Robust Aggregation in Sensor Networks

Jie Gao
Computer Science Department
Stony Brook University
Papers


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 lose $1/k$ 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 \( \text{SF()} \) and \( \text{SG()} \) 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

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. \(\Rightarrow\) duplicates preserving.
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. \( \text{Associative.} \)
Proof of ODI correctness

4. Replace $SF(s, s)$ by $s$. $\iff$ same-synopsis idempotent.
Proof of ODI correctness

5. Re-order the leaf nodes by the increasing canonical order. \(\leftarrow\) 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.
- Each element choose 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 CT(x) is a hash function (deterministic).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1/2$</td>
<td>$1/4$</td>
<td>$1/8$</td>
<td>$1/16$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

19
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 \( s \) 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 \( 1/2^i \)

\[
\begin{array}{ccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
i=3
\]
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.

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0
\end{array}
\]

$\text{OR}$  

\[
i=3
\]
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 overhead 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

Unaccounted node
# Simulation results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>% nodes</th>
<th>Error (Uniform)</th>
<th>Error (Skewed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG</td>
<td>&lt; 15%</td>
<td>0.87</td>
<td>0.99</td>
</tr>
<tr>
<td>TAG2</td>
<td>N/A</td>
<td>0.85</td>
<td>0.98</td>
</tr>
<tr>
<td>Rings</td>
<td>65%</td>
<td>0.33</td>
<td>0.19</td>
</tr>
<tr>
<td>Adapt. Rings</td>
<td>95%</td>
<td>0.15</td>
<td>0.16</td>
</tr>
<tr>
<td>Flood</td>
<td>≈ 100%</td>
<td>0.13</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Relative root mean square error
More ODI synopsis

- Distinct values
- SUM
- Second moment
- Uniform sample
- Most popular items
- Set membership --- Bloom Filter
Naïve approach: for an item $x$ with value $c$ times, make $c$ distinct copies $(x, j), j=1, \ldots, c$. 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 $\delta$ follow a binomial distribution: each item reaches $\delta$ with prob $2^{-\delta}$.

Explicitly insert those that reached bit $\delta$ by coin flipping.

Powerful building block.
Second moment

- Kth moment $\mu_k = \Sigma x_i^k$, $x_i$ is the number of sensor readings (frequency) of value i.
  - $\mu_0$ is the number of distinct elements.
  - $\mu_1$ is the sum.
  - $\mu_2$ 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 $h(x)$: $\{0,1,\ldots,N-1\} \rightarrow \{-1,1\}$
- 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: $\Pr\{h(i)=a, h(j)=b\} = \frac{1}{4}$.
- $E(z_i^2) = 1, E(z_i z_i) = E(z_i) E(z_i) = 0.$
Uniform sample

- Each sensor has a reading. Compute a uniform sample of a given size $k$.
- Synopsis: a sample of $k$ tuples.
- SG(): output $(\text{value}, r, \text{id})$, where $r$ is a uniform random number in range $[0, 1]$.
- SF(): output the $k$ tuples with the $k$ largest $r$ values. If there are less than $k$ tuples in total, out them all.
- SE(): output the values in $s$.
- ODI-correctness is implied by “MAX” and union operation in SF().
- Correctness: the largest $k$ random numbers is a uniform $k$ sample.
Most popular items

• Return the k values that occur the most frequently among all the sensor readings
• Synopsis: a set of k most popular items.
• SG(): output (value, weight) pair, with weight=CT(k), k>\log n.
• SF(): for each distinct value v, discard all but the pair with max weight. Then output the k pairs with max weight.
• SE(): output the set of values.
• Note: we attach a weight to estimate the frequency.
• Many aggregates that can be approximated by using random samples now have ODI-synthesis, e.g., median.
Set membership: Bloom Filter

- A compact data structure to encode set containment.
- Widely used in networking applications.

- Given: n elements $S=\{x_1, x_2, \ldots, x_n\}$.
- Answer query: whether $x$ is in $S$?

- Allow a small false positive (an element not in $S$ might be reported as “yes”).
Bloom filter

• An array of \( m \) bits.
• Insert: for \( x \in S \), use \( k \) random hash functions and set \( h_j(x) \) to “1”.
• Query: to check if \( y \) is in \( S \), search all buckets \( h_j(y) \), if all “1”, answer “yes”.
• No false negative. Small false positive.
Bloom filter tricks

• Union of $S_1$ and $S_2$:
  – Take “OR” of their bloom filters.
  – ODI aggregation.

• Shrink the size to half:
  – OR the first and second halves.
Counting bloom filter

- Handle element insertion and deletion
- Each bucket is a counter.
- Insert: increase by “1” on the hashed locations.
- Delete: decrease by “1”.

- Be careful about buffer overflow.
Spectral bloom filter

- Record multi-set \( \{x_1, x_2, \ldots, x_n\} \), each item \( x_i \) has a frequency \( f_i \).
- Insert: add \( f_i \) to each bucket.
- Retrieve: return the **smallest** bucket value from the hashed locations.
- Idea: the smallest bucket is unlikely to be polluted.
Bloom filter applications

• Traditional applications:
  – Dictionary, UNIX-spell checker.

• Network applications:
  – Cache summary in content delivery network.
  – Resource routing, etc.
  – Read the survey for more…

• Good for sensor network setting:
  – ODI, compact, many algebraic properties.
Conclusion

• Due to the high dynamics in sensor networks, robust aggregates that are insensitive to order and duplication are very attractive – they provide the flexibility of using any multi-path routing algorithms and re-transmission.

• Use ODI-synopsis as black box operators to replace naïve operators in more complex data structures.
Is the problem solved? NO

• Best effort multi-path routing does not guarantee all data have been incorporated.
  – Blackbox setting.

• ODI synopsis translates everything to MAX, which is not robust to outliers!
  – Sensor malfunction.
  – Malicious attacks.

• For exemplary aggregations (MAX, MIN), the final result is a single sensor value, but all nodes are examined.