Lecture 8:

ODI Synopsis, Probabilistic Counting and Multipath Aggregation

Rik Sarkar

Data aggregation

- The TinyDB approach:
- Compute aggregation tree
- Transmit data along the tree, with aggregation at intermediate nodes
- Problem: Susceptible to failures/errors
- If a message is lost, the entire aggregate it was carrying is lost

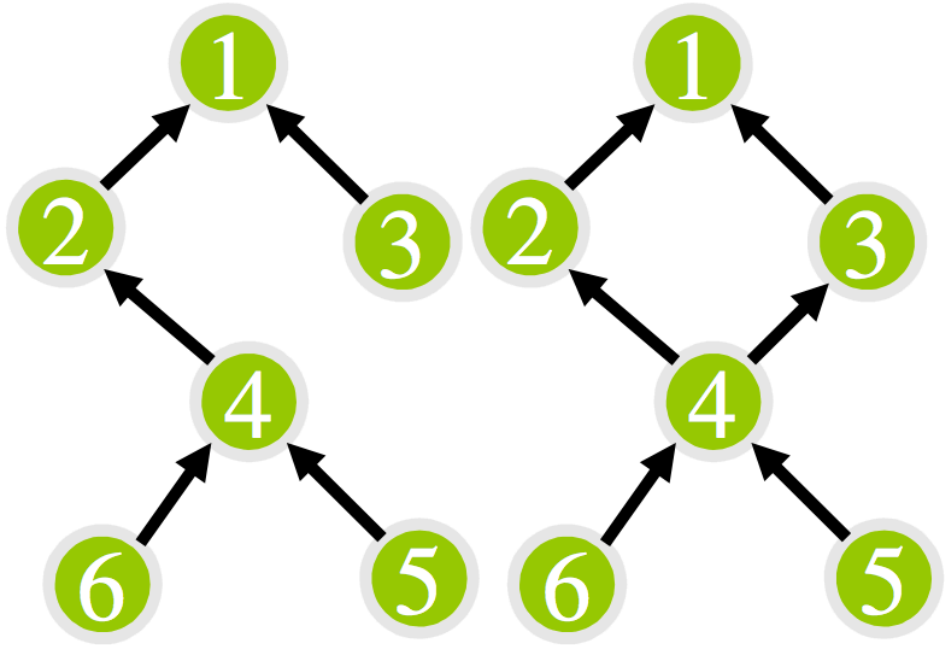
Papers

- Suman Nath, Phillip B. Gibbons, Zachary Anderson, and Srinivasan Seshan,
  Synopsis Diffusion for Robust Aggregation in Sensor Networks. SenSys'04

- Jeffrey Considine, Feifei Li, George Kollios and John Byers,
  Approximate Aggregation Techniques for Sensor Databases, ICDE 2004.

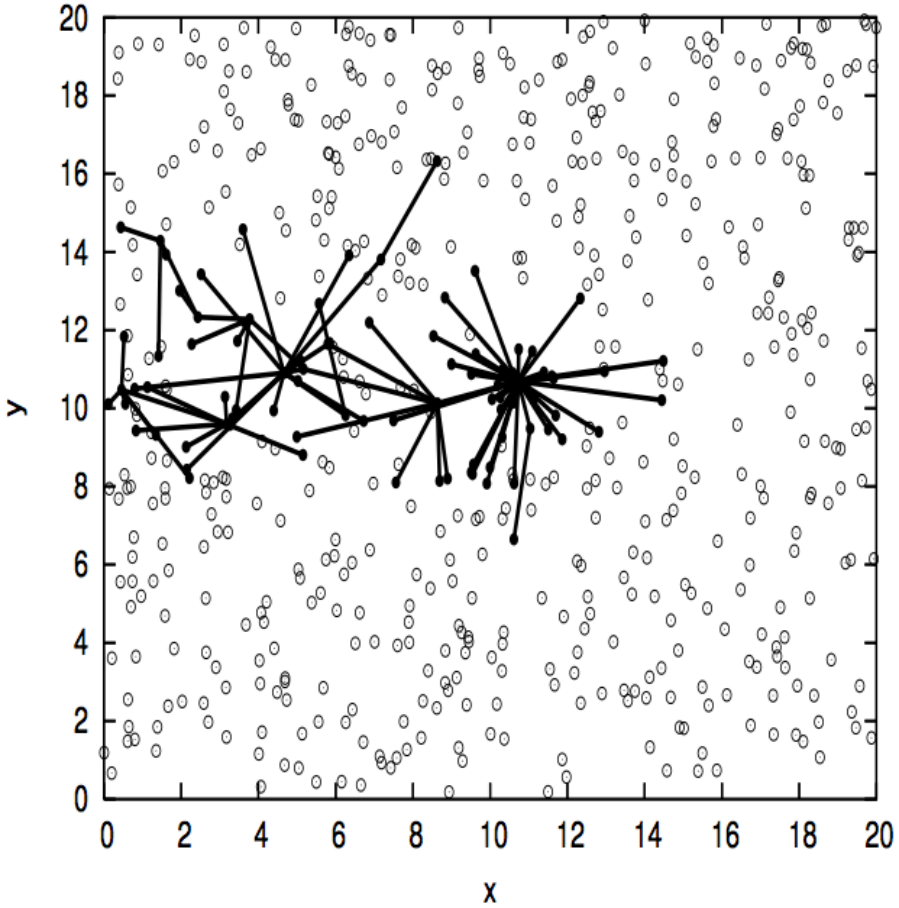
Aggregation by DAG

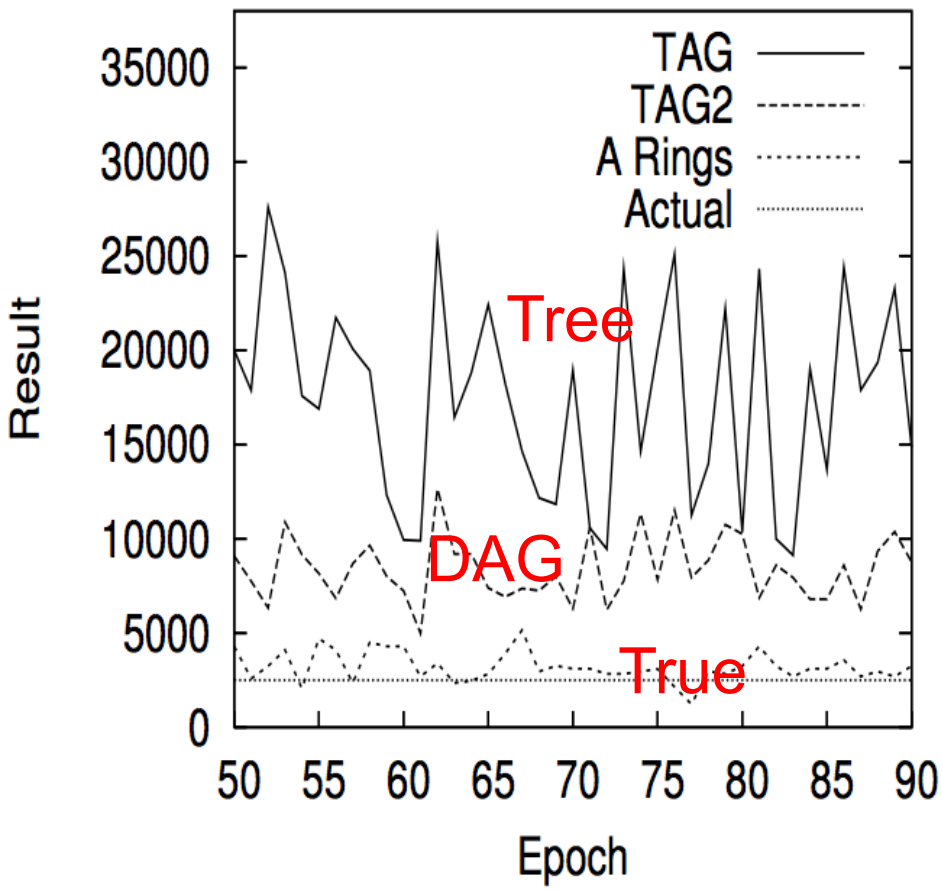
- Use a DAG instead of a tree.
- Each node has multiple parent, therefore multiple paths to the sink
- Send 1/k data to each parent
- A message failure loses at 1/k data instead of all data



Experiments on Tree and DAG

Radiation strength at center aggregated over all nodes





Aggregation and routing

- Loss of message loses data
- DAG method is better, but still loses data depending on routing quality
- Can we have a method whose accuracy depends less on routing quality?
- Sending the entire data to all parents is safer
  - Sink will get the entire data if one arrives
  - Problem: Same data may arrive multiple times
  - Same data may be added multiple times

Order and duplicate insensitive (ODI) synopsis

- Synopsis: Small description
- ODI synopsis : representation of aggregates
- Order insensitive
  - Description does not change with order of input
- Duplicate Insensitive
  - Description does not change with duplication of input
- E.g. Max, Min
- What happens for count, sum ?

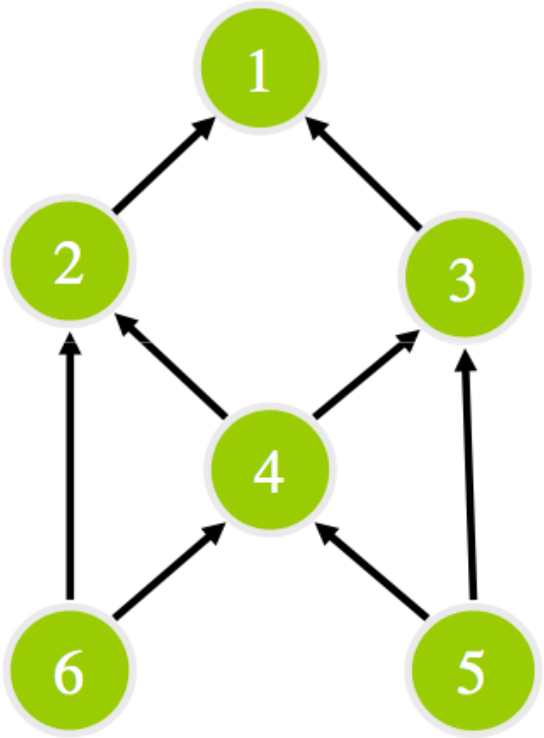
Aggregation method

- Fault tolerant aggregation
- Use multi path routing to guarantee delivery
- Use ODI synopsis to prevent double counting

- Leaf nodes: **Synopsis generation**: SG(.).
- Internal nodes: **Synopsis fusion**: SF(.) combines two synopsis to create a new one.
- Root nodes: **Synopsis evaluation**: SE(.) decodes the actual aggregate from synopsis.

Example: Synopsis for Max, Min

- **Synopsis generation**: SG(.)
  - The value itself
- **Synopsis fusion**: SF(.)
  - Take the Max/Min of the two inputs
- **Synopsis evaluation**: SE(.)
  - Output the synopsis



Building ODI Scheme

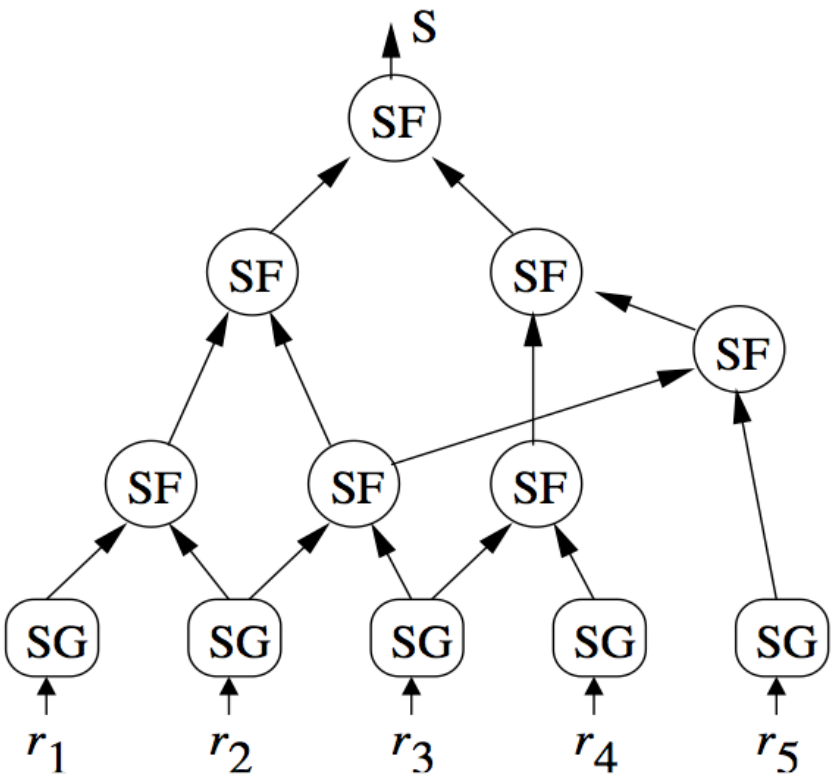
- What are the properties we need from synopsis?
- How to combine robust routing with synopsis
- How to design ODI synopsis?
  - Count
  - Sum

Correctness of ODI

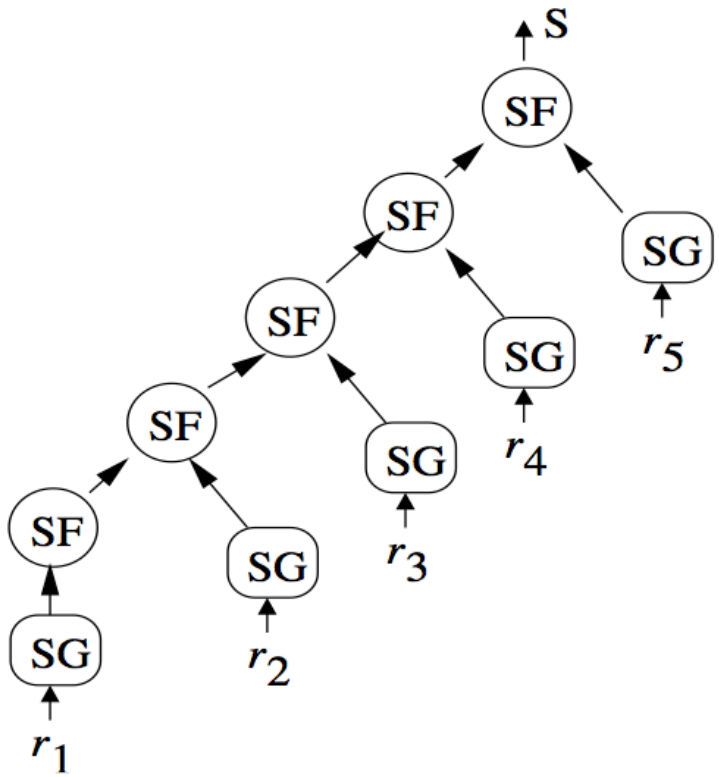
- A synopsis diffusion algorithm is ODI correct if SF () and SG() are order and duplicate insensitive
- For any aggregation DAG, the final synopsis must be identical to the synopsis in a canonical left-deep tree

- The output of SE must be independent of routing scheme
  - Independent of order of message arrival
  - Independent of message duplication

ODI correctness and canonical representation



(a) Aggregation DAG



(b) Canonical left-deep tree

Test for ODI correctness

- **SG(): preserves duplicates**: if v is constant, SG(v) is constant: nodes with same data generate same synopsis
- **SF() is commutative**
- **SF() is associative**
- **SF() is same synopsis idempotent, SF**(s,s)=s

Theorem: These properties are necessary and sufficient.

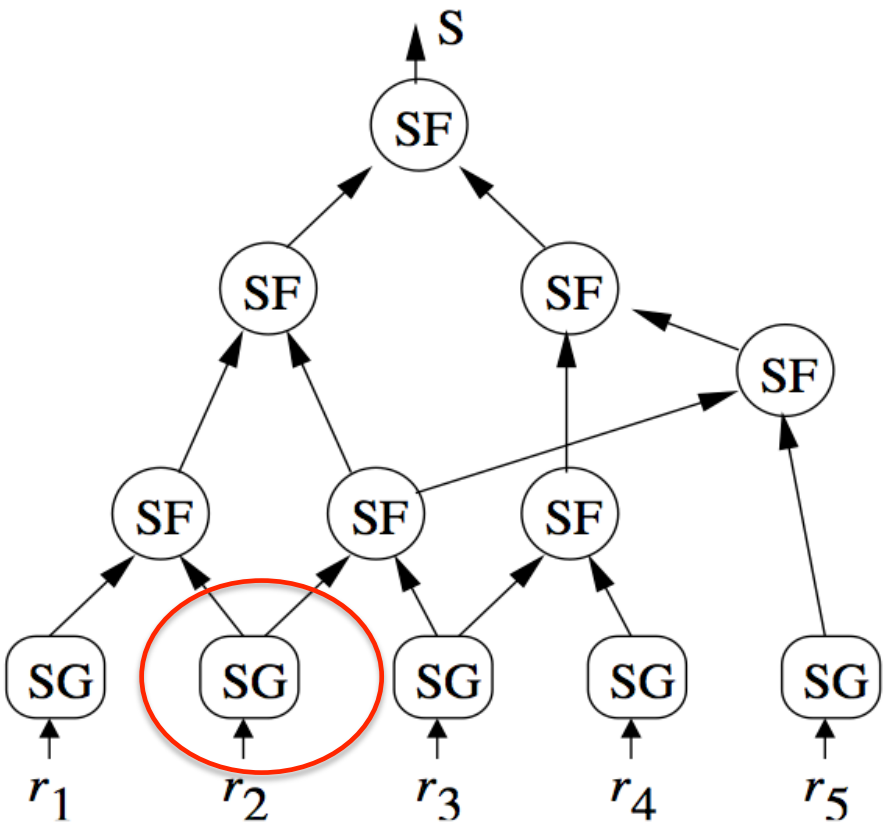
Proof: Show that any DAG can be converted to a canonical left-deep tree with same output

Proof of ODI correctness

- Start from the given DAG.
- Duplicate a node with out degree k into k nodes

Duplicate preserving

S remains same

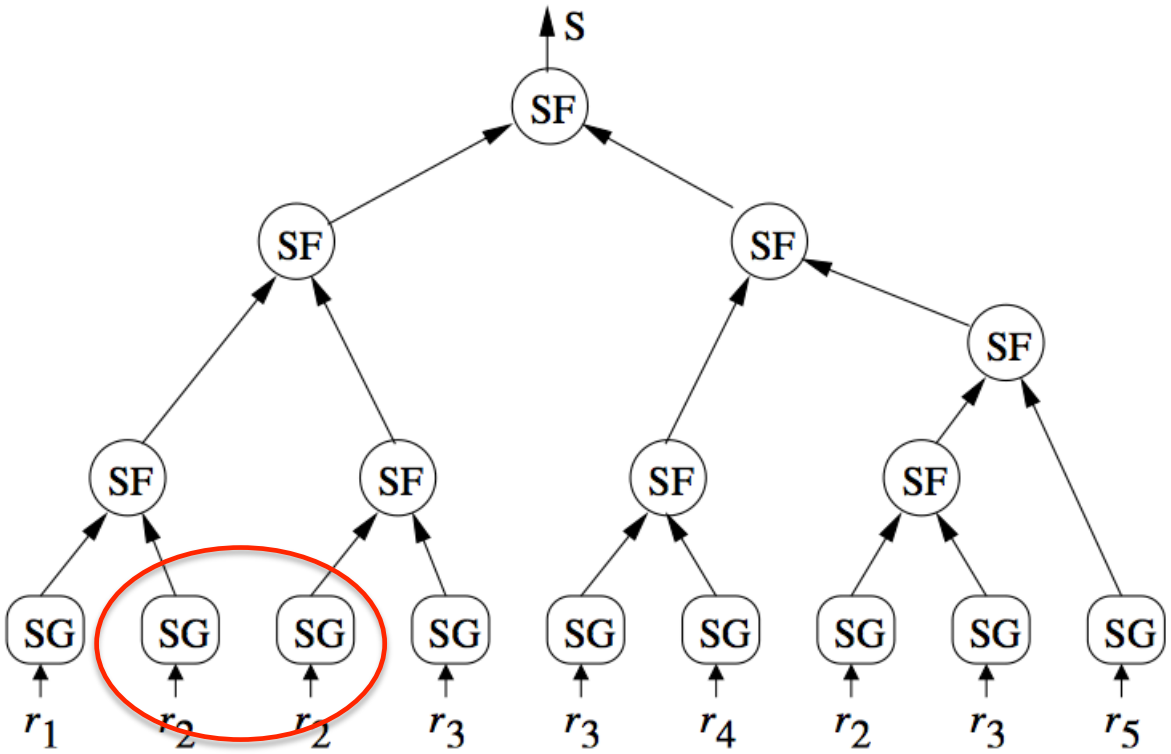


Proof of ODI correctness

- Reorder leaf nodes by increasing value of synopsis

Commutative, associative.

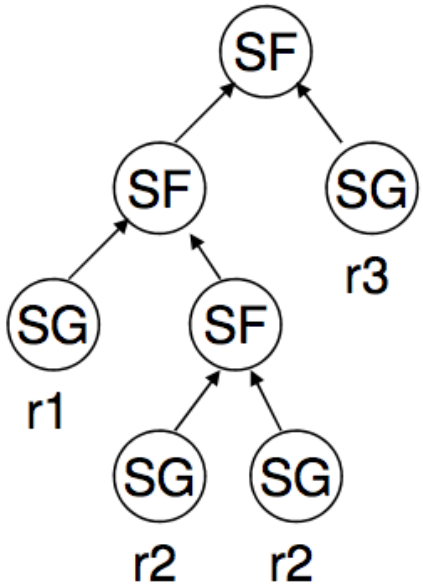
S remains same

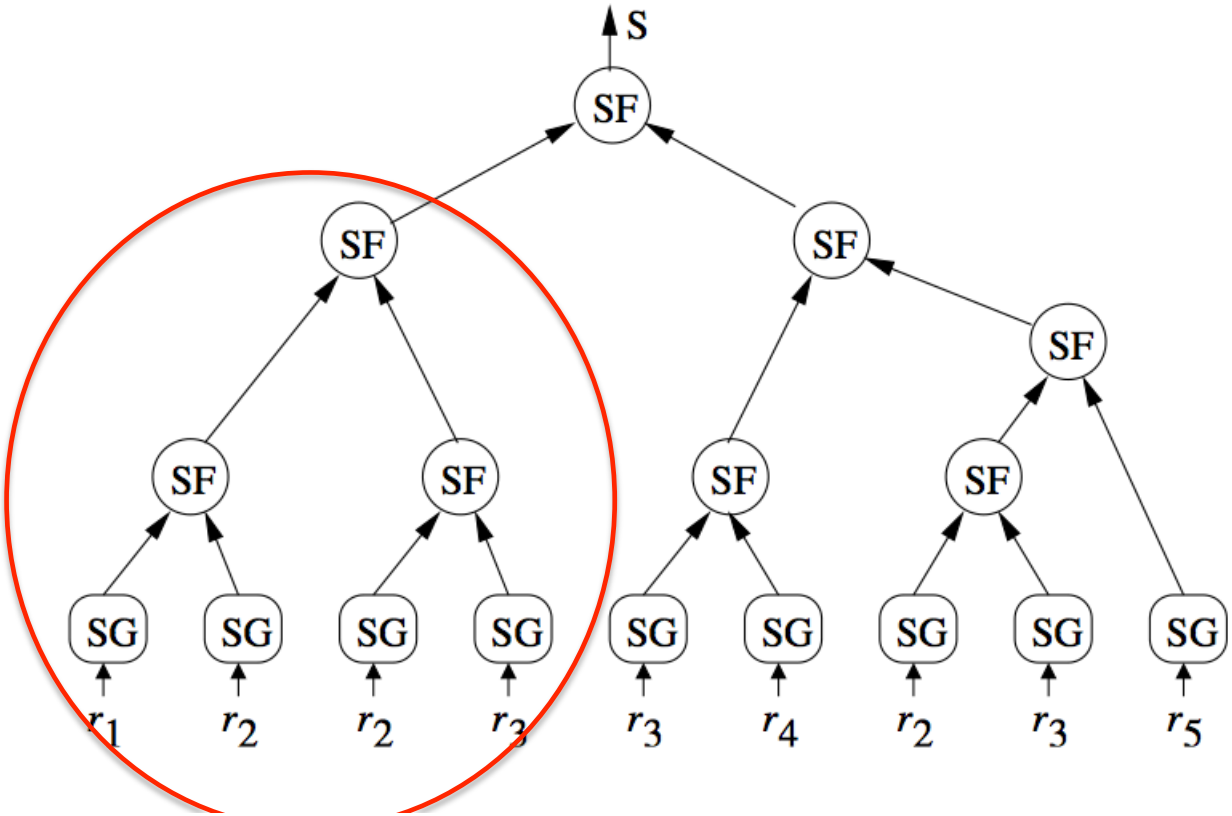


Proof of ODI correctness

- Re-organize the tree s.t. adjacent leaves with same value are are input to same SF node

Associative.
S remains same.



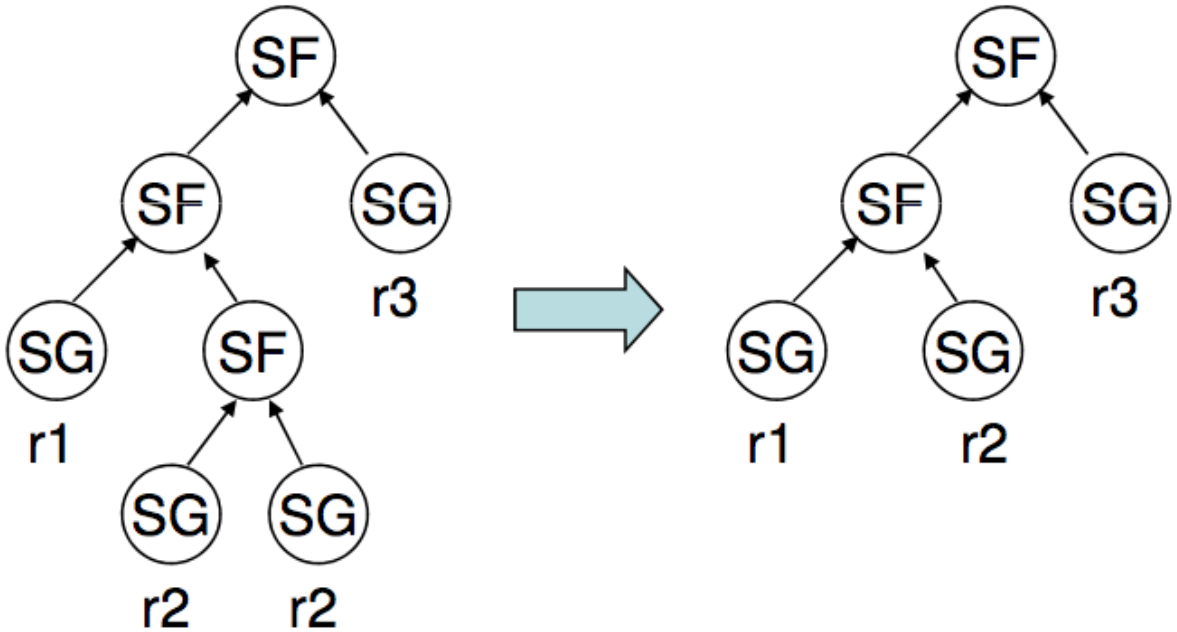


Proof of ODI correctness

- Replace $SF(s_2, s_2)$ by s_2

Same synopsis idempotent.

S remains same.



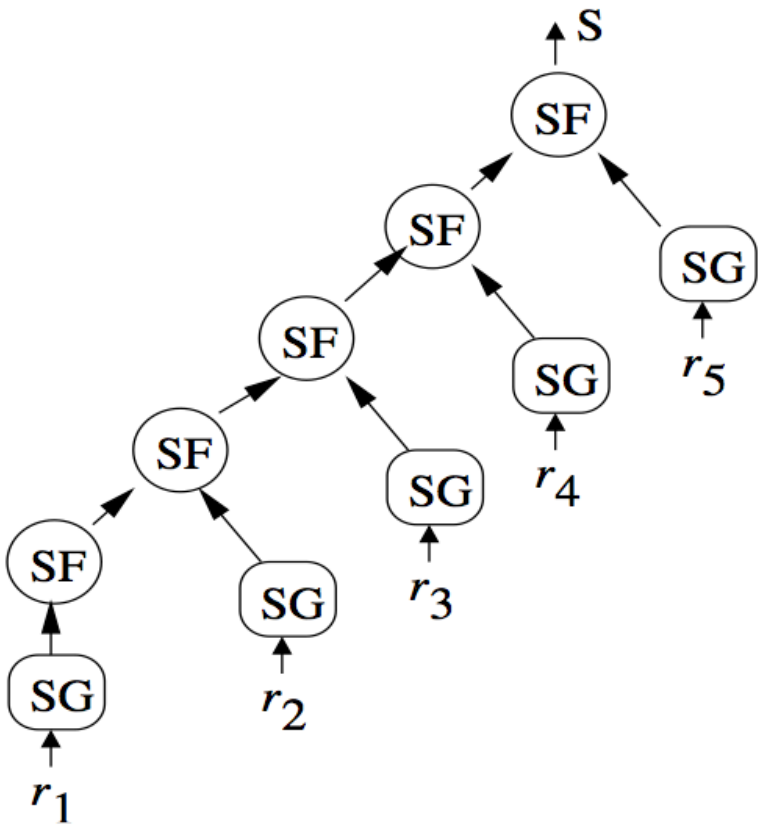
Proof of ODI correctness

- Reorder leaves by increasing canonical order

Commutative, associative.

S remains same.

QED!



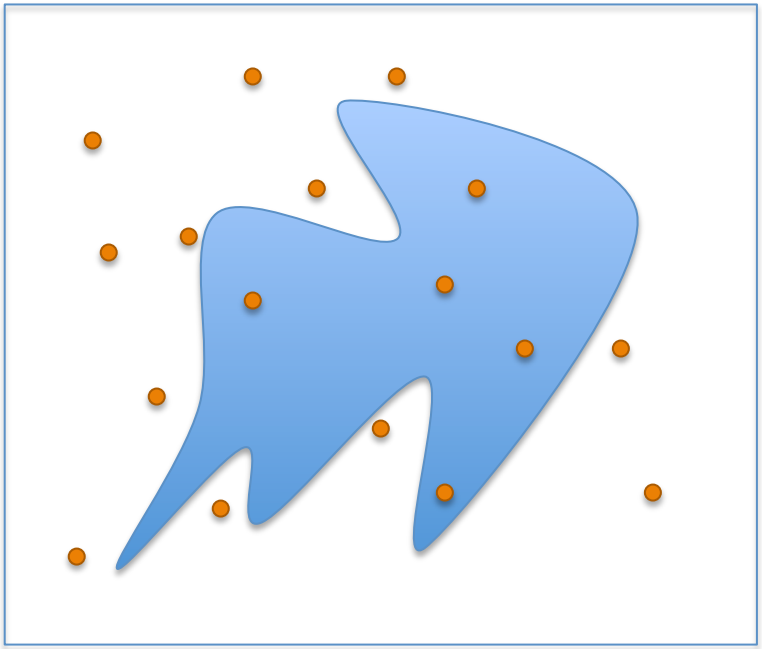
Designing ODI synopsis

- We saw that Max, Min are ODI
- Translate all other aggregates sum, count etc to computation of Min

- Counting number of distinct elements in a multi-set
- Flajolet - Martin sketches : 1985

A detour: Probabilistic estimates

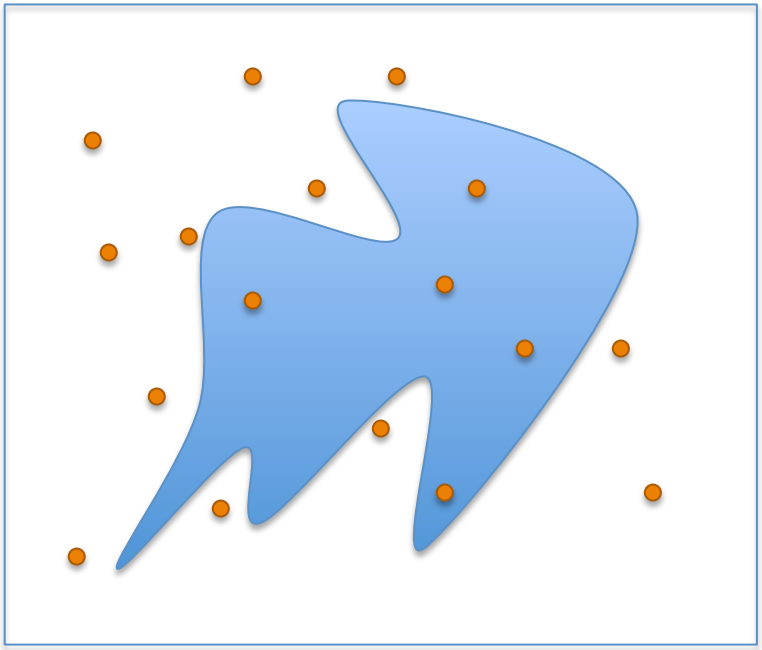
- What is the area of this region?
- Take a rectangle around it
- Throw random points
- See how many are inside
- Gives estimate of area



A detour: Probabilistic estimates

Fraction of points inside = fraction of area covered (in the limit)

More points gives better accuracy.



ODI synopsis: Count distinct

- Each sensor generates a sensor reading. Count the total number of different readings

- Use a bit vector S to encode (x)
- Use a hash function

```
i = 0;
while hash(x,i) = 0 do
    i = i + 1;
end while
S[i] = 1;
```

ODI computation (SG)

- Set S[0] = 1 with probability ½
- If S[0] == 0, set S[1] = 1 with probability ½
- ....

```
i = 0;
while hash(x,i) = 0 do
    i = i + 1;
end while
S[i] = 1;
```

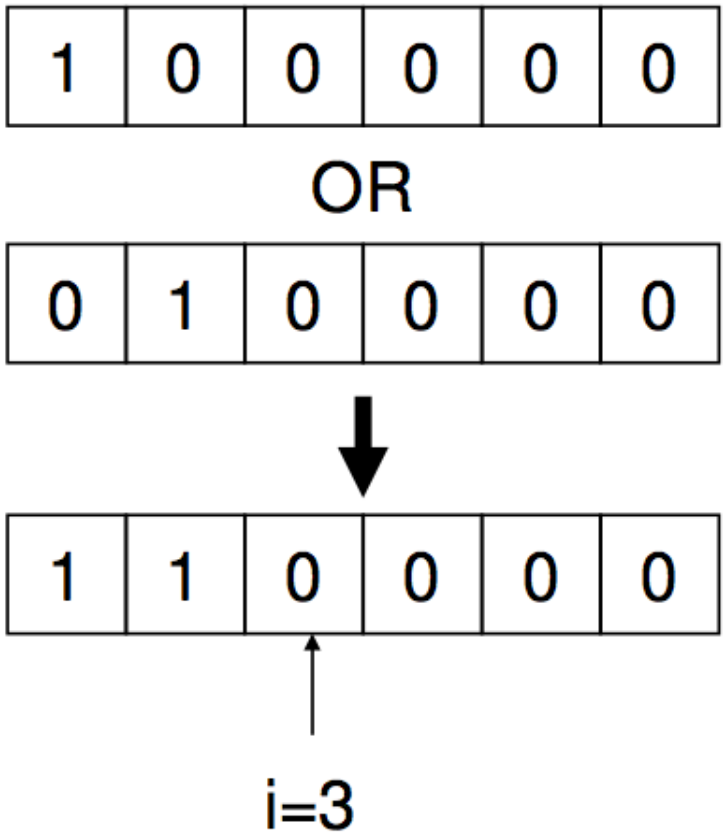
Result : Earlier bits are likely to be set, while later bits are Unlikely to be set

1	0	0	0	0	0

½ ¼ 1/8 1/16 .....

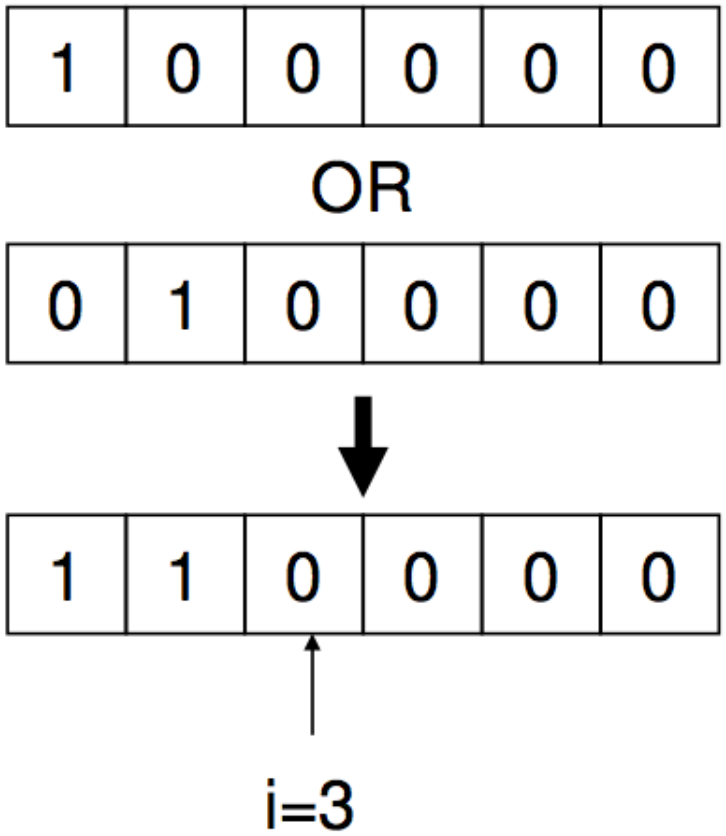
ODI computation (SF)

- Synopsis: bit vectors
- SG : bit vectors computed by hash
- **SF : bitwise boolean OR**



ODI computation (SE)

- Synopsis: bit vectors
- SG : bit vectors computed by hash
- SF : bitwise boolean OR
- **SE : if i is the lowest index that is still 0, output $2^{i-1}/0.775351$**
- Idea : a higher I means that there are more distinct elements



How accurate is this method?

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

- The expected position of the first zero is $\log(0.775351n) + P(\log n)+o(1)$, where $P()$ is a periodic function with amplitude $< 10^{-5}$
- Error bound can be improved by using multiple copies

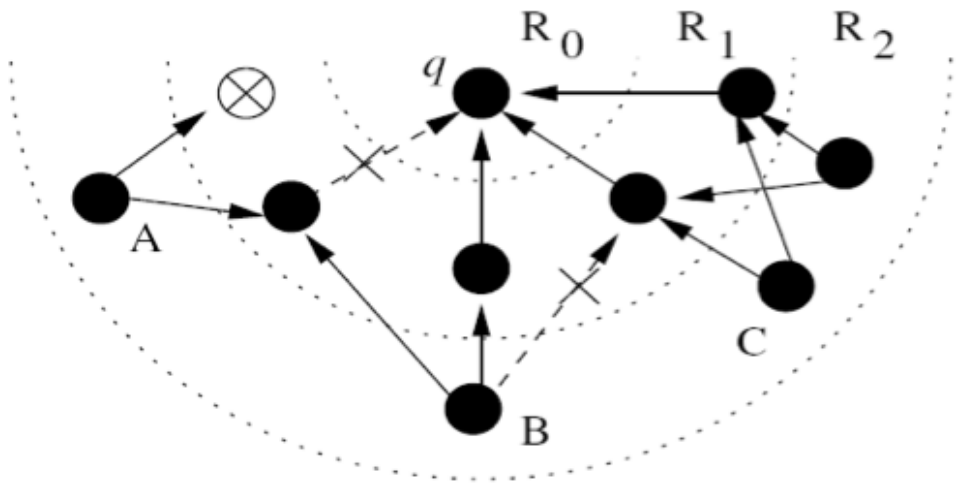
Counting distinct elements

- ODI correctness
- Duplicate insensitive. Using hash function, we get same SG(x) for x every time
- Boolean OR is commutative, associative and same synopsis idempotent

- Total storage O(logn) bits

Robust Routing with ODI

- Use DAG in place of Tree
- Rings :
  - Nodes in ring j are j hops from query node q.
  - Aggregation: Nodes in ring j receive messages from nodes in ring j+1



Adaptive rings

- Handling network dynamics, node deletions etc
- Nodes at ring j can overhear their parents' transmissions
- They monitor success rates of parents (no need for Acks!)
- If success rates are low, it connects to other nodes and sets them as parents
- Nodes at ring 1 transmit multiple times to ensure that the message gets across

Implicit acknowledgements

- Explicit acknowledgements
  - 3 way handshake
  - Used in wired networks
- Implicit acks
  - Used in wireless (broadcasts)
  - Suppose u sends message to v
  - u can also listen to v's transmission and check that u's value has been aggregated
  - Saves energy, time of sending acks
- Does not work with usual aggregates
  - U sends value 5 to v. Later hears v transmitting 20
  - U does not know if the transmission has been included

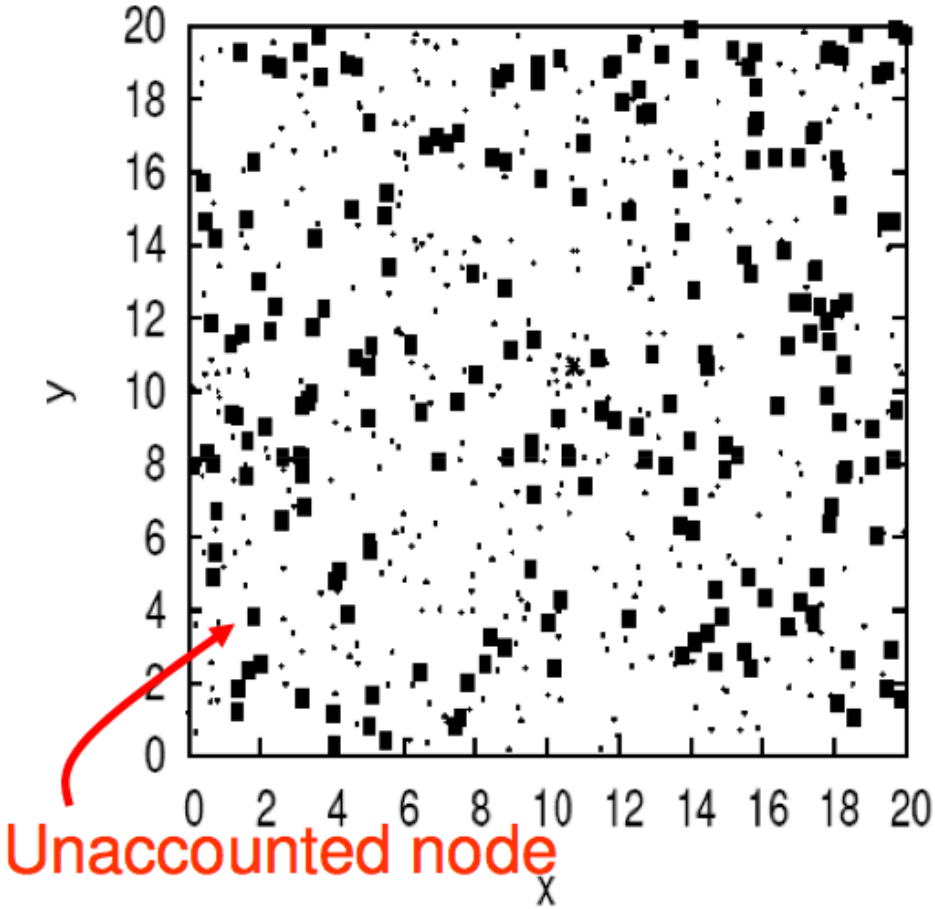
Implicit acknowledgement

- Synopsis enables implicit acknowledgements
  - U sends synopsis x to v
  - U hears v transmitting synopsis z
  - U checks : SF(x,z) = z?
  - If so, then the value is already included
  - Otherwise resend x.

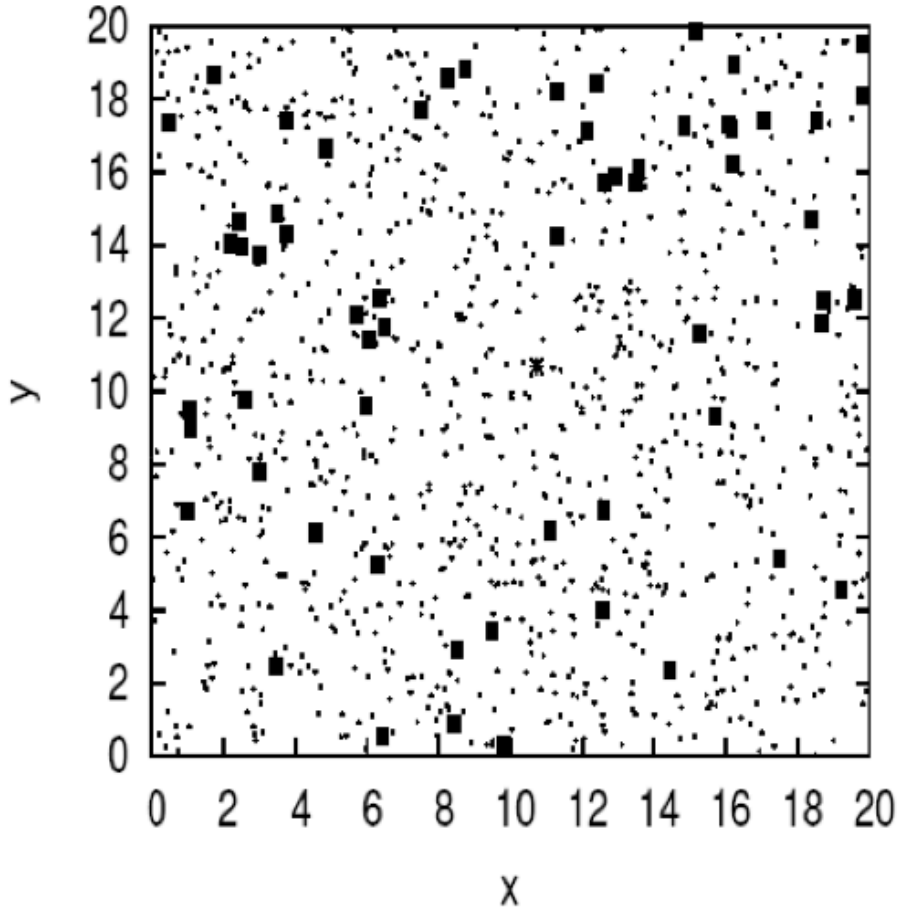
Errors in data communication

- Algorithmic error
  - Due to randomization and approximation
  - Depends on algorithm
  - Can be reduced, by choice of algorithms, and sending longer messages
- Communication error
  - Depends on network dynamics and routing algorithms

Simulations



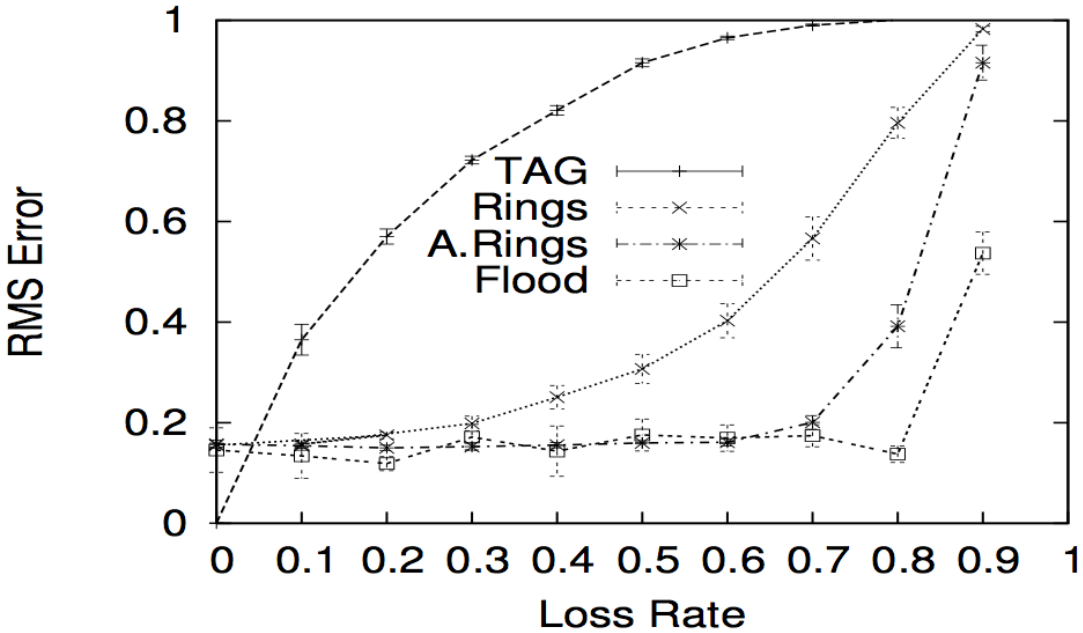
(a) Rings



(b) Adaptive Rings

Simulation: Sum computation

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



Other ODI Synopsis

- Count
- SUM
- Second moment
- Uniform sample
- Most popular items

Sum

- Simple: for a value c, make c copies and count.
- When c is large, we set bits equivalent to c successive insertions
- First set $\delta = \log c - \log\log c$ bits to 1
- An element may need more bits with prob. $2^{-\delta}$
- Insert this elements by explicitly executing the algorithm
- SUM, count are fundamental. If we can do these, we can do many other things

Second moment

- Kth moment $\mu_k = \Sigma x_i^k$, where x_i is number of elements with value I
  - μ_0 is number of distinct elements
  - μ_1 is sum
  - μ_2 is square of L_2 norm : variance

- There are sketch algorithms for frequency moments that can be turned into ODI synopsis using ODI sum