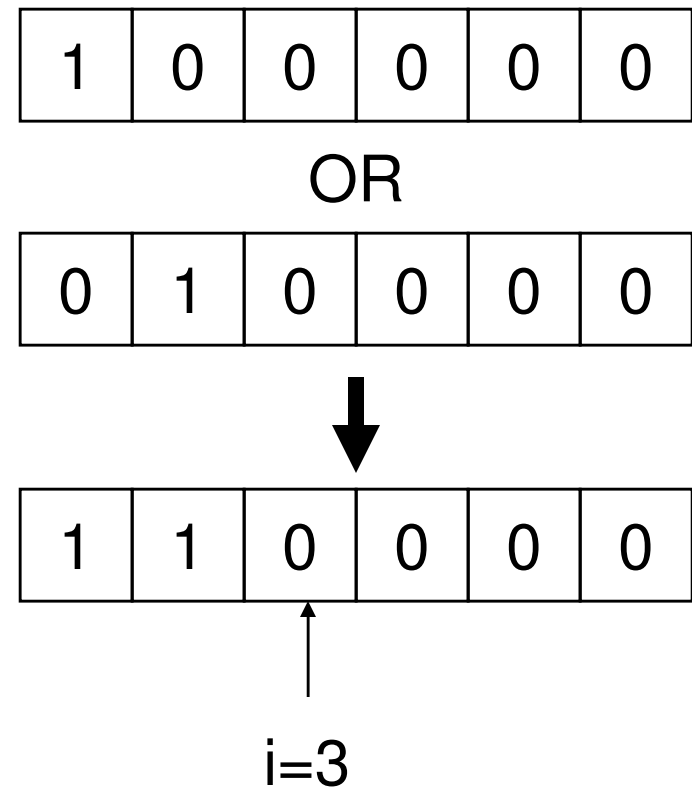
- Lemma: For $i < \log n - 2 \log \log n$, $FM[i] = 1$ with high probability (asymptotically close to 1). For $i \geq 3/2 \log n + \delta$, with $\delta \geq 0$, $FM[i] = 0$ with high probability.



- The expected value of the first zero is $\log(0.7753n) + P(\log n) + o(1)$, where $P(u)$ is a periodic function of u with period 1 and amplitude bounded by 10^{-5} .
- The error bound (depending on variance) can be improved by using multiple copies or stochastic averaging.

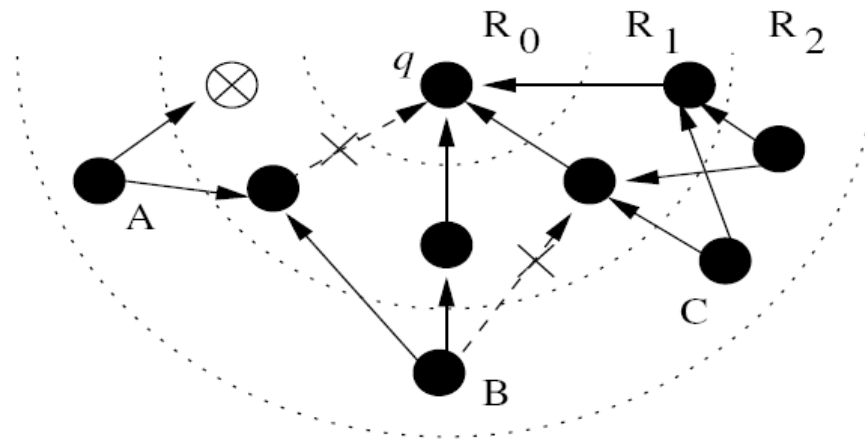
Counting distinct elements

- Check the ODI-correctness:
 - Duplication: by the hash function. The same reading x generates the same value $CT(x)$.
 - Boolean OR is commutative, associative, same-synopsis idempotent.
- Total storage: $O(\log n)$ bits.



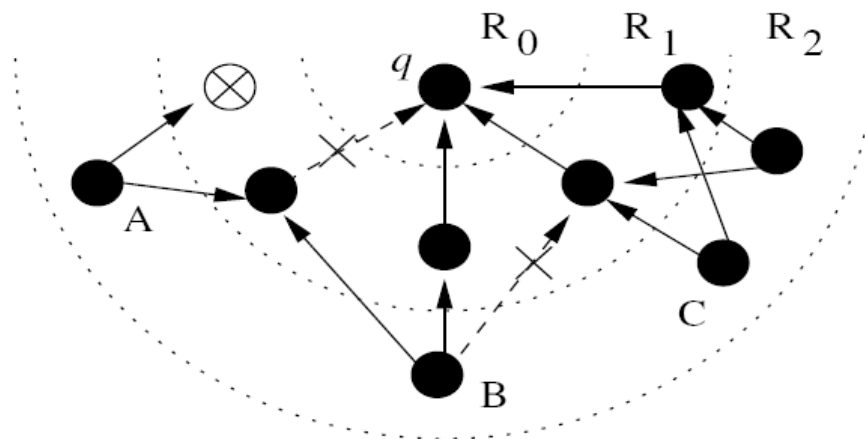
Robust routing + ODI

- Use Directed Acyclic Graph (DAG) to replace tree.
- Rings overlay:
 - Query distribution: nodes in ring R_j are j hops from q .
 - Query aggregation: node in ring R_j wakes up in its allocated time slot and receives message from nodes in R_{j+1} .



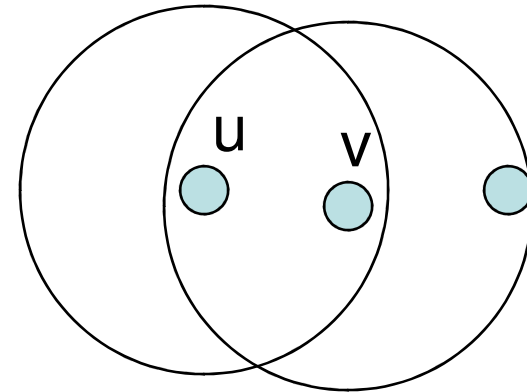
Rings and adaptive rings

- Adaptive rings: cope with network dynamics, node deletions and insertions, etc.
- Each node on ring j monitor the success rate of its parents on ring $j-1$.
- If the success rate is low, the node connects to other node whose transmission is overheard a lot.
- Nodes at ring 1 may transmit multiple times to ensure robustness.



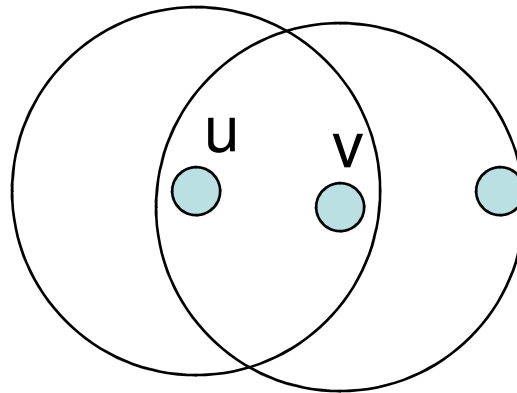
Implicit acknowledgement

- Explicit acknowledgement:
 - 3-way handshake.
 - Used for wired networks.
- Implicit acknowledgement:
 - Used on ad hoc wireless networks.
 - Node u sending to v snoops the subsequent broadcast from v to see if v indeed forwards the message for u.
 - Explores broadcast property, saves energy.
- With aggregation this is problematic.
 - Say u sends value x to v, and subsequently hears value z.
 - U does not know whether or not x is incorporated into z.



Implicit acknowledgement

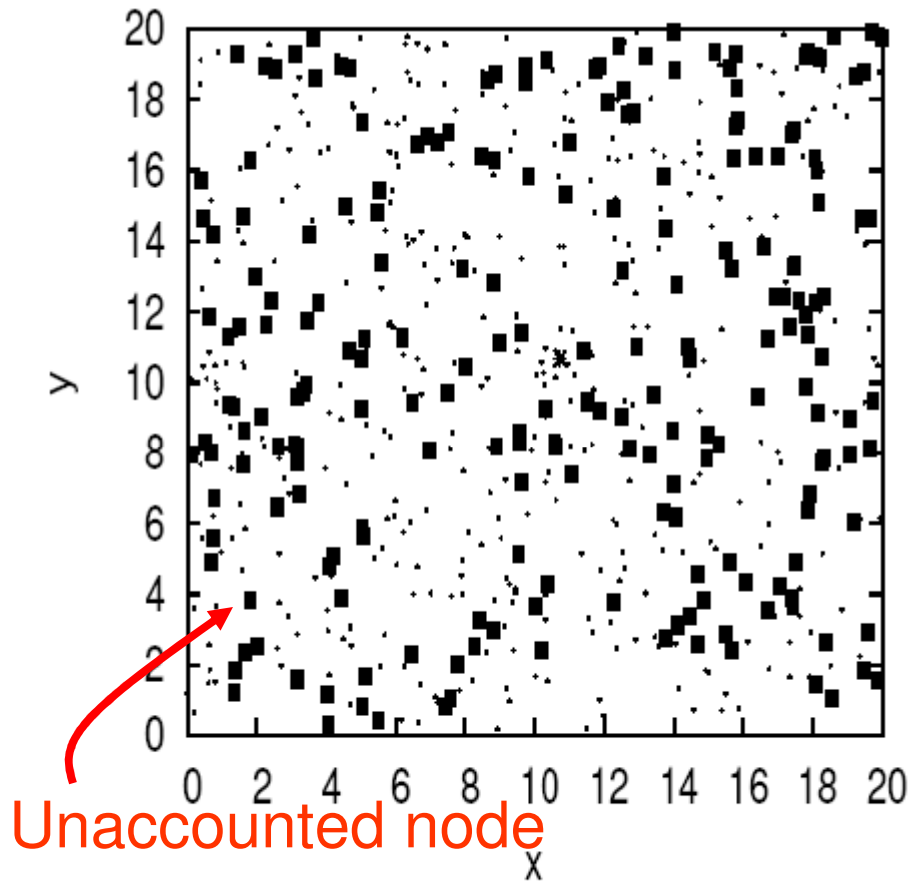
- ODI-synopsis enables efficient implicit acknowledgement.
 - u sends to v synopsis x.
 - Afterwards u hears that v transmitting synopsis z.
 - u verifies whether $SF(x, z)=z$?



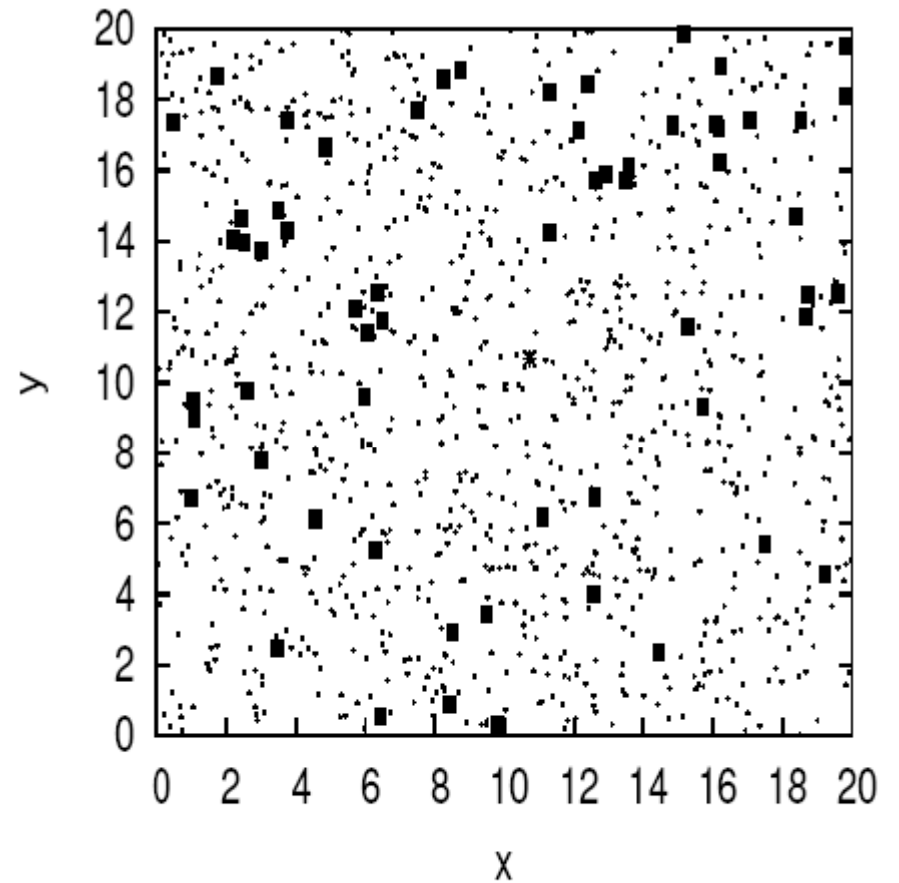
Error of approximate answers

- Two sources of errors:
 - Algorithmic error: due to randomization and approximation.
 - Communication error: the fraction of sensor readings not accounted for in the final answer.
- Algorithmic error depends on the choice of algorithm and thus relatively controllable.
- Communication error depends on the network dynamics and robustness of routing algorithms.

Simulation results



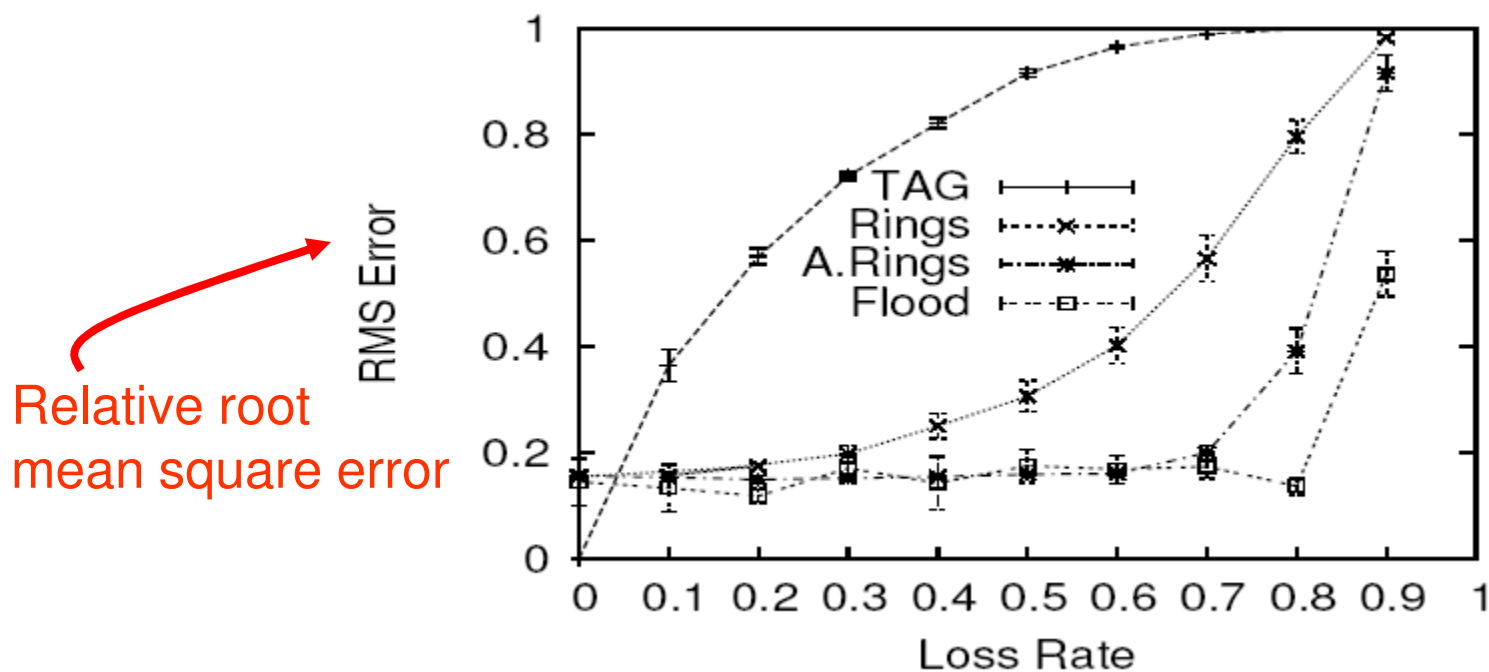
(a) Rings



(b) Adaptive Rings

Simulation results

Scheme	% nodes	Error(Uniform)	Error(Skewed)
TAG	< 15%	0.87	0.99
TAG2	N/A	0.85	0.98
RINGS	65%	0.33	0.19
ADAPT. RINGS	95%	0.15	0.16
FLOOD	$\approx 100\%$	0.13	0.13



More ODI synopsis

- Distinct values
- SUM
- Second moment
- Uniform sample
- Most popular items
- Set membership --- Bloom Filter

Sum

- Naïve approach: for an item x with value c times, make c distinct copies (x, j) , $j=1, \dots, c$. Now use the distinct count algorithm.
- When c is large, we set the bits as if we had performed c successive insertions to the FM sketch.
- First set the first $\delta = \log c - \log \log c$ bits to 1.
- Those who reached δ follow a binomial distribution: each item reaches δ with prob $2^{-\delta}$.
- Explicitly insert those that reached bit δ by coin flipping.
- Powerful building block.

Second moment

- Kth moment $\mu_k = \sum x_i^k$, x_i is the number of sensor readings (frequency) of value i .
 - μ_0 is the number of distinct elements.
 - μ_1 is the sum.
 - μ_2 is the square of L_2 norm (variance, skewness of the data).
- The sketch algorithm for frequency moments can be turned into an ODI easily by using ODI-sum.

[The space complexity of approximating the frequency moments](#),
N. Alon, Y. Matias, and M. Szegedy. *STOC 1996*.

Second moment

- Random hash $h(x): \{0,1,\dots,N-1\} \rightarrow \{-1,1\}$
- Define $z_i = h(i)$
- Maintain $X = \sum_i x_i z_i$
- $E(X^2) = E(\sum_i x_i z_i)^2 = E(\sum_i x_i^2 z_i^2) + E(\sum_{i,j} x_i x_j z_i z_j)$.
- Choose the hash function to be pairwise independent: $\Pr\{h(i)=a, h(j)=b\} = 1/4$.
- $E(z_i^2) = 1$, $E(z_i z_j) = E(z_i) E(z_j) = 0$.
- Now $E(X^2) = \sum_i x_i^2$.

- ODI: Each sensor generates $x_i z_i$, then use ODI-sum.