Self-Stabilized Fast Gossiping Algorithms

STEFAN DULMAN and ERIC PAUWELS, CWI

In this article, we explore the topic of extending aggregate computation in distributed networks with self-stabilizing properties to withstand network dynamics. Existing research suggests that fast gossiping algorithms, based on the properties of order statistics applied to families of exponential random variables, are a viable solution for computing functions of the values stored in the network. We focus on the specific case in which network changes and failures occur in batches with a minimum frequency in the order of the diameter of the network. Our contribution consists in two self-stabilizing mechanisms, allowing fast gossiping algorithms to be applicable to dynamic networks with minor increase in resources usage. The resulting algorithms can be deployed in networks exhibiting churn, node stop-failures and resets, and random topological changes. The theoretical results are verified with simulations on synthetic data, showcasing desirable properties for large-scale network designers such as scalability, lack of single points of failure, and anonymity.

Categories and Subject Descriptors: C.2.1 [**Network Architecture and Design**]: Distributed Networks

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Self-stabilization, distributed network, gossiping algorithm

ACM Reference Format:
Stefan Dulman and Eric Pauwels. 2015. Self-stabilized fast gossiping algorithms. ACM Trans. Auton. Adapt. Syst. 10, 4, Article 29 (December 2015), 20 pages.
DOI: http://dx.doi.org/10.1145/2816819

1. INTRODUCTION

Advances in electronics, telecommunication, and user interface design have led to recent deployments of myriads of networked embedded platforms around us. Smartphones, wireless sensor networks, swarms of robotic devices (drones, intelligent cars, etc.) are just a few examples of systems being increasingly found in our environment. Making these multinode systems "intelligent" and able to autonomously adapt to internal and external changes is of acute interest with immediate societal impact. The bottleneck in such designs is usually their distributed control: due to their sheer size, networks are subject to emergent behavior that more often than not disrupts their functionality. Examples abound in smart energy grids, large-scale infrastructure, internet of things, and smart cities applications.

Traditional centralized control fails to work beyond a certain network size, reasons including limited bandwidth and real-time constraints violated mainly by the continuous changes in network topology. However, in many cases an adequate control strategy relies on some form of measurement of global parameters of the network—in other words, estimating if the network is doing the "right thing"—and adopting corrective

This work was partly funded by the Rijksdienst voor Ondernemend Nederland grant TKISG01002 SG-BEMS.
Authors' addresses: S. Dulman and E. Pauwels, Science Park 123, 1098 XG Amsterdam, The Netherlands; emails: {stefan.dulman, eric.pauwels}@cwi.nl.

DOI: http://dx.doi.org/10.1145/2816819

measures if not. Our work focuses on employing gossip algorithms [Boyd et al. 2005] to achieve distributed state estimation for large-scale networks, thus providing the basic measurement building block for distributed control.

As the name suggests, gossip algorithms attempt to compute aggregate (i.e., global) values for important network parameters by relying exclusively on local exchanges of information. Put differently, network nodes only communicate with their neighbors, but nevertheless manage to efficiently compute a reliable value for global network parameters (e.g., the network-wide average, maximum value, various statistic parameters, etc.). Such an approach obviates the need to establish a central control authority: since the resulting estimates diffuse across the network, every node will harbor the appropriate value. Furthermore, our work extends these algorithms to be self-stabilizing in the sense that changes in the network topology (e.g., resulting from rearrangements of nodes) are rapidly and automatically reflected in the final result. A vast body of literature exists around the concept of epidemic algorithms (such as gossiping [Boyd et al. 2005])—unfortunately, probably due to the lack of centralized control, their adoption in practice is very limited at best (peer-to-peer applications being notable exceptions).

"Traditional gossiping" [Boyd et al. 2005] is a slow converging protocol. We focus on recent developments, such as the work of Shah [2009], which proposes trade-offs leading to very fast converging algorithms. The authors exploit a property of order statistics applied to exponential random variables to achieve convergence in $O(D \log N)$ timesteps instead of the "traditional" $O(D^2 \log N)$ timesteps (D being the diameter of the network and N the number of nodes). In the following, we refer to this former class of epidemic algorithms as *fast gossiping algorithms*.

Dynamics (i.e., changes in network topology such as nodes leaving the network, stop-failing, resetting, etc.) heavily affect the results of gossiping algorithms, leading to solutions employing rounds of communication that need to be synchronized [Jelasity et al. 2005]. The reasoning for this is that during short intervals of time the networks will experience very few modifications, so the shorter the rounds, the better the estimates. Drawing on our previous work [Pruteanu and Dulman 2012], we make use of synchronization-free approaches, in which simple mechanisms are employed to add self-stabilization properties to gossiping algorithms. In this article, we show how making use of simple timer mechanisms allows "removal" of old values from a network and triggers the computed aggregate to change reflecting changes in the network-stored values. Our solution is limited to the scenarios in which changes occur in batches, with a period larger than the time needed for the network to stabilize.

As an example, let us consider the situation of a large crowd of people taking part in a city-wide public event. People will carry smartphones and, most usual than not, the network infrastructure will be assumed unavailable due to the large traffic incurred. The number of people in the demonstration is a measure of interest to several parties, including emergency services that accompany such an event. We achieve this computation by using the ad hoc communication capabilities of smartphones (leading to a multihop mesh network) and by employing a simple aggregate function (i.e., the sum of the values in the network) of the values in the network (i.e., each smartphone holds the value 1). In this scenario, dynamics are represented by a multitude of events: people randomly joining and leaving the event, continuously changing network topology, obvious communication failures, etc. Self-stabilizing fast gossiping algorithms provide an elegant solution in this case: not only that the value of interest is computed fast and reliable on all nodes despite the dynamics, but the convergence time depends mainly on the diameter of the network and not on the number of participants. In other words, the size of the area in which the event takes place influences the convergence speed. As we show in Section 4 convergence speed remains almost constant even when varying the number of nodes across several orders of magnitude.

Various decentralized approaches solutions exist in literature—they tend to rely on assumptions invalidated by the previous example case. A few examples include network-wide synchronization [Jelasity et al. 2005], averaging of estimates of network parameters running in parallel [Bicocchi et al. 2010], or simply tracking nodes joining or leaving the network. In line with our previous work [Pruteanu and Dulman 2012; Iyer et al. 2011], we are advocating for solutions that do not require tracking of individual nodes and are built in the basic mechanism rather than added as layers of complexity above it. For example, we introduce in Section 3.1 a simple timer mechanism that allows "removal" of values belonging to nodes that already left the network, without the need for a tracking mechanism or even for unique identifiers for the nodes. Furthermore, in Section 3.2 we show how to achieve a significant speed improvement by the alteration of the timer expiry function, leading from a linear to an exponential behavior.

The article is structured as follows. Section 2 introduces fast gossiping algorithms and provides references to the relevant literature. Section 3 presents the main contribution of the article, two self-stabilization extensions, and theoretical characterization of their convergence time. The algorithms are evaluated numerically and analyzed in Section 4. We provide information on related work on aggregate computation in Section 5 and conclude the article with Section 6.

2. PRELIMINARIES—FAST GOSSIPING ALGORITHMS

For the scope of this article, we target geometric random graphs (i.e., mesh networks), where nodes can communicate mainly with their direct neighbors. From the perspective of the communication model, we assume that time is discrete. During one *timestep* each node will pick and communicate with a random neighbor. Major changes, disruptions, and updates in the network occur just once in a large number of timesteps. We will make use of the concept of synchronized *time rounds* and ask the nodes to update their local data at the beginning of the rounds. The bootstrap problem and round-based time models received a lot of attention in literature [Jelasity et al. 2005; Bicocchi et al. 2010; Pruteanu and Dulman 2012] and is not the focus in this article—loose constraints allow for algorithms like the one presented in Werner-Allen et al. [2005].

We make no assumptions with respect to nodes stop-failing or new nodes joining the network. The mechanism described in Section 3 can accommodate these cases and the computation results will adapt themselves to such changes.

Our contributions are an extension to the basic primitive for computing sums in a distributed network via fast gossiping mechanism presented in Mosk-Aoyama and Shah [2008]. The algorithm presented in Mosk-Aoyama and Shah [2008] uses a property of order statistics applied to a series of N exponential random variables with parameters λ_i, $i \in (1, N)$, which leads to the sum of the parameters λ_i (Table I). The algorithm resembles gossiping algorithms [Jelasity et al. 2005] but differs in a number of important points.

Essentially, it trades communication for convergence speed. By relying on the propagation of an extreme value (the minimum value in this case), locally computable, it achieves the fastest possible convergence in a distributed network—$O(D \log N)$ timesteps (with D being the diameter of the network). This speed is significant compared to the original gossiping algorithms that converged in $O(D^2 \log N)$ timesteps [Boyd et al. 2005].

The paid price is the increase in exchanged messages size of $O(\delta^{-2})$. δ is a parameter defining the precision of the final result. If λ is the ground-truth result, the algorithm offers an estimate in the interval $[(1-\delta)\lambda, (1+\delta)\lambda]$ with an error $\epsilon = O(1/polynomial(N))$.

Zooming into the details of the algorithm (see Algorithm 1), each node i holds a *positive* variable λ_i from which it generates an exponential random variable vector $\mathbf{v}$. At each timestep, each node chooses a random neighbor and they exchange their

Table I. Notations Used in the Article

Symbol	Meaning
D	Diameter of the network
N	Number of nodes in the network
T	Maximum value for the time-to-live field
M	Maximum number of samples in a vector
$\mathcal{C}$	Decay constant for time-to-live (by default 0.5)
δ	Precision of the estimation
ϵ	Probability that the estimation falls within a certain range
i, i_1, i_2	Indexes for the nodes in the network
j	Index for the samples in a vector of size M
λ_i	Local variable for node i
$\mathbf{v}_i$	Vector of exponentially distributed random values on node i
$\mathbf{v}_i^0$	Original vector of random values on node i
$\mathbf{u}$	Vector containing the minimum values from all $\mathbf{v}_i$
$\boldsymbol{\tau}_i$	Vector of time-to-live values on node i
n_+	Number of nodes holding an old value
n_-	Number of nodes holding a negative value
n_0	Number of nodes holding a new value
ψ	Average time-to-live value on negative value nodes
k	Integer index, $k \geq 1$
$a_{k_1}, a_{k_2}, \ldots$	Real coefficients
σ_1	Initial sum of values in the network
σ_2	Intermediate sum of values in the network
σ_3	Final sum of values in the network

ALGORITHM 1: Mosk-Aoyama-Shah Algorithm

```
/* λ_i - local parameter for node i */
/* m - number of samples in the random vectors */
/* n - number of nodes in the network */
/* S - set of received value vectors in the last timestep */
/* v_local - local value vector */
/* v^0 - original value vector */
/* initialization - run once */
if v^0 is uninitialized then
    for j = 1 to m do
        v^0[j] = random number from exponential distribution with parameter λ_i
    v_local = v^0
/* periodic update - run every timestep */
for each vector v in S do
    for j = 1 to m do
        if v_local[j] > v[j] then
            v_local[j] = v[j]
    Broadcast v_local
/* estimation of sum of λ-s */
∑_{i=1}^{n} λ_i ≈ m / ∑_{j=1}^{m} v[j]
```

values, both keeping the smallest value on each position. In other words, after two nodes i_1 and i_2, holding the values $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_2}$ communicate, each of them will hold a vector $\mathbf{v} = \mathbf{v}'_{i_1} = \mathbf{v}'_{i_2}$ with the property that $\mathbf{v}[j] = \min(\mathbf{v}_{i_1}[j], \mathbf{v}_{i_2}[j])\ \forall j \in (1, M)$. Thus, the smallest value on each position in the vectors propagates fast in the network,

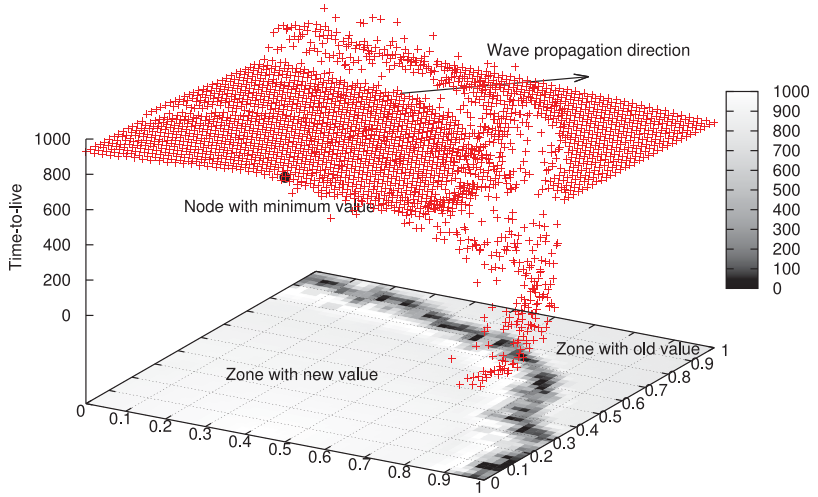

Fig. 1. Value removal mechanism wave-alike propagation (geometric random graph with 100,000 nodes; diameter 40; random values).

in $O(D \log N)$ timesteps via this push-pull gossiping mechanism (see Shah [2009], Section 3.2.2.4, p. 32).

The authors of Shah [2009] further show that, after all vectors in the network converge to the minimum-holding vector $\mathbf{u}$, the sum of λ_i values in the network is approximated by the maximum likelihood estimator: $\sum_{i=1}^{N} \lambda_i \approx \frac{m}{\sum_{j=1}^{m} \mathbf{u}[j]}$ (see Shah [2009], Property 5.1, p. 72).

Furthermore, the precision of the result is independent of the network size. The message size is dependent only on the precision δ, $O(\delta^{-2})$ (see Shah [2009], Theorem 5.1, p. 74 and Section 5.2.5.4, p. 75).

3. SELF-STABILIZATION EXTENSIONS

In this section, we extend the minimum value propagation mechanisms presented in Section 2 to account for dynamics in the network. Specifically, we add a *time-to-live field* to each value—an integer value that decreases with time and marks the age of the current value. This mechanism takes care of nodes leaving the network, stop-crashing, or resetting without the need of tracking each node. The time-to-live mechanism works also in the failure case in which a group of nodes changes their values at a given moment. Furthermore, we extend the time-to-live expiry mechanism to achieve a $O(D \log N + \log T)$ timesteps *active value removal*. In other words, if a certain minimum value belonging to a node that changed its variable λ_i propagated through the network, we mark it as "expired" and assure its associated time-to-live value to expire (reach 0) within $O(D \log N + \log T)$ timesteps. Intuitively, in a multihop mesh network, the value removal mechanism takes the shape of a wave that replaces the expired value in the network (see Figure 1).

3.1. Passive Mechanism: Counter-Based Self-Stabilization

We extend the algorithm presented in Mosk-Aoyama and Shah [2008] by adding to each node i a new vector $\boldsymbol{\tau}_i$ holding a time-to-live counter for each value. This new vector is initialized with a default value T, larger than the convergence time of the original algorithm (choosing a proper value is explained next).

The values in $\boldsymbol{\tau}_i$ decrease by 1 every time slot, with one exception. The node generating the minimum $\mathbf{v}_i[j]$ on the position $j \in (1, M)$ sets $\boldsymbol{\tau}_i[j]$ to T (see Algorithm 3,

ALGORITHM 2: PropagateMinVal(v, τ)

```
1  /* v, τ - received value and time-to-live */
2  /* v_local, τ_local - local value and time-to-live */
3  /* create temporary variables */
4  (v_m, v_M) ← (min(v, v_local), max(v, v_local))
5  (τ_m, τ_M) ← corresponding (τ, τ_local) to (v_m, v_M)
6  /* update logic */
7  if v_m == v_M then
8  |  if v_m < 0 then                                   /* equal negative values */
9  |  |  τ_m ← C τ_m
10 |  else                                              /* equal positive values */
11 |  |  min(τ_m, τ_M) ← max(τ_m, τ_M) − 1
12 else
13 |  if v_m < 0 then                                   /* at least one negative value */
14 |  |  if v_m == −v_M then
15 |  |  |  (τ_m, τ_M) ← (T, T)
16 |  |  else
17 |  |  |  (τ_m, τ_M) ← (C τ_m, C τ_M)
18 |  else                                              /* two different positive values */
19 |  |  τ_M ← τ_m − 1
20 /* update local variables */
21 (v, v_local) ← (v_m, v_m)
22 (τ, τ_local) ← corresponding (τ_m, τ_M)
```

line 9). In the absence of any other dynamics, all properties proved in Shah [2009] remain unchanged as the output of our approach is identical to the original algorithm.

The main reason for adding the time-to-live field is to account for nodes leaving the network or nodes that fail-stop. In this way we avoid complicated mechanisms in which nodes need to keep track of neighbors. An interesting side consequence is that this mechanism does not require node identifiers, thus applications built on top preserve the privacy of the nodes in the network.

The intuition behind the counter-based mechanism is that a node i_0 generating the network-wide minimum on position $j \in (1, M)$ will always advertise it with the accompanying time-to-live set to the maximum T. Any other node i will adopt the min value $\mathbf{v}_{i_0}[j]$ as $\mathbf{v}_i[j]$ and have a value $\boldsymbol{\tau}_i[j]$ decreasing with the distance from the minimum setting node i_0. T is chosen to be larger than the maximum number of gossiping steps it takes the minimum to reach any node in the network.

In a gossiping step between two nodes i_1 and i_2, if $\mathbf{v}_{i_1}[j] = \mathbf{v}_{i_2}[j]$, then the largest of the $\boldsymbol{\tau}_{i_1}[j]$ and $\boldsymbol{\tau}_{i_2}[j]$ will propagate (Algorithm 2, line 11). This means that $\boldsymbol{\tau}_i[j]$ on all nodes i will be strictly positive for as long as the node is online. If the node that generated the minimum value on the position j goes offline, all the associated $\boldsymbol{\tau}_i[j]$ values in the network will steadily decrease (Algorithm 3, line 11) until they will reach 0 and the minimum will be replaced by next smallest value in the network (Algorithm 3, lines 12–14). Hence, it takes T timesteps for the network to "forget" the value on position j.

3.2. Active Mechanism: Value Removal Algorithm

The second self-stabilizing mechanism targets nodes changing their values at runtime but remain in the network and are therefore able to actively remove old versions of their values that might have propagated in the network. Assume a node i changes its value

Table II. Value Propagation

Propagation	Ordering	Previous	Intermediate	Final
none	$\mathbf{u}[k] < \mathbf{v}_i[k] < \mathbf{v}'_i[k]$	$\mathbf{u}[k]$	$\mathbf{u}[k]$	$\mathbf{u}[k]$
	$\mathbf{u}[k] < \mathbf{v}'_i[k] < \mathbf{v}_i[k]$	$\mathbf{u}[k]$	$\mathbf{u}[k]$	$\mathbf{u}[k]$
slow	$\mathbf{v}_i[k] < \mathbf{u}[k] < \mathbf{v}'_i[k]$	$\mathbf{v}_i[k]$	$\mathbf{v}_i[k]$	$\mathbf{u}[k]$
	$\mathbf{v}_i[k] < \mathbf{v}'_i[k] < \mathbf{u}[k]$	$\mathbf{v}_i[k]$	$\mathbf{v}_i[k]$	$\mathbf{v}'_i[k]$
fast	$\mathbf{v}'_i[k] < \mathbf{u}[k] < \mathbf{v}_i[k]$	$\mathbf{u}[k]$	$\mathbf{v}'_i[k]$	$\mathbf{v}'_i[k]$
	$\mathbf{v}'_i[k] < \mathbf{v}_i[k] < \mathbf{u}[k]$	$\mathbf{v}_i[k]$	$\mathbf{v}'_i[k]$	$\mathbf{v}'_i[k]$

ALGORITHM 3: ComputeSum ($\mathbf{v}$, $\boldsymbol{\tau}$)

```
1  /* v^0 - original random samples vector on this node */
2  /* v, τ - received value and time-to-live vectors */
3  /* update all elements in the data vector */
4  for j = 1 to length(v) do
5  |  PropagateMinVal(v[j], τ[j])
6  /* time-to-live update - do once every timeslot */
7  for j = 1 to length(v) do
8  |  if v[j] == v^0[j] then                          /* reinforce a minimum */
9  |  |  τ[j] ← T
10 |  else
11 |  |  τ[j] ← τ[j] − 1                             /* decrease time-to-live */
12 |  |  if τ[j] <= 0 then                            /* value expired */
13 |  |  |  v[j] ← v^0[j]
14 |  |  |  τ[j] ← T
15 /* estimate the sum of elements */
16 s ← 0
17 for j = 1 to length(v) do
18 |  s ← s + abs(v[j])
19 return length(v)/s
```

λ_i to λ'_i at some time t. This change will trigger a regeneration of its original samples from the exponential random variable $\mathbf{v}_i$ to $\mathbf{v}'_i$. Let j be an index with $j \in (1, M)$. Let $\mathbf{u}$ be the vector containing the minimum values in the network if the node i *would not exist*. In order to understand the change happening when transitioning from λ_i to λ'_i we need to look at the relationship between the individual values $\mathbf{v}_i[j]$, $\mathbf{v}'_i[j]$, and $\mathbf{u}[j]$.

As shown in Table II, if $\mathbf{u}[j]$ is the smallest of all three values, then no change will propagate in the network. If $\mathbf{v}'_i[j]$ is the smallest value, then this will propagate fast, in $O(D \log N)$ timesteps, via the basic minimum propagation mechanism. If $\mathbf{v}_i[j]$ is the smallest, then this value will remain in the network until its associated time-to-live field expires in $O(T)$. As usually $T \gg D$, this leads to a very slow process. We designed the value removal mechanism to speed up the expiration of this value from the network.

The removal mechanism is triggered by the node owning the value that needs to be removed (in our case node i) and works as follows: node i will mark the value $\mathbf{v}_i[j]$ as "expired" by propagating a negative value $-\mathbf{v}_i[j]$. This change will not affect the minimum value propagation mechanism (Algorithm 2, lines 4, 21—the negative value of a positive minimum remains the minimum in the network) or the estimation of the sum (notice the use of the absolute value function in Algorithm 3, line 18). If node i contacts a node also holding the value $\mathbf{v}_i[j]$, then first, it will propagate the negative sign for the value, also maximizing its time-to-live field to a large value T. Intuitively, as long as the $\mathbf{v}_i[j]$ is present in the network, the $-\mathbf{v}_i[j]$ will propagate, overwriting it.

Considering the large range of unique float or double numbers versus the number of values in a network at a given time, we assume the values in the network to be unique.

The time-to-live field of any negative value will halve with each gossiping step (by default $C = 0.5$) if it does not meet the $\mathbf{v}_i[j]$ value (Algorithm 2, lines 9, 17). Intuitively, if a negative value is surrounded by values other than $\mathbf{v}_i[j]$, it will overwrite the neighbors' positive values while canceling itself at the same time with an exponential rate. This mechanism somewhat resembles a predator-prey model [Arditi and Ginzburg 1989], where prey is represented by the $\mathbf{v}_i[j]$ variable and predators by $-\mathbf{v}_i[j]$. We designed it such that the populations cancel each-other, targeting the fixed point at the origin as the solution for the accompanying Lotka-Volterra equations—details are offered in Section 3.3.

LEMMA 3.1 (VALUE REMOVAL DELAY). *By using the value removal algorithm, the new minimum propagates in the network in* $O(D \log N + \log T)$ *timesteps.*

PROOF. In the worst case scenario, the whole network contains the minimum value $\mathbf{v}_i[k]$ on position k, with the time-to-live field set at the maximum T. The negative value, being the smallest one in the network, propagates in $O(D \log N)$ in the whole network. Again, in the worst case scenario, we have a network with each node having the value $-\mathbf{v}_i[k]$ on position k with the time-to-live set to the maximum T. From this moment on, the time-to-live will halve at each gossip step on each node, reaching 0, in the worst case scenario in $O(\log T)$ timesteps. This is the worst case because nodes may be contacted by several neighbors during a timestep leading to a much faster cancellation. Overall, the removal mechanism will be active for at most $O(D \log N + \log T)$ timesteps. This bound is an upper bound. In reality, the spread and cancellation mechanisms will act in parallel, leading to tighter bounds. □

This result gives us the basis for choosing the T constant. Ideally, T should be chosen as small as possible, in line with the diameter of the network. The fact that the removal mechanism is affected only by $\log T$ lets us use an overestimate of T, which even if a few orders of magnitude larger than the diameter of the network will have little impact on the convergence speed but will leave an effect on the passive expiry mechanism. For example, if the network diameter is between 10 and 30 and the values change roughly every 10,000 timesteps, we may safely set T anywhere between 100 and 1,000. This will not affect the convergence of the sum computation mechanism but allow for a timely account for a node removal (under the assumption that nodes leaving the network happens less frequently than nodes changing their values).

Put together, the mechanisms presented in this section lead to the sum computation mechanism *ComputeSum*() presented in Algorithm 3. It holds the properties of the original algorithm described in Mosk-Aoyama and Shah [2008] and it additionally showcases self-stabilization properties to account for network dynamics in the form of node removal and nodes changing their values in batches.

3.3. A Model for the Value Removal Mechanism

The value removal mechanism maps onto a predator-prey model, in which the predators (negative values $-\mathbf{v}_i[j]$) need to consume all the prey (positive values $\mathbf{v}_i[j]$) before going extinct. Graphically, the results are presented in Figure 2. In this section, we provide a mathematical model for this mechanism, with the goal of validating that it leads to the extinction of both predators and prey, meaning, in our case, that a new value propagates in the network. For the sake of clarity, we assume that the vector of values has a single element ($M = 1$), the extension to $M \neq 1$ being straightforward. We will drop the index when referring to vectors, such that $\mathbf{v}[1]$, $\mathbf{u}[1]$, and $\boldsymbol{\tau}[1]$ will be addressed as v, u, and τ. *The old value* refers to the minimum in the network (u), *the negative value* refers to $-u$, and *the new value* to a new minimum u'.

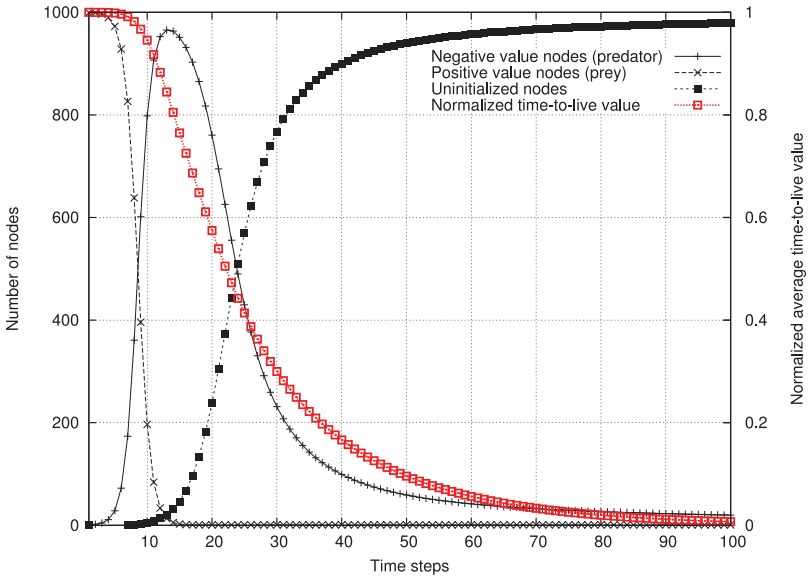

Fig. 2. Predator-prey simulation with 1,000 nodes ($\mathcal{C}_1 = \mathcal{C}_2 = 0.5$; initial configuration consists of a single predator node and 999 prey nodes).

We focus on a fully connected network case, as this constitutes the most difficult scenario. Intuitively, the value removal mechanism acts as a wave starting at a node and encompassing all the network. In the case of a fully connected network, nodes can talk to any of the neighbors, meaning also that the wave has a difficult time "catching" values diffused in this communication model.

Let n_+ be the number of nodes that hold the old value, n_- the number of nodes holding the negative value, and n_0 the number of nodes that do not hold either—*uninitialized nodes* ($n_+ + n_- + n_0 = N$). We propose a differential equations model to capture the variation of these quantities in the network. Targeting the worst case scenario, initially all nodes are assumed initialized with the old value except one, which is initialized with the negative value ($n_+ = N - 1, n_- = 1$).

Modeling the time-to-live mechanism is difficult due to its nonlinearity. We use the following approximation: define a variable ψ that holds *the average* time-to-live value on the n_- nodes. The ψ variable changes each time a node with a negative value interacts with another node.

We focus on a node i with a negative value $-v_i$, having the time-to-live value τ_i. If it interacts with a positive value node (its counterpart or any uninitialized node), then the tuple of time-to-live values on the two nodes becomes $(\tau_i \approx \psi, -) \to (T, T)$ and $\psi \to (\psi + \frac{2(T-\psi)}{n_-+1})$, thus increasing (the state transitions are shown in Figure 3). If the negative value node meets an uninitialized node, then the tuple of time-to-live values becomes $(\tau_i, -) \to (\mathcal{C}\tau_i, \mathcal{C}\tau_i)$ and $\psi \to (\psi + \frac{2(\mathcal{C}-1)\psi}{n_-+1})$, with $\mathcal{C} \in (0, 1)$, thus decreasing.

When several negative value nodes meet uninitialized nodes (say a fraction h), then ψ changes to ψ':

$$\psi = \frac{\sum_i \tau_i}{n_-},$$
$$\psi' = \frac{\sum_i \tau_i - hn_-\psi + 2hn_-C\psi}{n_- + hn_-} = \psi + 2(C-1)\psi\frac{h}{1+h}. \quad (1)$$

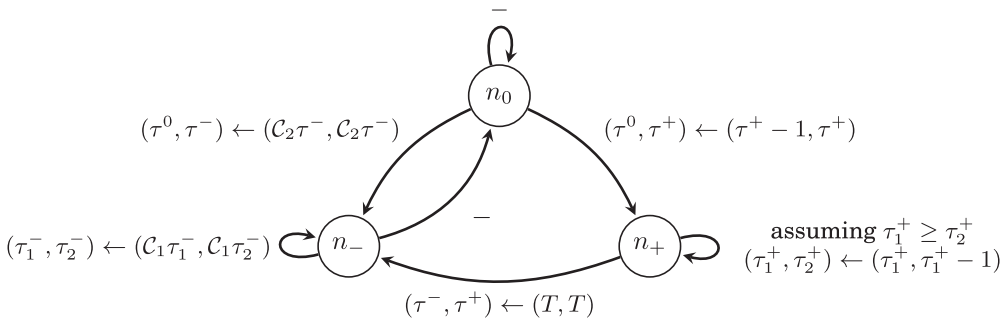

Fig. 3. State transition effects on time-to-live (indices $_1$, $_2$ distinguish between two nodes of the same type interacting; the arrows indicate transition between states—they can be interpreted also as interactions between nodes in the two states they connect; diagram does not account for the decrease of time-to-live with each timestep).

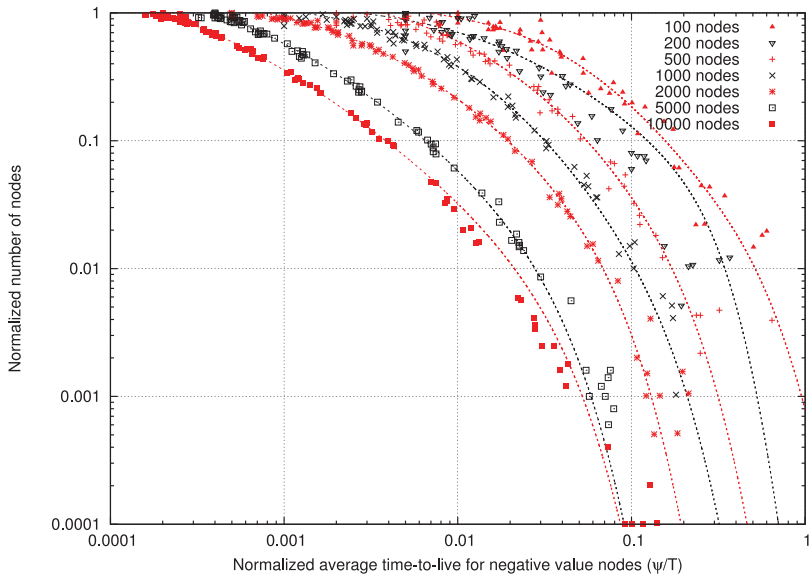

Fig. 4. Variation of the negative value nodes with respect to average time-to-live value (interpolated lines represent possible $f(\psi)$ fittings).

From a probabilistic perspective, each of the n_- nodes meets on average $\frac{n_0}{N}$ unitialized nodes, thus $h = \frac{n_0}{N}$. This leads to $\psi \rightarrow (\psi + 2(C-1)\psi\frac{n_0}{n_0+N})$. By replacing h with $\frac{n_-}{N}$ a similar formula is obtained for negative value nodes interacting with other negative value nodes). Following the same reasoning, we obtain $\psi \rightarrow (\psi + 2(T-\psi)\frac{n_+}{n_++N})$ for negative value nodes meeting positive value nodes.

Negative value nodes interacting with other negative value nodes decrease their time-to-live values fast and transform themselves in uninitialized nodes. We can approximate the whole process with a fraction $f(\psi)n_-$. Regarding the profile of the $f(\psi)$ function, few nodes become uninitialized when ψ is close to T. As ψ decreases approaching 0, negative-values nodes increasingly become uninitialized. Figure 4 shows this dependency, while Figure 5 shows the general variation of the number of nodes with the average value of the time-to-live field for negative value nodes.

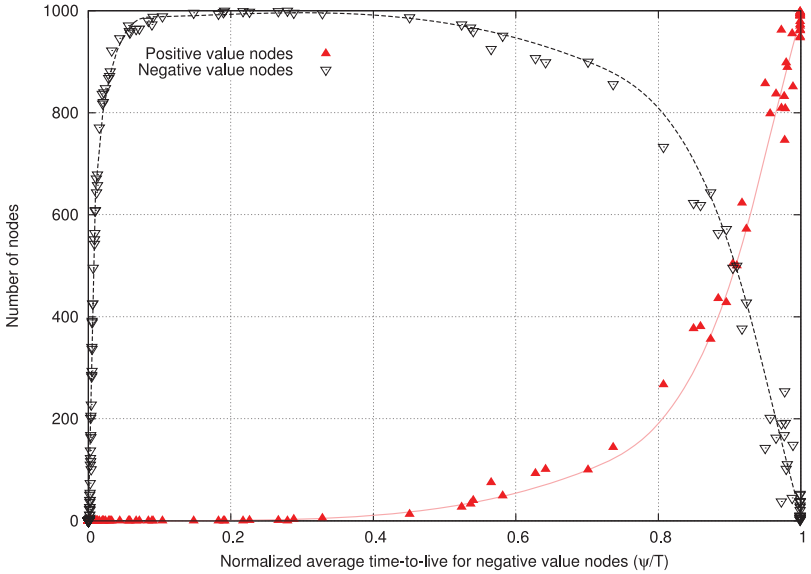

Fig. 5. Number of positive and negative value nodes with respect to average time-to-live value (geometric random graph with 1,000 nodes; diameter 10; points are taken from 10 different simulations ran till convergence).

Based on the behavior observed in practice, the function $f(\psi)$ can be approximated with an expression in the form

$$f(\psi) = \frac{1}{\sum_{k\geq 1} a_{k_1} \exp\left(a_{k_2}\left(\frac{\psi}{T}\right)^k\right) + \sum_{k\geq 1} a_{k_3}\left(\frac{\psi}{T}\right)^k} \tag{2}$$

with k being an index and a_{k_*} suitably chosen real number coefficients.

This leads to the system

$$\begin{aligned}
\frac{dn_+}{dt} &= \frac{n_0}{N}n_+ - \frac{n_-}{N}n_+,\\
\frac{dn_-}{dt} &= \frac{n_+}{N}n_- + \frac{n_-}{N}n_0 - f(\psi)n_-,\\
\frac{d\psi}{dt} &= 2(T-\psi)\frac{n_+}{n_+ + N} + 2\psi(\mathcal{C}_1 - 1)\frac{n_-}{n_- + N} + 2\psi(\mathcal{C}_2 - 1)\frac{n_0}{n_0 + N},
\end{aligned} \tag{3}$$

where $\mathcal{C}_1, \mathcal{C}_2 \in (0, 1)$ are constants. For simplicity, we will consider these two constants equal ($\mathcal{C}$). It is easy to check that the trivial solution ($n_+ = 0$, $n_- = 0$, $\psi = 0$) verifies the system. This model captures well the behavior of the value removal mechanism, but it still remains an approximation. Its main drawback is that the time scale tends to be larger than in simulations. The approximation comes from the use of the function $f(\psi)$ for which we do not have an exact expression and was obtained via fitting.

Nevertheless, when exploring the solution spaces as a function of $\mathcal{C}$ it turns out that the system converges to the trivial solution as long as $\mathcal{C}$ is above a certain value, dependent on the network size. This is shown graphically in Figure 6, in which we plotted the variation of the number of negative value nodes as a function of time, for various values of the constant $\mathcal{C}$. As seen also in practice, for a large range of $\mathcal{C}$, the system converges fast to the trivial solution. As $\mathcal{C}$ gets smaller, the convergence speeds up and after a certain threshold the system converges to a different solution,

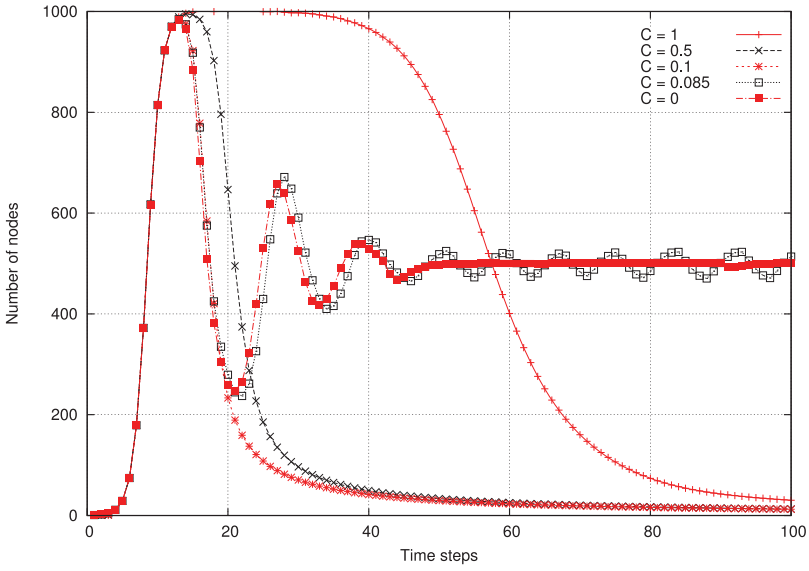

Fig. 6. Variation of the number of negative value nodes with time in a network with 1,000 nodes ($\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}$; initial configuration consists of a single predator node and 999 prey nodes).

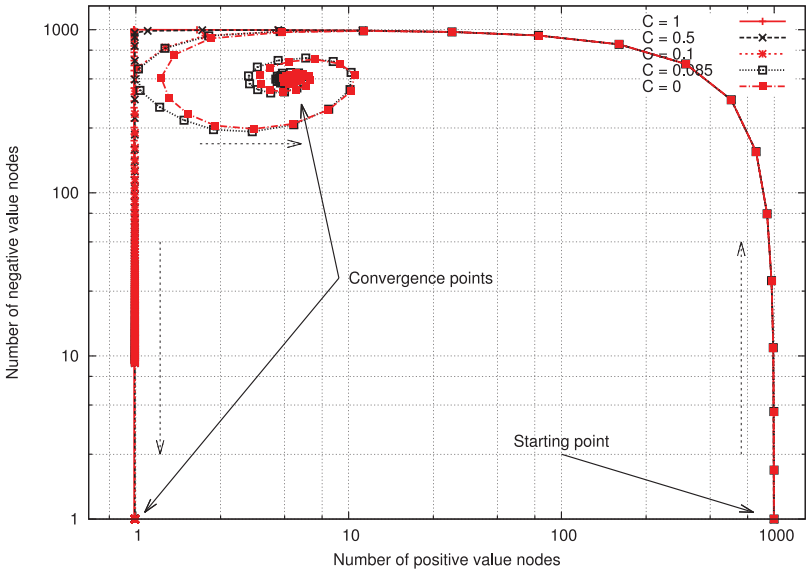

Fig. 7. Variation of the number of positive value nodes function of the number of negative value nodes ($\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}$; 1,000 nodes; initial configuration consists of a single predator node and 999 prey nodes; arrows indicate time evolution).

exhibiting oscillatory behavior—better observed in Figure 7, which plots the variation of the number of positive value nodes versus the number of negative value nodes. Intuitively, the oscillations and convergence value are the result of the negative value nodes being too short-lived to cover all the network. We have tested network sizes ranging from a few hundred nodes to several tens of millions of nodes and found that, for example, $\mathcal{C} = 0.5$ is a good choice guaranteeing system convergence at a fast rate.

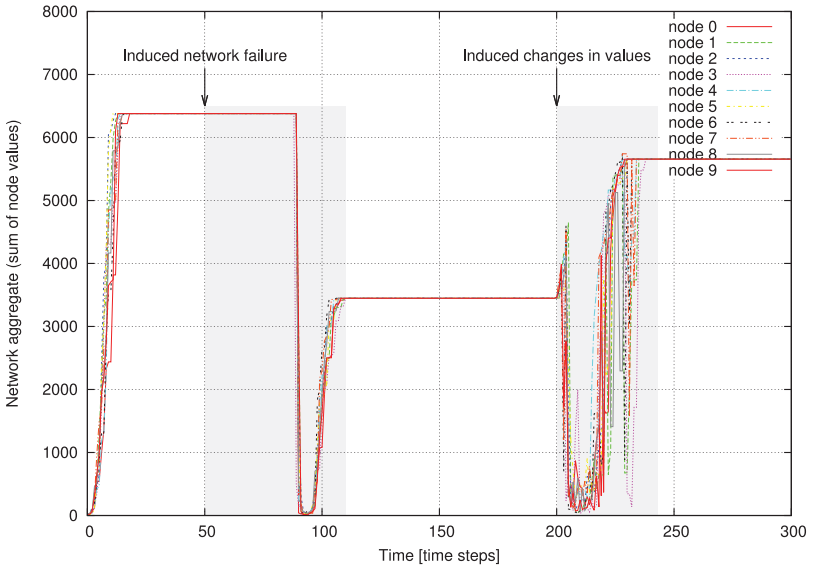

Fig. 8. Sum computation during network dynamics with value removal mechanism *enabled* (geometric random graph with 1,000 nodes initially; diameter 14; random values; half of the network is disconnected at time 50; 30% nodes change their values at time 200). Notice the reduction in transition period compared to Figure 9.

4. DISCUSSIONS

Our distributed approach solves most of the scaling issues and proves to be highly robust against network dynamics (e.g., network nodes becoming unavailable due to failures, reconfiguration, new nodes joining the system, etc.) as long as these changes occur in batches, on a time scale comparable to a time round. As we show in the following, our approach is very fast for a typical network, outperforming the speed of a centralized approach. As the protocols rely on anonymous data exchanges, privacy issues are alleviated, as the identities of the system participants are not needed in the computations.

The downsides of our approach maps onto the known properties of this class of epidemic algorithms. Although anonymity is preserved, an authentication system [Jesi et al. 2007] is needed to prevent malicious data corrupting the computations.

In the following subsections, we numerically characterize some of these properties. The simulations were performed using Matlab and C++. Nodes were randomly deployed onto a square surface and the circular transmission range was varied until the desired diameter of the network was obtained. All networks were verified to be formed of one main cluster, with disconnected nodes not considered. The communication model was push-pull gossip. We used synchronized time rounds to model time. Unless otherwise stated, each result point represents an average over 100 simulations.

4.1. Assumptions on Synchronized Changes

Although the two mechanisms introduced in Section 3 do not make use of any synchronization, their functionality is guaranteed only if the changes in the network happen in batches, with a period in the order $O(D \log N)$. The question we address in this section refers to what happens if this assumption is violated.

Figure 8 gives a graphical representation of the two mechanisms at work. The counter-based self-stabilization mechanism activates when part of the network is disabled (e.g., due to a permanent hardware failure, network partitioning, software

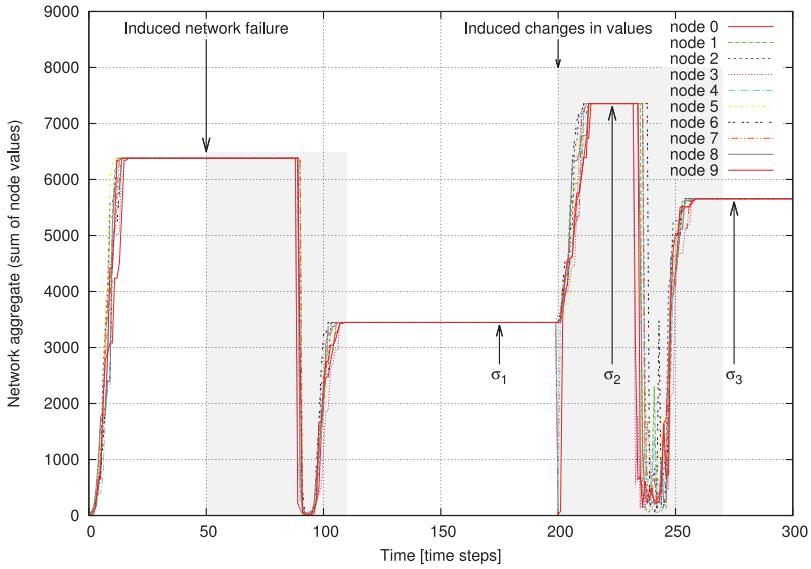

Fig. 9. Sum computation during network dynamics with value removal mechanism *disabled* (geometric random graph with 1,000 nodes initially; diameter 14; random values; half of the network is disconnected at time 50; 30% nodes change their values at time 200).

bug invalidating a number of nodes, etc.). While the timers decrease toward expiration (timesteps 50–80), the aggregate computed in the network remains constant. Once the timers on various nodes expire, a sudden "dip" occurs in the curve (see timesteps 80–90). This is due to the fact that once a minimum value expires on a node, the node replaces it with its own value from the original random value vector. As this is usually larger than the minimum, the computation in Algorithm 3 (Lines 7–19) leads to a very small value. As a new minimum propagates through the network, the computed value begins to rise toward the final value.

In the case in which we maintain the change in the network values at the timestep 200, with *the value removal mechanism disabled*, the network still converges to the proper value, only slower. This situation is presented in Figure 9. Let $\sigma_1 = \sum_{i=1}^{N} \lambda_i$ be the sum of the values in the network before the induced change in values at time 200 (the *initial sum*). After 30% of the nodes change their value, we notice a sudden drop in values at time 200, which we explained earlier. Then, the network converges to an intermediate sum σ_2 (see Figure 9) and the timers associated with the slow propagating values (see Table II and Section 3.2) decrease toward 0. Once they expire (around time 230), the network fluctuates once more and stabilizes to the value σ_3.

It is interesting to notice that the difference $\sigma_2 - \sigma_1$ equals *the sum of the values that changed*. In other words, assume that a subset of nodes $i \subset \mathcal{I}$ changed their values λ_i to λ_i' (30% of nodes in the example in Figure 9).

$$\sigma_2 = \frac{1}{\sum_{j=1}^{M} \min(\min_{i\in(1,N)}(\mathbf{v}_i[j]), \min_{i\subset\mathcal{I}}(\mathbf{v}_i'[j]))} \stackrel{\text{def}}{=} \sum_{i=1}^{N} \lambda_i + \sum_{i\subset\mathcal{I}} \lambda_i'$$

leading to

$$\sigma_2 - \sigma_1 = \sum_{i\subset\mathcal{I}} \lambda_i'. \tag{4}$$

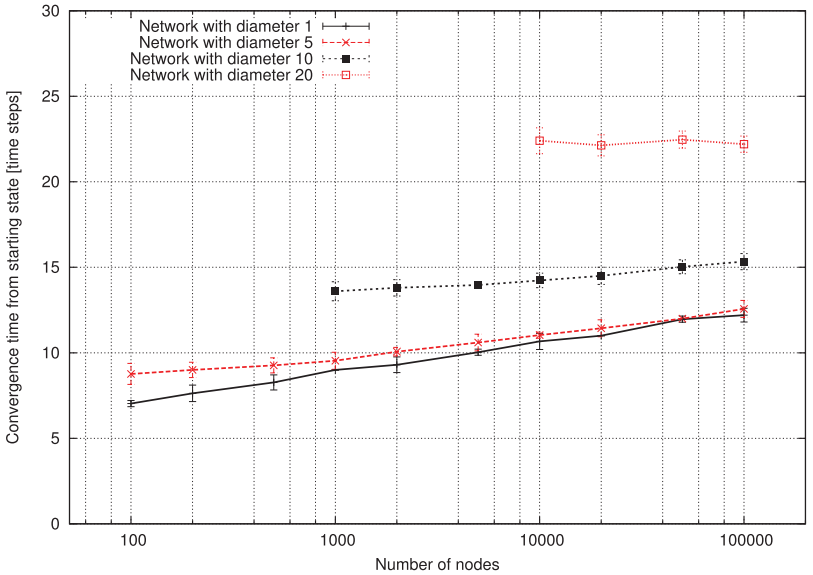

Fig. 10. Convergence of network starting from a clean state indicating that convergence time is basically independent of N and linearly dependent on the diameter D (geometric random graph; nodes initialized with random values; summation as local aggregate; error bars represent standard deviation; 100 simulations).

Employing the value removal mechanism speeds up the convergence process, by "removing" the intermediate convergence level σ_2 (in reality it exponentially expires the timers during this level) and leads to the situation shown in Figure 8. The information on how much change will occur in the network ($\sigma_2 - \sigma_1$) is no longer available before the network finally stabilizes.

4.2. Scalability Aspects

One of the main characteristics of our approach is that the algorithm we propose scales very well with the number of nodes in the network. As seen again Figure 10 and Figure 11, the number of nodes has little influence in the final results only as a $O(\log N)$ term. The simulation explored a space in which we varied the number of nodes over four orders of magnitude and the results hint that tighter boundaries might exist than the ones we proposed in this article. We noticed that for a fully connected network, the recovery time varies with 34% between a network with 1,000 nodes and one with 100,000 nodes, while the variation drops to a mere 2.4% for a 20-hop network varying from 1,000 nodes to 100,000 nodes.

These results are very important for large-scale network applications such as the smart energy grid. As the network will be linked to a physical space (a country or in general, a region), fully covering it, the diameter of the network is expected to, at most, decrease with the addition of new nodes. Intuitively, when thinking of nodes as devices with a fixed transmission range, adding more devices in the same region may lead to shorter paths between various points. The aggregate computation approach we propose shows, on one hand, an almost invariance to the increase in the number of nodes in the network and a linear variation with the diameter. These properties are essential for any solution that needs to take into account that the number of participants in the network will increase over time.

We are also interested in understanding the effects the time-to-live of the negative fields has on the convergence and scalability properties. We have considered a 10-hop

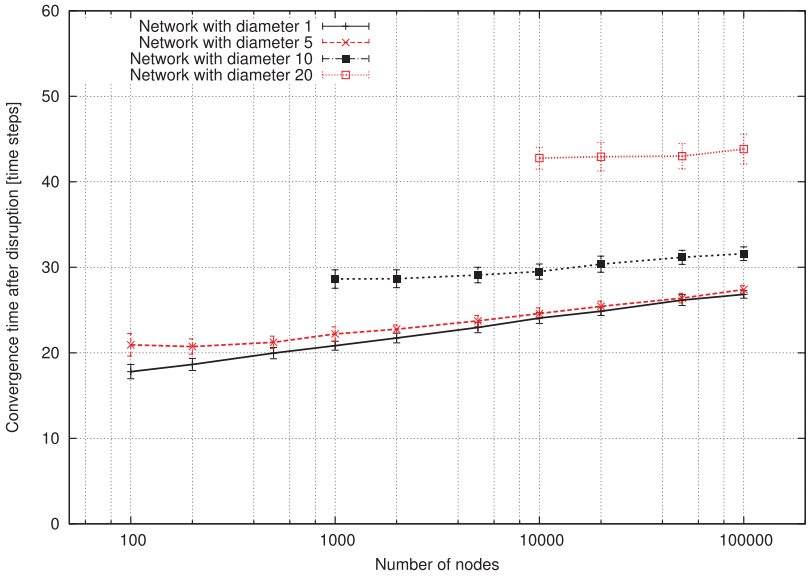

Fig. 11. Convergence of network after a disruption exhibiting the same characteristics as Figure 10 (geometric random graph; half of the nodes change their values after initial network convergence; summation as local aggregate; error bars represent standard deviation; 100 simulations).

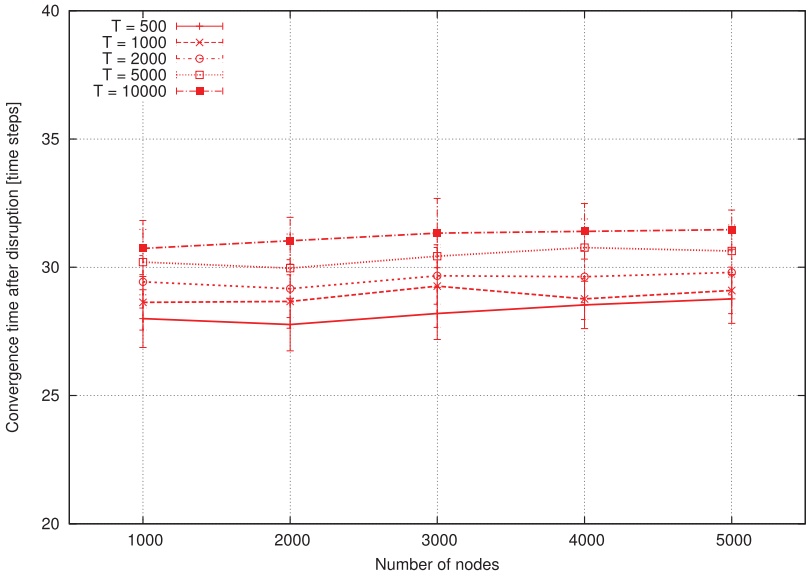

Fig. 12. Influence of T parameter—behavior is basically constant in T as long as it is larger than the diameter D of the network (random geometric graph; 10-hop network; half of the nodes change their values randomly after initial network convergence; summation as local aggregate; error bars represent standard deviation; 100 simulations).

network with 1,000 to 5,000 nodes and varied the time-to-live for negative values between 500 and 10,000. Figure 12 confirms Lemma 3.1 with respect to the $\log T$ term. As the data shows, the convergence time was affected very little by the chosen parameters. As expected, the diameter of the network has the larger influence in this mechanism.

4.3. Influence of Communication Topology

The underlying communication network for a large-scale network (such as a smart energy grid) can be implemented in a number of ways, mapping to different communication topologies. For example, one might choose to use the internet backbone, allowing any-to-any communication in the network, leading to a fully connected graph. In the first experiment, we have initialized the network with a set of random variables and recorded the time when the aggregated sum converges to the same value on all nodes. As seen in Figure 10, fully connected networks lead to the fastest aggregate computation. In a second experiment, once the network stabilized, we introduced a change in the form of half of the nodes in the network changing their value to a different one. Again, we recorded the time until the network stabilized after this change. As expected, Figure 11 shows that fully connected networks stabilize the fastest after a disruption.

These results assume the internet backbone to work perfectly and able to route the high level of traffic generated. A more realistic scenario is considering that the various data collection points obtain data from the individual consumers via some radio technology (e.g., GPRS modems) and are themselves connected to the internet backbone. To keep the traffic in the network to a minimum, the data collection points only communicate with their network-wise first-order neighbors, leading to a mesh network deployment type. As seen in Figure 10 and Figure 11, the diameter of the network clearly has the major impact factor on the results, confirming the theoretical results. The information needs at least $O(D)$ timesteps to propagate through the network. The constant in the $O()$ notation is influenced on one hand by the average connectivity in the network (a node can only contact a single neighbor per timestep, slowing information dissemination) and the push-pull communication model on the other (a node may be contacted by several neighbors during a timestep, speeding up information dissemination).

5. RELATED WORK

Aggregate computation in large-scale networks is a topic that received significant attention across a large number of fields. It is fueled by both need (ultra-large-scale systems make decision taking a highly complex task [Northrop et al. 2006]) and imagination (programmable matter requires a control paradigm [Goldstein et al. 2005]). A myriad of domain specific programming languages were developed trying to ease the task of controlling large-scale networks. The authors of Beal et al. [2013] survey efforts in fields including amorphous computing, synthetic biology, wireless sensor networks, pervasive computing, swarm robotics, and parallel and distributed computing, to name a few.

The basic building blocks of all these languages are primitives that guarantee that the system converges to a desired state. Complicated networking protocols are usually not employed as complexity was shown to arise even from combination of simple rules [Wolfram 2002]. This holds true for both simple cellular automata [Chopard and Droz 1998] and highly complex systems combining modern communication means with large-scale fixed infrastructure (e.g., autonomic computing [Kephart and Chess 2003]).

In this context, our research focuses on basic primitives, based on local interactions, which give designers the possibility of creating complex behaviors in a tractable manner. While solutions involving global information available at each node are easily rendered useless by increasing system scales (e.g., tracking the status of all individual nodes is infeasible), aggregate information about the system behavior is achievable. Gossiping [Boyd et al. 2005] is such a mechanism, allowing computation of aggregates in a timely manner without the need of precise topology information. Complex

functions can be achieved with simple local rules, ranging from statistical information about information distributed all over the network [Kempe et al. 2003] to network overlays [Jelasity and Babaoglu 2006]. The authors of Mosk-Aoyama and Shah [2008] show that a trade-off exists between the convergence time and the amount of information exchanged in the gossiping process, leading to fast converging algorithms (in $O(D \log N)$ timesteps) [Shah 2009].

It is interesting to notice that even simple aggregates received a lot of attention. For example, counting the nodes in a network (thus summing up all the values 1 held locally) led to a large number of solutions [Kostoulas et al. 2007; Massoulié et al. 2006; Madden et al. 2002]. Of interest and in line with the work presented in this article is the synopsis diffusion mechanism [Nath et al. 2004], which employs statistics techniques to achieve a result fast. Unfortunately, these works are usually one-shot solutions, requiring some type of synchronization mechanisms (preferably distributed solutions such as Werner-Allen et al. [2005]). Self-stabilization techniques [Dolev 2000] received a lot of interest leading to approaches such as periodic synchronization [Jelasity et al. 2005], parallel running algorithms [Bicocchi et al. 2010], or asynchronous periodic resets [Pruteanu and Dulman 2012].

Self-stabilization techniques like the ones proposed in this article are tightly related to the classical problem of leader election. The basis for our work, the Mosk-Aoyama-Shah algorithm [Mosk-Aoyama and Shah 2008], has as an underlying feature the propagation of a minimum value in a network—a common technique used in leader election protocols [Dijkstra 1982]. Using timers for detecting changes or deadlocks in the network state [Mayer et al. 1992] is ubiquitous in computer science protocols. Also, from the perspective of topology, a large number of leader-election algorithms present similarities with the underlying analysis of convergence [Shah 2009] (ring graphs [Garcia-Molina 1982], tree networks [Antonoiu and Srimani 1996], or mesh networks [Malpani et al. 2000]). More common techniques between these fields have already been identified in Angluin et al. [2008]. Nevertheless, the second extension we propose in this article breaks away from the classical topics of leader election and token management as the assumptions it builds upon (e.g., values of nodes in the network changing often) are rarely applicable in the works cited previously.

Returning to the problem of engineers having access to a design process for complex behavior, our work provides a measurement component for distributed control approaches. The primitives we propose can be used in creating complex algebras (such as the Presburger arithmetic employed in population protocols [Angluin et al. 2007, 2008]) or even in more exploratory solutions such as swarm chemistry [Sayama 2009].

The results of this theoretical work form the basis for building a wide range of applications. Internet of things, smart cities, and Industry 4.0 are just a few technologies highly relevant at the moment that may make use of our work, building on the lessons of already deployed systems (e.g., monitoring of cloud computing server farms—Astrolabe [Van Renesse et al. 2003] or computing complex robustness metrics in smart energy grid applications [Koç et al. 2013]).

6. CONCLUSIONS AND FUTURE WORK

In this article, we focused on adding self-stabilizing properties to fast gossiping algorithms. The motivation is that, in the case in which the changes in the network occur in batches separated in time, no additional synchronization mechanism is needed. A simple extension based on counters is enough to guarantee the network stabilization in a timely manner after the disturbance.

To this end, we propose two mechanisms (the counter-based self-stabilization and the value removal mechanism) that together allow fast gossiping algorithms to

withstand network dynamics. The basic properties of the fast gossiping algorithms still hold, leading to a solution that is highly scalable, network topology agnostic, has no single-point-of-failure and allows real-time results dissemination at all the nodes in the network.

Regarding the future work, we noticed that the self-stabilizing approaches to large-scale highly dynamic systems are quite limited in number, offering nice investigating venues. For example, we would like to extend the proposed counter-based mechanism to setups in which changes can occur without any restrictions. One direction of research includes limiting the fluctuations in the aggregate computation to a smaller dynamic range, preventing it from decreasing toward 0 once timers expire or values change.

REFERENCES

Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. 2007. The computational power of population protocols. *Distributed Computing* 20, 4 (2007), 279–304.

Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. 2008. Self-stabilizing population protocols. *ACM Transactions on Autonomous and Adaptive Systems* 3, 4 (2008), 13.

Gheorghe Antonoiu and Pradip K. Srimani. 1996. A self-stabilizing leader election algorithm for tree graphs. *Journal of Parallel and Distributed Computing* 34, 2 (1996), 227–232.

Roger Arditi and Lev R. Ginzburg. 1989. Coupling in predator-prey dynamics: Ratio-dependence. *Journal of Theoretical Biology* 139, 3 (1989), 311–326. DOI:http://dx.doi.org/10.1016/S0022-5193(89)80211-5

Jacob Beal, Stefan Dulman, Mirko Viroli, Nikolaus Correll, and Kyle Usbeck. 2013. Organizing the aggregate. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments* (2013), 436.

Nicola Bicocchi, Marco Mamei, and Franco Zambonelli. 2010. Handling dynamics in diffusive aggregation schemes: An evaporative approach. *Future Generation Computer Systems* 26, 6 (2010), 877–889.

Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. 2005. Gossip algorithms: Design, analysis and applications. In *Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, Vol. 3. IEEE, 1653–1664.

Bastien Chopard and Michel Droz. 1998. *Cellular Automata Modeling of Physical Systems*. Vol. 24. Cambridge University Press, Cambridge.

Edsger W. Dijkstra. 1982. Self-stabilization in spite of distributed control. In *Selected Writings on Computing: A Personal Perspective*. Springer, 41–46.

Shlomi Dolev. 2000. *Self-Stabilization*. MIT Press.

Hector Garcia-Molina. 1982. Elections in a distributed computing system. *IEEE Transactions on Computers* 100, 1 (1982), 48–59.

Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. 2005. Programmable matter. *Computer* 38, 6 (2005), 99–101.

Venkat Iyer, Andrei Pruteanu, and Stefan Dulman. 2011. Netdetect: Neighborhood discovery in wireless networks using adaptive beacons. In *2011 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'01),*. IEEE, 31–40.

Márk Jelasity and Ozalp Babaoglu. 2006. T-Man: Gossip-based overlay topology management. In *Engineering Self-Organising Systems*. Springer, 1–15.

Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. 2005. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems* 23, 3 (2005), 219–252.

Gian Paolo Jesi, David Hales, and Maarten van Steen. 2007. Identifying malicious peers before it's too late: A decentralized secure peer sampling service. In *1st International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07).* 237–246. DOI:http://dx.doi.org/10.1109/SASO.2007.32

David Kempe, Alin Dobra, and Johannes Gehrke. 2003. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science.*. IEEE, 482–491.

Jeffrey O. Kephart and David M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

Yakup Koç, Martijn Warnier, Robert E. Kooij, and Frances M. T. Brazier. 2013. A robustness metric for cascading failures by targeted attacks in power networks. In *Proceedings of the 10th IEEE International Conference on Networking, Sensing and Control (ICNSC'13),* IEEE, 48–53.

Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta, Kenneth P. Birman, and Alan J. Demers. 2007. Active and passive techniques for group size estimation in large-scale and dynamic distributed

systems. *Journal of Systems and Softwware* 80, 10 (Oct. 2007), 1639–1658. DOI: http://dx.doi.org/10.1016/j.jss.2007.01.014

Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2002. TAG: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Systems Review* 36, SI (Dec. 2002), 131–146. DOI: http://dx.doi.org/10.1145/844128.844142

Navneet Malpani, Jennifer L. Welch, and Nitin Vaidya. 2000. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*. ACM, 96–103.

Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. 2006. Peer counting and sampling in overlay networks: Random walk methods. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC'06)*. ACM, New York, NY, 123–132. DOI: http://dx.doi.org/10.1145/1146381.1146402

Alain Mayer, Yoram Ofek, Rafail Ostrovsky, and Moti Yung. 1992. Self-stabilizing symmetry breaking in constant-space. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. ACM, 667–678.

Damon Mosk-Aoyama and Devavrat Shah. 2008. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory* 54, 7 (2008), 2997–3007.

Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. 2004. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. ACM, 250–262.

Linda Northrop, Peter Feiler, Richard P. Gabriel, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman, Mark Klein, Douglas Schmidt, Kevin Sullivan, and others. 2006. Ultra-large-scale systems—The software challenge of the future. (2006).

Andrei Pruteanu and Stefan Dulman. 2012. LossEstimate: Distributed failure estimation in wireless networks. *Journal of Systems and Software* 85, 12 (2012), 2785–2795.

Hiroki Sayama. 2009. Swarm chemistry. *Artificial Life* 15, 1 (2009), 105–114.

Devavrat Shah. 2009. *Gossip Algorithms*. Now Publishers Inc.

Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. 2003. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems* 21, 2 (2003), 164–206.

Geoffrey Werner-Allen, Geetika Tewari, Ankit Patel, Matt Welsh, and Radhika Nagpal. 2005. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*. ACM, 142–153.

Stephen Wolfram. 2002. *A New Kind of Science*. Vol. 5. Wolfram Media Champaign.

Received January 2015; revised June 2015; accepted August 2015