

# Error in Self-Stabilizing Spanning-Tree Estimation of Collective State

Yuanqiu Mo  
University of Iowa  
Iowa City, Iowa 52242  
Email: yuanqiu-mo@uiowa.edu

Jacob Beal  
Raytheon BBN Technologies  
Cambridge, MA, USA 02138  
Email: jakebeal@ieee.org

Soura Dasgupta  
University of Iowa\*  
Iowa City, Iowa 52242  
Email: soura-dasgupta@uiowa.edu

**Abstract**—Estimating collective state is an important component of many distributed systems, but has inherent challenges in balancing the availability of estimates against their accuracy. In this paper, we analyze the error bounds and dynamics of a commonly used family of self-stabilizing state estimation algorithms based on spanning trees. We find that in the worst case transients can duplicate values leading to exponential overestimates or can drop values leading to near total loss of information. The same analysis, however, also suggests that these problems can be mitigated by prioritizing smoothness in the adaptation of distance estimates used to maintain the spanning tree, and this mitigating effect is supported by results in simulation.

## I. INTRODUCTION

Across the vast diversity of distributed systems there are a few widely shared key patterns of interaction, such as information spreading, collective state estimation, and symmetry breaking [1], [2]. In this paper we focus on collective state estimation, a common task in many distributed systems in which values held at many individual devices are combined to produce a single value intended to quantify some property of the entire network of devices (or some defined subset). For example, collective state estimation can be used to count the number of participating devices, estimate the availability of system resources, track a set of errors and ongoing problems, or check whether there is consensus on a decision.

Efficient resilient algorithms have been found for special cases of collective state estimation, such as the use of gossip for estimating monotonic functions (e.g., [3]–[5]). General collective state estimation, however, may be cast in terms of a distributed snapshot of values and as such is subject to various of the well-known “pick two of three” impossibility results for consensus and related algorithms (e.g. [6]–[8]), which state that no algorithm can simultaneously ensure correctness, liveness, and partition tolerance. For large-scale collective adaptive systems, liveness is generally required and partition events happen frequently, which means that we must instead tolerate some degree of error in the estimates returned by collective state estimation. The challenge is to understand and manage the dynamics of error in collective state estimation well enough to predict its effects and ensure stability in distributed systems that make use of collective state estimation.

\* Prof. Dasgupta is also a Visiting Professor at Shandong Computer Science Center, Shandong Provincial Key Laboratory of Computer Networks, China

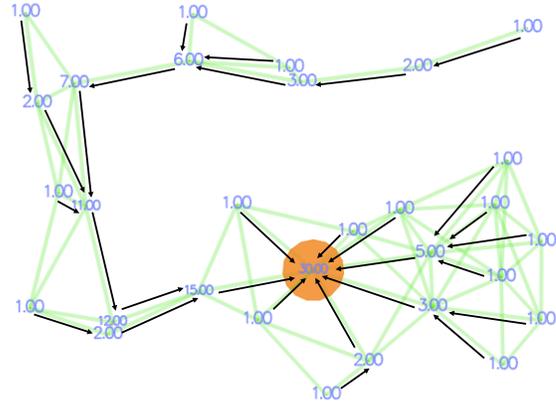


Fig. 1. Example of spanning-tree collective state estimation in simulation on a network of 30 devices: all devices compute their distance (here, by hops) from a designated *source* device (orange), then select their lowest-valued neighbor as a *parent* to create a spanning tree (arrows). Each device then reduces over its own input value and those of its children to produce an estimate (blue numbers) for its sub-tree: here the number of devices in the network is estimated by setting every device’s input value to 1 and reducing by addition.

In this paper, we focus our study on spanning-tree aggregation, one of the most frequently used approaches to collective state estimation (e.g., [9]–[13]). In particular, we present a first analysis of error dynamics in the self-stabilizing spanning-tree estimation algorithm from [1] and [14], an example of which is shown in Figure 1. We find that transient duplication of values has the potential to introduce exponential overestimates, and but that in practice estimation errors should be expected to more often be underestimates driven by transient loss of values, particularly due to the potential for persistent local minima in the potential function used in distributed construction and maintenance of a spanning tree. We confirm these results through simulation.

## II. SPANNING-TREE BASED STATE ESTIMATION

Amongst the many variants of spanning tree distributed state estimation, we focus on one introduced in [1]. That paper observed that a relatively small set of operations can serve as a basis set of “building blocks” that can be composed to cover a broad class of distributed systems. In particular, the paper identified a system of self-stabilizing [15] operators as a generalized cover for the function of a large family

```

1 // Aggregate values of type T along a spanning tree
2 // w. function reduce following maximum potential decrease.
3 // @param potential number, used to build tree
4 // @param reduce (T, T) -> T, aggregation function
5 // @param local T, local value
6 // @param null T, value of an empty aggregation; must
7 // be idempotent, i.e., reduce(v, null) = v
8 // @return T, aggregated value v
9 public def C(potential, reduce, local, null) {
10   rep (v <- local) { // Initialize estimate to local value
11     reduce.apply(local, // Combine local with children's v
12       hood( // Combine all neighbor values:
13         (a, b) -> { reduce.apply(a, b) }, // use reduce fn
14         null, // value of an empty field
15         mux(nbr(getParent(potential))==self.getDeviceUID() {
16           nbr(v) // Use values from children
17         } else { null } // Ignore others (idempotent null)
18       ))
19   }
20 }
21 def getParent(potential) {
22   // if some neighbor has lower potential...
23   mux (minHood(nbr(potential)) < potential) {
24     // ... take the lowest, tiebreaking arbitrarily by UID.
25     minHood(nbr([potential, self.v.getDeviceUID()])).get(1);
26   } else { NaN } // otherwise, device is spanning tree root
27 }

```

Listing 1. Code for C self-stabilizing spanning-tree state estimation algorithm, adapted from Protelis [18] supporting libraries.

of well-established distributed algorithms. These operators are: gradient-based information spreading (G), collective state estimation (C) symmetry-breaking (S), and temporary state (T), and all programs that can be expressed as a functional composition of these operations and “pointwise” local computation are also guaranteed to be self-stabilizing. This notion has been further developed in [16] and [14] in the framework of “aggregate programming,” including the result that these operators can be equivalently substituted by more specialized versions with better convergence dynamics, thus improving system performance. We have begun the analysis of dynamics with the use of G to estimate distance (the Adaptive Bellman-Ford algorithm) in [17], and now turn to its complement: collective state estimation with C.

Listing 1 shows a canonical implementation of operator C in Protelis, a Java-like language for aggregate programming [18]. This algorithm takes a field of local information, information for combining local information and aggregates, and a field of potential values representing a distance function from a designated *source* device (this distance computed by G or another equivalent self-stabilizing distributed distance estimation). At each (unsynchronized) round of execution, every device chooses its neighbor (neighbors being determined by the underlying communication graph) with the lowest potential as its “parent,” effectively constructing a spanning tree down the potential gradient, and then computes an estimate for its subtree as the reduction of its value and the estimates of its children. At the source, which is the root of the spanning tree, this value is a collective state estimate for the entire aggregate. C has been proved to be self-stabilizing, as must be its composition with any self-stabilizing distance estimate used to compute potential [14]. Self-stabilization, however,

only means it will eventually converge to a correct answer if inputs remain unchanged: we now turn to the dynamics of this convergence.

### III. ERROR DYNAMICS OF OPERATOR C

In this section, we first analyze the worst-case errors in the collective state estimate that can be produced by C, then extend to consider the interaction of C with the distance estimation algorithms that may be used to produce its potential input.

#### A. Maximum Error Bound on C

We first introduce two definitions:

**Definition 1.** Given a spanning tree  $T$ , a node is said to be at level  $i$  if it is  $i$  hops away from the source node.

**Definition 2.** Given two nodes  $a$  and  $b$  in a spanning tree, if  $a$  and  $b$  are connected and  $b$  is one level greater than  $a$ , then  $a$  is defined as the parent of  $b$  and  $b$  as the child of  $a$ .

In this section, we consider the perturbation of estimates computed by C caused by a spanning tree that varies with time. Specifically, in each cycle, each node may change to another parent on the same level, informing both the new and old parents when it transmits its estimate. Cycles are not synchronized, so updates can occur in any order. As such:

- A sub-tree estimate is duplicated if node  $a$  switches parent from  $b$  to  $c$ , and the nodes update in order  $b < a < c$ , since  $b$  sends an estimate including  $a$  as a child, then  $a$  notifies  $b$  and  $c$  of the switch, and finally  $c$  sends an estimate also including  $a$  as a child.
- A sub-tree estimate is lost if node  $a$  switches parent from  $b$  to  $c$  and the nodes update in order  $c < a < b$ , since  $c$  sends an estimate without  $a$  as a child, then  $a$  notifies  $b$  and  $c$  of the switch, and finally  $b$  sends an estimate also without  $a$  as a child.

We will evaluate duplication and loss potential by considering the case where C is applied to count devices, i.e., summing a local constant of 1 from each node. First, we have the following lemma:

**Lemma 1.** Under duplicating perturbation, if there is more than one node at level  $i$ , then the sum of transmitted values  $O_j$  of all nodes  $j \in S_i$ , where  $S_i$  denotes the set of nodes at level  $i$ , satisfies

$$\sum_{j \in S_i} O_j \leq 2 \sum_{j \in S_{i+1}} O_j + |S_i|$$

*Proof.* Suppose in this case, there are  $n$  nodes at level  $i$ , and the number of nodes with decreased values in the current cycle

is  $m$ . Then we have

$$\begin{aligned}
\sum_{j \in S_i} O_j &= \sum_{j=1}^m O_j + \sum_{j=m+1}^n O_j \\
&\leq \sum_{j \in S_{i+1}} O_j + m + \sum_{j=m+1}^n O_j \quad (1) \\
&\leq \sum_{j \in S_{i+1}} O_j + m + \sum_{j \in S_{i+1}} O_j + n - m \quad (2) \\
&= 2 \sum_{j \in S_{i+1}} O_j + |S_i|
\end{aligned}$$

Equality in (1) holds when in the previous cycle, all the nodes at level  $i+1$  are children of those  $m$  nodes at level  $i$  while the remaining  $n-m$  nodes at level  $i$  have no children. Equality in (2) holds when those  $m$  nodes lose all their children and all their children choose the remaining  $n-m$  nodes as their parents in the current cycle. ■

Based on Lemma 1, we have the following theorem.

**Theorem 1.** Under the perturbation described above, for a spanning tree with  $n$  nodes, value of the source node satisfies the tight bound

$$O_s \leq \begin{cases} 2^{\frac{n+1}{2}} - 1 & n \text{ is odd} \\ 2^{\frac{n}{2}} + 2^{\frac{n}{2}-1} - 1 & n \text{ is even} \end{cases}$$

where  $O_s$  denotes the value of source node.

*Proof.* First, we consider the case that all levels in the spanning tree have more than one node, then we consider the case that some or all levels have only one node, and show that value of the source node resulted from the former case will be larger than that in the latter case.

For the former case where each level has more than one node, when  $n$  is odd, suppose value of the source node is maximized when there are  $l$  levels in the spanning tree (here we assume the bottom level is level  $l$ ), according to Lemma 1,  $O_s$  satisfies

$$O_s \leq \sum_{i=1}^l 2^{i-1} |S_i| + 1 \quad (3)$$

where  $S_i$  denotes the set comprised of the nodes at level  $i$ .

As we can see from (3), the sum grows exponentially with the number of levels and grows linearly with the number of nodes at each level. Thus, value of the source node will be maximized by maximizing the number of levels, which means that each level should have two nodes and  $l = (n-1)/2$ , then value of the source node satisfies

$$\begin{aligned}
O_s &\leq \sum_{i=1}^{(n-1)/2} 2^{i-1} |S_i| + 1 \\
&= 2^{\frac{n+1}{2}} - 1 \quad (4)
\end{aligned}$$

where  $|S_i| = 2$  for  $i = 1, 2, \dots, l$ .

In (4), we assume that all values of nodes transmitted from higher levels to lower levels all get doubled. This is a tight bound, and an example is shown in Figure 2.

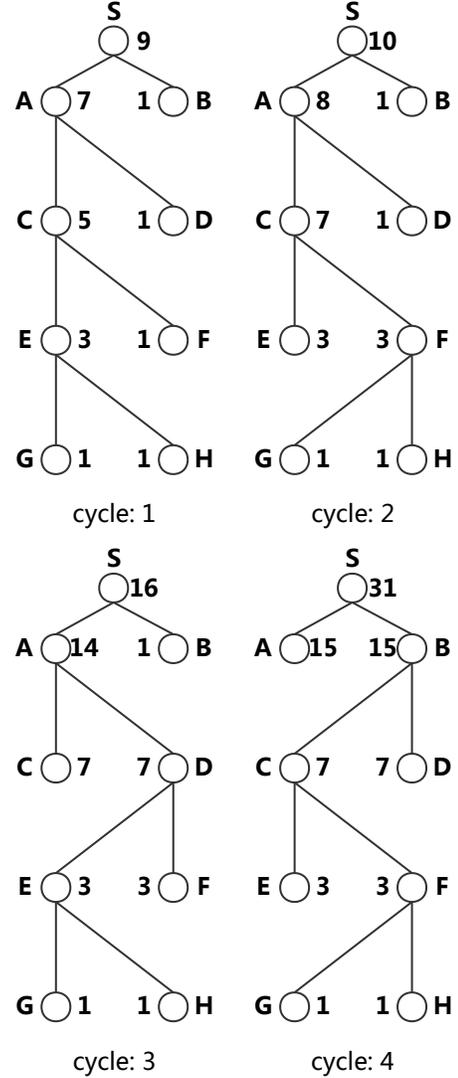


Fig. 2. Example of achieving the maximum source value

As shown in Figure 2, all nodes in the same level share the same parent in every cycle, and nodes at the highest level change their common parent back and forth from the second cycle, nodes at the second highest cycle follow this from the third cycle, and so on. The source value grows exponentially and finally achieves its maximum value defined in (4).

If the number of nodes in the graph is even and value of the source node is maximized when there are  $l$  levels in the spanning tree. Since the number of nodes is even, there will be one level with a single node. The question is that where to put this single node in order to achieve the maximum value of the source node. Here we claim that value of the source node will be maximized when the single node is at the highest level. The proof (omitted due to space constraints) follows similar reasoning as before, in essence finding that it is always better to replace two layers of single nodes with one layer of two nodes, and that such replacements are better at lower

layers. ■

### B. Minimum Error Bound on C

Next, we show the lower bound of value of the source node. First, we provide the following lemma.

**Lemma 2.** *Under loss perturbation, if there is more than one node at level  $i$ , then the sum of values of all nodes transmitted from level  $i$  satisfies*

$$\sum_{j \in S_i} O_j \geq |S_i|$$

*Proof.* Suppose in this case, there are  $n$  nodes at level  $i$ , and the number of nodes with increased values in the current cycle is  $m$ . Then we have

$$\begin{aligned} \sum_{j \in S_i} O_j &= \sum_{j=1}^m O_j + \sum_{j=m+1}^n O_j \\ &\geq m + \sum_{j=m+1}^n O_j \end{aligned} \quad (5)$$

$$\begin{aligned} &\geq m + n - m \\ &= |S_i| \end{aligned} \quad (6)$$

Equality in (5) holds when those  $m$  nodes have no children in the previous cycle, and equality in (6) holds when the remaining  $n - m$  nodes lose all their children in the current cycle. ■

Based on this, we have the following theorem.

**Theorem 2.** *Under the perturbation described above, for a spanning tree with  $n$  nodes, value of the source node satisfies*

$$O_s \geq \begin{cases} n & n \leq 2 \\ 3 & n \geq 3 \end{cases}$$

*Proof.* Cases  $n = 1$  and  $n = 2$  are trivial, since there is no parent switching. When  $n \geq 3$ , we have

$$O_s \geq \sum_{j \in S_1} O_j + 1 \quad (7)$$

If there is only one node at level 1, then  $O_s \geq 3$  since there must be at least one node at level 2. According to Lemma 2, if there are more than one node at level 1, then under the perturbation,

$$O_s \geq |S_1| + 1 \quad (8)$$

$$\geq 3 \quad (9)$$

Equality in (8) holds when nodes in level 1 satisfy the conditions mentioned in Lemma 2, and equality in (9) holds when there are two nodes at level 1. ■

### C. Effect of Distance Estimation on Error

Our analyses of the maximum and minimum bounds on error assume that the set of parent/child relations does, in fact, form a spanning tree at each point in time. This is not necessarily the case: when changes occur in the network, it can result in changes in the potential function that may take multiple rounds to resolve. Notably, as analyzed in [17], overestimates of potential resolve quickly, while underestimates may linger for many rounds. These persistent underestimates also imply a local minimum that breaks the spanning tree and creates a “black hole” for all information from devices with underestimated potential value, thereby preventing it from reaching the root.

We may also note that even when the spanning tree assumption holds, the origin of both duplication and loss of information is the transients that can occur when the potential function causes devices to change parents and thus the path by which paths to the root. Both of these indicate that the performance of C is most critically dependent not on its own behavior but on the adaptation dynamics of the algorithm used to compute the potential and maintain the spanning tree.

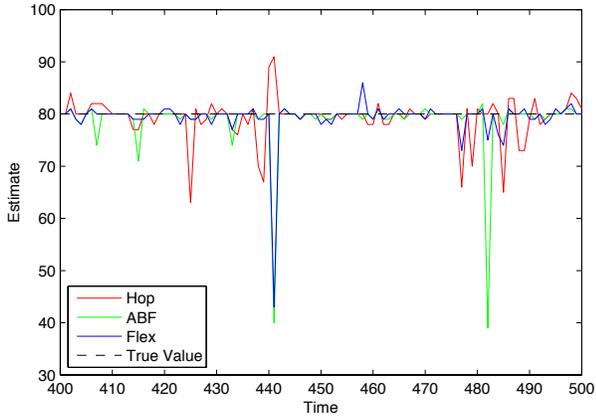
## IV. EMPIRICAL COMPARISON

To test these predictions, we ran experiments using using MIT Proto [19] to simulate a network of unsynchronized devices. In particular, we ran tests on devices randomly arranged in a rectangular region with neighbors determined by a unit disc graph. In each test, we estimated the number of devices in the network using three instances of

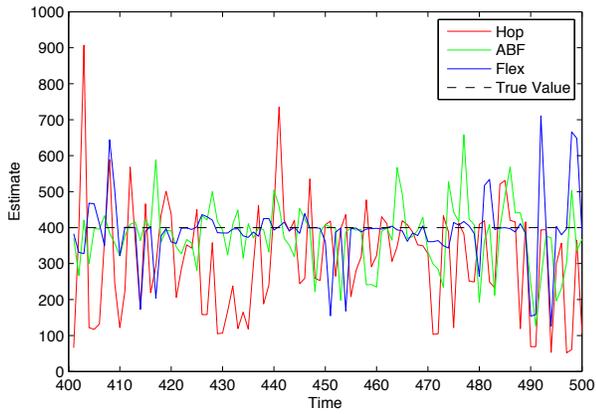
```
def countDevices(potential) {
  C(potential, sum, 1, 0)
}
```

in parallel, calculating `potential` from a single fixed source devices in a different manner for each instance: one using hop-count (i.e., Adaptive Bellman-Ford with unit distance), one by straight Adaptive Bellman-Ford [17], and one by Flex-Gradient [20]—the last of which prioritizes smoothness of slope over correctness of estimate. Error in state estimation is then the difference between the number of devices in the network and the state estimate output from C at the source device.

To test the effects of scale, we ranged the width of the arena from 2 to 20 units in steps of 2 units, while keeping its other dimension at 2 units, and placed the source device initially at the one extreme of the long axis of the arena. The number of devices was scaled proportional to the area of the arena, at 10 devices per square unit, i.e., from 40 devices in a  $2 \times 2$  arena to 400 devices in a  $20 \times 2$  arena. To inject continual perturbation into this network, each device moved randomly following a reactive Levy walk (a scale-free form of constrained random walk [21]) at a rate of 0.0025 units/second. Each trial was run for 1000 simulated seconds, 10 trials per condition, recording values at each second. Our analysis, however, drops the first 100 seconds of each trial as being potentially still affected by convergence from initialization.



(a) Example estimate variation over time in a 4 unit width space

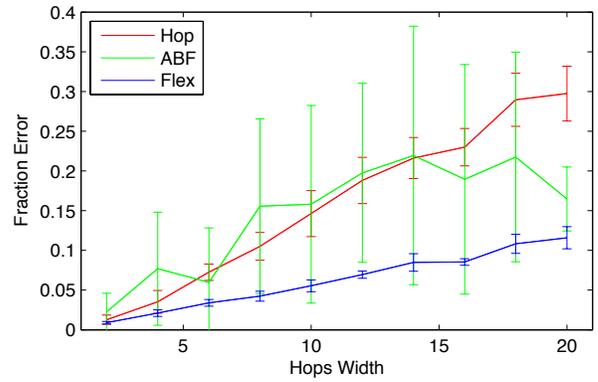


(b) Example estimate variation over time in a 20 unit width space

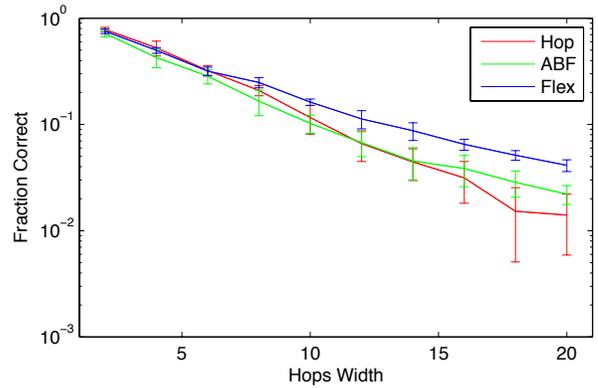
Fig. 3. Examples of typical estimate variation over time, showing excerpts from a individual simulation run on a network dispersed through a space (a) 4 units in width and (b) 20 units in width. Notice that most transients are relatively short, but they are much more frequent for hop-count than for Adaptive Bellman-Ford or Flex-Gradient, and they tend to be under-estimates rather than over-estimates, particularly for hop-count.

When diameter is low, it is expected that there should not be many opportunities for disruption on any given chain of parents to the source, and indeed we see that the estimates are often correct, but with frequent transients, sometimes causing large transients in estimate value. At high diameter, most devices are far from the source, so there are many opportunities for information to be duplicated or lost. Moreover, many of the devices share at least part of their path to the source, creating critical links whose disruption is likely to cause large transients. Figure 3 shows excerpts from two of trials, illustrating that the typical patterns in how estimates were observed to vary from the true value follow these predictions. Most individual transients are relatively short, but they are much more frequent for hop-count than for Adaptive Bellman-Ford or Flex-Gradient, and they tend to be under-estimates rather than over-estimates, particularly for hop-count.

Analysis of the overall statistics of errors versus width bears out these observations. Figure 4(a) shows that as the width of the space increases (and thus the diameter of the network



(a) Error vs. Width



(b) Correctness vs. Width

Fig. 4. The more hops spanned by the network, the higher the relative error in estimates (a) and the smaller amount of time that the estimate is correct. Furthermore, the smoother the input potential, the better estimation performs.

risers), the mean relative error in estimates rises approximately linearly—though the high degree of variation in the behavior of Adaptive Bellman-Ford makes difficult to verify for that case. Complementarily, Figure 4(b) shows that the amount of time that the estimate spends equal to the true value decreases, approximately exponentially with increasing width. This is as would be expected if we consider reconfiguration (and thus transient duplication or loss) to be equally likely to occur at any location in the network. In general, having a smoother input potential produces better results: Adaptive Bellman-Ford slightly outperforms hop-count distance values, and Flex-Gradient produces much better performance than both.

A deeper inspection of the errors finds that the distribution of individual error values is also consistent with the prediction of the importance of smoothness of potential from our analysis. Figure 5 shows a typical histogram of error ratios, in this case from the collection of trials with a width of 16 hops. Running C with all three potential algorithms results in a clear “spike” with a plurality of values being equal (or almost equal) to the true value, and all three have nearly identical rates of overestimates from transient duplication of values. The three potential algorithms differ starkly, however, in the distribution of underestimates. Hop-count distances are the least able to distinguish between alternative paths, and appear to pay

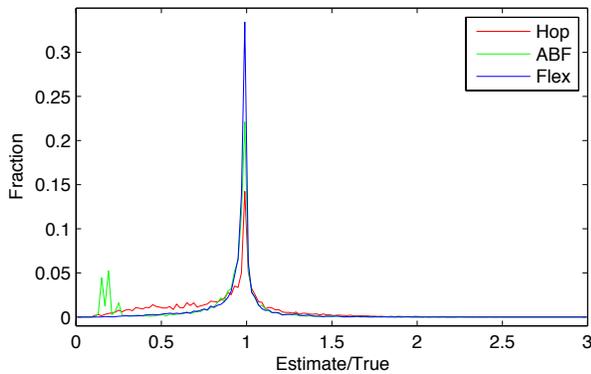


Fig. 5. Typical histogram of error values, showing that smooth adaptation of potential prevents transient value loss.

for this in generally increased volatility. Adaptive Bellman-Ford usually performs better, but in certain circumstances can experience long underestimate transients due to the “rising value problem” [17], [22]: when this problem occurs, it can cause a severe underestimate to last for a long time, as seen in the distribution spikes low values. Flex-Gradient, on the other hand, because it preserves smoothness at the cost of accuracy in distance estimates, suffers from much less loss of value than both of the others, explaining its superior performance.

## V. CONTRIBUTIONS

In this paper, we have presented a first analysis of the error dynamics of self-stabilizing approaches to collective state estimation via spanning-tree aggregation. Although we find the theoretical potential for exponential overestimates based on duplication of data, we also find that underestimates based on data loss are more likely to occur and to persist over long periods, particularly when the dynamics of using distance estimates to maintain a spanning tree are taken into account. These findings are validated in simulation, along with the prediction that prioritizing smoothness of slope in distance estimation should decrease error in collective state estimates.

In future work, we plan to extend these results to more classes of networks, to other classes of perturbation, such as link failures or devices joining and leaving, and to a broader class of estimations. The results that we have obtained also suggest possible approaches to decreasing the effect of network perturbations on state estimates, such as by smoothing to decrease the effect of transients or changes to the computation of the spanning tree to increase its resilience. Finally, these results suggest existing applications of collective state estimation can be improved by smoothness-prioritizing distance estimates like Flex-Gradient.

## ACKNOWLEDGMENT

This work has been supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0049. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department

of Defense or the U.S. Government. This document does not contain technology or technical data controlled under either U.S. International Traffic in Arms Regulation or U.S. Export Administration Regulations. Approved for public release, distribution unlimited.

## REFERENCES

- [1] Jacob Beal and Mirko Viroli, “Building blocks for aggregate programming of self-organising applications,” in *IEEE ESelf-Adaptive and Self-Organizing Systems Workshops*, 2014, pp. 8–13.
- [2] Jose Luis Fernandez-Marquez, Giovanna Di Marzo Serugendo, Sara Montagna, Mirko Viroli, and Josep Lluís Arcos, “Description and composition of bio-inspired design patterns: a complete overview,” *Natural Computing*, vol. 12, no. 1, pp. 43–67, 2013.
- [3] Ken Birman, “The promise, and limitations, of gossip protocols,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 8–13, 2007.
- [4] Devavrat Shah, *Gossip algorithms*, Now Publishers Inc, 2009.
- [5] Danilo Pianini, Jacob Beal, and Mirko Viroli, “Improving gossip dynamics through overlapping replicates,” in *Coordination Models and Languages*, 2016, pp. 192–207.
- [6] Michael J Fischer, Nancy A Lynch, and Michael S Paterson, “Impossibility of distributed consensus with one faulty process,” *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.
- [7] Nancy Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco, USA, 1996.
- [8] Seth Gilbert and Nancy Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” *Sigact News*, vol. 33, no. 2, 2002.
- [9] Whyay Chiou Lee, “Spanning tree method for link state aggregation in large communication networks,” in *INFOCOM’95*, 1995, pp. 297–302.
- [10] Hüseyin Özgür Tan and Ibrahim Körpeoglu, “Power efficient data gathering and aggregation in wireless sensor networks,” *ACM Sigmod Record*, vol. 32, no. 4, pp. 66–71, 2003.
- [11] Marc Lee and Vincent WS Wong, “An energy-aware spanning tree algorithm for data aggregation in wireless sensor networks,” in *Communications, Computers and signal Processing, 2005. PACRIM. 2005 IEEE Pacific Rim Conference on*. IEEE, 2005, pp. 300–303.
- [12] Lehel Nyers and Márk Jelasity, “A comparative study of spanning tree and gossip protocols for aggregation,” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4091–4106, 2015.
- [13] Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai, “On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: Np-completeness and approximation algorithms,” *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 3109–3121, 2016.
- [14] Mirko Viroli, Jacob Beal, Ferruccio Damiani, and Danilo Pianini, “Efficient engineering of complex self-organising systems by self-stabilising fields,” in *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on*. IEEE, 2015, pp. 81–90.
- [15] Shlomi Dolev, *Self-Stabilization*, MIT Press, 2000.
- [16] Jacob Beal, Danilo Pianini, and Mirko Viroli, “Aggregate programming for the internet of things,” *IEEE Computer*, vol. 48, no. 9, pp. 22–30, 2015.
- [17] Soura Dasgupta and Jacob Beal, “A Lyapunov analysis for the robust stability of an adaptive Bellman-Ford algorithm,” in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 7282–7287.
- [18] Danilo Pianini, Mirko Viroli, and Jacob Beal, “Protelis: practical aggregate programming,” in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1846–1853.
- [19] Jacob Beal and Jonathan Bachrach, “Infrastructure for engineered emergence on sensor/actuator networks,” *IEEE Intelligent Systems*, vol. 21, no. 2, pp. 10–19, 2006.
- [20] Jacob Beal, “Flexible self-healing gradients,” in *Proceedings of the 2009 ACM Symposium on Applied Computing*, New York, NY, USA, 2009, SAC ’09, pp. 1197–1201, ACM.
- [21] Jacob Beal, “Superdiffusive dispersion and mixing of swarms with reactive levy walks,” in *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2013)*. IEEE, September 2013, pp. 141–148.
- [22] Jacob Beal, Jonathan Bachrach, Dan Vickery, and Mark Tobenkin, “Fast self-healing gradients,” in *ACM Symposium on Applied Computing*, New York, NY, USA, March 2008, ACM.