Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Distributed Averaging on Asynchronous Communication Networks

Mortada Mehyar, Demetri Spanos, John Pongsajapan, Steven H. Low, Richard M. Murray
California Institute of Technology
{morr, demetri, pongsaja, slow, murray}@caltech.edu

Abstract— **Distributed algorithms for averaging have attracted interest in the control and sensing literature. However, previous works have not addressed some practical concerns that will arise in actual implementations on packet-switched communication networks such as the Internet. In this paper, we present several implementable algorithms that are robust to asynchronism and dynamic topology changes. The algorithms do not require global coordination and can be proven to converge under very general asynchronous timing assumptions. Our results are verified by both simulation and experiments on a real-world TCP/IP network.**

I. Introduction

This article focuses on a distributed iterative procedure for calculating averages over an asynchronous communication network. This style of asynchronous computing has seen a renewed interest in recent years as a consequence of new developments in low-cost wireless communication and local computation. While asynchronous iterative computing is not new in itself (see the classic reference [1]), some new twists arise when one attempts to implement such schemes on unstructured, packet-switched, communication networks.

Much recent research has focused on various distributed iterative algorithms. For a representative sample of developments in this area, we direct the reader to [2], [3], [4], [5], and references therein. We also refer the reader to [6] and [1] for a general and accessible overview of distributed asynchronous computing.

Averaging serves as a useful prototype for asynchronous iterative computations both because of its simplicity, and its applicability to a wide range of problems. On a sensor network, one may be interested in the average of physical measurements over the whole network. In problems that concern vehicle formation, the quantities being averaged can be the coordinates of the vehicles, and the average can represent the center of mass. A network of servers may wish to collaboratively calculate the average process load, in order to implement some load balancing scheme. A peer-to-peer file-sharing network on the Internet can compute other application-specific averages of interest.

In principle, one can choose to calculate averages by flooding the whole network with all the values, or by using structured message propagation over a known overlay network (e.g. a spanning tree). These are both natural methods for solving a distributed averaging problem, but the former has very large messaging complexity, and the latter requires a structured overlay network. Further, these require global exchange of information. While it is not clear that this is necessarily a problem in the applications we have discussed above, it seems likely that a scheme involving only local exchange may be desirable.

In many scenarios an exact average is not required, and one may be willing to trade precision for simplicity. The scalability, robustness, and fault-tolerance associated with iterative schemes can be superior in many situations where exact averaging is not essential. These schemes also resolve the global exchange problem, as they only require communication of variables among local neighbors.

In this paper, we will present two practically implementable iterative algorithms, and show their convergence in a general asynchronous environment. Our analysis is verified by simulation and experiments on a real-world TCP/IP network. In combination, these results show that the method proposed is both analytically understandable, and practically implementable.

II. Background and Problem Setup

Consider a network, modeled as a connected undirected graph $G = (V, E)$. We refer to the vertices (elements of V) as nodes, and the edges (elements of E) as links. The nodes are labeled $i = 1, 2, \ldots, n$, and a link between nodes i and j is denoted by ij.

Each node has some associated numerical value, say z_i, which we wish to average over the network. We will refer to the vector $\mathbf{z}$ whose ith component is z_i. Each node on the network also maintains a dynamic variable x_i, initially set to the static value z_i. We call x_i the *state* of the node i.

When we wish to show the time dependence, we will use the notation $x_i(t)$. We use the notation $\mathbf{x}$ to denote the vector whose components are the x_i terms. Intuitively each node's state $x_i(t)$ is its current estimate of the average value of the network $\sum_{i=1}^{n} z_i/n$. The goal of the averaging algorithms, is to let *all* states $x_i(t)$ go to the average $\sum_{i=1}^{n} z_i/n$, as $t \to \infty$.

The work in [7] proposes the following discrete-time system as a mechanism for calculating averages in a network:

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \gamma L \mathbf{x}(t) \tag{1}$$

where γ is a stepsize parameter, and L is the *Laplacian* matrix associated with the undirected graph G.

The Laplacian matrix L is defined as

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -1 & \text{if there is a link between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

0-7803-9568-9/05/$20.00 ©2005 IEEE

where d_i is the degree, or the number of neighbors node i has. The algorithm (1) can be viewed as an iterative gradient method for solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbf{R}^n} \quad & \tfrac{1}{2}\mathbf{x}^T L\mathbf{x} \\ \text{s.t.} \quad & \textstyle\sum_i x_i = \sum_i z_i \quad . \end{aligned}$$

Therefore it is not hard to show that this algorithm drives all states x_i to the average, provided the stepsize γ satisfies

$$0 < \gamma < \frac{1}{2d_{\max}}$$

where $d_{\max}$ is the maximum of all the node degrees d_i.

Other authors have also considered similar iterative mechanisms, including the work of [8] which examines the possibility of topology optimization for maximizing the convergence rate of the algorithm.

Unfortunately, all of these results share the drawback of not being directly implementable on a packet-switched network like the Internet. The main problem is the implicit synchronization. Real-world networks constitute an inherently asynchronous environment with dynamic network delays; synchronization is impractical and undesirable. Another problem with the algorithm (1) is that each node must use exactly the same stepsize. Moreover, the allowable stepsize bound depends on global properties of the network. This information is not available locally and therefore global coordination must be involved. In the following sections, we will propose algorithms that do not require synchronization or global coordination.

III. Algorithm A1

In this section we will introduce our first algorithm. We denote it A1 in distinction to another algorithm we will show later. At each node i there is a local stepsize parameter γ_i, $0 < \gamma_i < 1$ upon which the node's computation is based. They do not need to be coordinated.

The fundamental "unit" of communication in our scheme is a pairwise update between two nodes. We require two (distinguishable) types of messages, identified in a header. We refer to these two types as *state* messages and *reply* messages. An update is initiated whenever a node sends out a state message containing the current value of its state.

An overview of the message-passing scheme that will enable the pairwise update is as follows:

MP1: At some time, node i initiates a *state* message containing its current state value to some other node j. At some later time, node j receives this message.

MP2: Node j implements a local computation based on the value it receives. It records the result of this computation in a *reply* message, and sends this message back to node i.

MP3: At some later time, node i receives j's reply, and implements a local computation based on the content of the reply message.

In addition to the message-passing scheme, in order to make sure that communications between different pairs of nodes will not interfere, we require that the nodes implement *blocking*. Whenever a node sends out a state message, it does not reply to incoming state messages until it receives a reply from the receiver. Instead, it sends back a negative acknowledgement (NACK) indicating that it already has a pairwise update in progress. It also does not initiate any other updates while blocking.

Whenever a node receives a NACK, the update terminates prematurely with no effect on either of the local variables, and the node stops blocking. With the blocking mechanism in place, a pairwise update is specified as follows:

PW1: Node j receives a state message from node i. If it is blocking, it does nothing and sends a NACK to node i.

PW2: Otherwise, it implements $x_j \leftarrow x_j + \gamma_j(x_i - x_j)$.

PW3: Then, it generates a reply message containing the numerical value $\gamma_j(x_i - x_j)$, and sends it to i.

PW4: Node i receives the reply message, and implements $x_i \leftarrow x_i - \gamma_j(x_i - x_j)$.

Note that node i does not need to know γ_j; all it needs to know is how much change node j has made, which is contained in the reply message. Also note that after an update, node i has exactly compensated the action of node j, in the sense that the *sum of the states is conserved.*

For the moment, we do not specify the timing or triggering for this event; we will propose one possible scheme (implementation) in section V. We will merely make the following assumption:

Eventual Update Assumption: for any link ij and any time t, there exists a later time $t_l > t$ such that there is an update on link ij at time t_l.

This assumption is very similar to the totally asynchronous timing model in [1]. It turns out that this very general asynchronous timing assumption is sufficient to guarantee convergence of the state values under algorithm A1.

IV. Convergence of Algorithm A1

Because of the blocking behavior, updates that happen on one link will never interfere with updates on another. This generates a property that is very useful for analysis:

> *With blocking, although updates on different links can span overlapping time intervals, the resulting state values of the network will be as if the updates were non-overlapping, and therefore sequential in time.*

Thus, aside from the timing details of when updates are initiated, it is equivalent to consider a sequence of synchronized updates enumerated in discrete time $T = \{0, 1, 2, ...\}$, and there is only one update at each time instant. We will do so for the purposes of the analysis to follow.

We need to show that any algorithm satisfying the Eventual Update Assumption and implementing the interaction

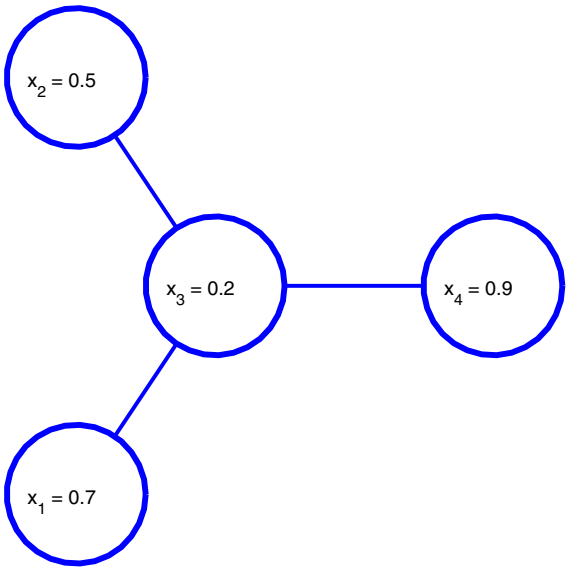

Fig. 1. An example network consisting of four nodes in a "star" topology.

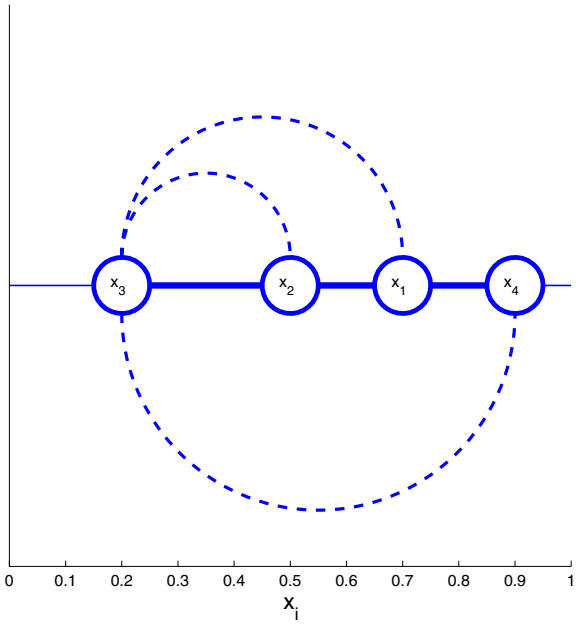

Fig. 2. The four node network embedded on the real line according to node value x_i. The bold lines indicate *segments*, i.e. intervals on the real line separating two adjacent values. The dashed curves indicate the communication topology from Figure 1. Thus, an update on the link between node 1 and node 3 will *claim* two segments, $[x_3, x_2]$ and $[x_2, x_1]$.

(with blocking) described in section III must converge to the average. Our proof will make use of the following "potential" function:

$$P(t) = \sum_{\forall(i,j)} |x_i(t) - x_j(t)| \tag{2}$$

where the sum is over all $\frac{n(n-1)}{2}$ possible pairs (i, j). For instance, the potential function for the network in Figure 1 is

$$\begin{aligned} &|x_1 - x_2| + |x_1 - x_3| + |x_1 - x_4| \\ &+ |x_2 - x_3| + |x_2 - x_4| + |x_3 - x_4| \end{aligned}$$

Lemma 1: If nodes (i, j) update at time t while node i being the sender, then at the next time unit $t + 1$

$$P(t+1) \leq P(t) - 2\min\{\gamma_j, 1-\gamma_j\}|x_i(t) - x_j(t)| \tag{3}$$

Proof: In summary at time $t + 1$

$$\begin{cases} x_i(t+1) = (1-\gamma_j)x_i(t) + \gamma_j x_j(t) \\ x_j(t+1) = \gamma_j x_i(t) + (1-\gamma_j)x_j(t) \\ x_k(t+1) = x_k(t), \ \forall k \neq i, j \end{cases} \tag{4}$$

Therefore besides the term $|x_i - x_j|$, $n - 2$ terms of the form $|x_k - x_j|$ and $n-2$ terms of the form $|x_i - x_k|$, $k \neq i, j$ in the potential function $P(t)$, are affected by the update. First of all we have

$$|x_i(t+1) - x_j(t+1)| = |(1-2\gamma_j)||x_i(t) - x_j(t)| \tag{5}$$

Now consider the sum of two of the affected terms $|x_k(t) - x_i(t)| + |x_k(t) - x_j(t)|$. If we look at the relative positions of $x_i(t)$, $x_j(t)$, and $x_k(t)$ on the real line, then *either* x_k is on the same side of x_i and x_j *or* it is in between them. Therefore as long as $0 < \gamma_i < 1$, it is clear geometrically in both cases we have

$$\begin{aligned} &|x_k(t+1) - x_i(t+1)| + |x_k(t+1) - x_j(t+1)| \\ &\qquad \leq |x_k(t) - x_i(t)| + |x_k(t) - x_j(t)| \end{aligned}$$

Therefore

$$\begin{aligned} P(t+1) - P(t) &\leq |x_i(t+1) - x_j(t+1)| \\ &\quad -|x_i(t) - x_j(t)| \\ &\leq -2\min\{\gamma_j, 1-\gamma_j\}|x_i(t) - x_j(t)| \end{aligned}$$

■

The quantity $\min\{\gamma_j, 1 - \gamma_j\}$ can be thought of as an effective stepsize for node j since a stepsize of .6, say, is equivalent to .4 in terms of reducing the relative difference in absolute value.

Lemma 2: At any time t, there exists a later time $t' > t$ such that at time t' there has been at least one update on every link since time t. Furthermore,

$$P(t') \leq \left(1 - \frac{8\gamma^*}{n^2}\right) P(t) \tag{6}$$

where $\gamma^* = \min_i \min\{\gamma_i, 1 - \gamma_i\}$

Proof: Without loss of generality, suppose at time t we have $x_1(t) \leq x_2(t) \leq ... \leq x_n(t)$. We call the $n - 1$ terms of the form $|x_i(t) - x_{i+1}(t)|$, $i \in \{1, 2, ..., n-1\}$, *segments* of the network at time t. By expanding every term in the potential function as a sum of segments, we see that the potential function can be written as a linear combination of all the segments:

$$P(t) = \sum_{i=1}^{n-1} (n-i) i \left| x_i(t) - x_{i+1}(t) \right| \tag{7}$$

We say that a segment $|x_i(t) - x_{i+1}(t)|$ at time t is *claimed* at time $t' > t$, if there is an update on a link of nodes r and s such that the interval $[x_s(t'), x_r(t')]$ (on the real line) contains the interval $[x_i(t), x_j(t)]$. For instance, for the network in Figure 1, the segments are $|x_3 - x_2|$, $|x_2 - x_1|$, and $|x_1 - x_4|$, as shown in Figure 2. Thus, an update on the link between node 1 and node 3 will claim segments $[x_3, x_2]$ and $[x_2, x_1]$.

Clearly by using the Eventual Update Assumption on each link, the existence of t' is guaranteed. From Lemma 1 it is clear that whenever a segment is claimed, it contributes a reduction in the potential function proportional to its size (see (3)). Referring to Figure 2, it is clear an update that does not claim a segment can only leave the segment unchanged or make it larger. Therefore no matter *when* a segment is claimed after time t, it will contribute at least $2\gamma^* |x_i(t) - x_{i+1}(t)|$ reduction in the potential function.

Now connectedness of the network implies that for each segment, there is at least one link such that an update on that link will claim the segment. Therefore by time t' *all* segments will be claimed. Thus the total reduction in the potential function between t and t' is at least

$$2\gamma^* \sum_{i=1}^{n-1} \left| x_i(t) - x_{i+1}(t) \right|.$$

It follows that

$$\begin{aligned} P(t') &\leq P(t) - 2\gamma^* \sum_{i=1}^{n-1} \left| x_i(t) - x_{i+1}(t) \right| \\ &= \left(1 - \frac{\sum_{i=1}^{n-1} 2\gamma^* \left| x_i(t) - x_{i+1}(t) \right|}{\sum_{i=1}^{n-1} (n-i) i \left| x_i(t) - x_{i+1}(t) \right|} \right) P(t) \\ &\leq \left(1 - \frac{8\gamma^*}{n^2} \right) P(t) \end{aligned}$$

where in the last inequality we use the fact that $n(n-i) \leq n^2/4$. ∎

With the above lemmas, we are ready to show convergence:

Theorem 1: $\lim_{t\to\infty} x_i(t) = \frac{1}{n} \sum_{i=1}^{n} z_i$, i.e. the average of the initial conditions of the network, $\forall i \in \{1, 2, ..., n\}$.

Proof: Repeatedly applying Lemma 2, we see that

$$\lim_{t\to\infty} P(t) = 0 \tag{8}$$

Therefore

$$\lim_{t\to\infty} \left| x_i(t) - x_j(t) \right| = 0, \forall i, j \tag{9}$$

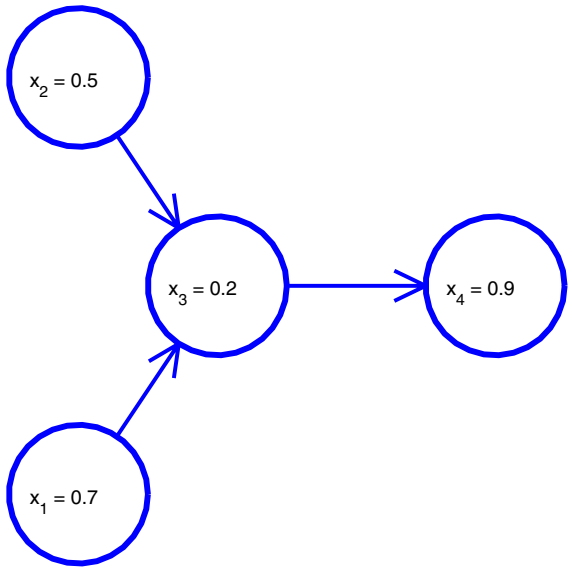

Fig. 3. The graph H for the example network, where the node indices are taken as the UIDs.

now by the conservation property

$$\sum_{i=1}^{n} x_i(t) = \sum_{i=1}^{n} z_i, \forall t \tag{10}$$

we see that

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n} \sum_{i=1}^{n} z_i. \tag{11}$$

∎

V. Implementation and Deadlock Avoidance

Any implementation that satisfies the Eventual Update Assumption is within the scope of the convergence proof of A1. However we have not, as yet indicated a specific mechanism for the update triggering. Caution must be taken because of the blocking behavior. Without a properly designed procedure for initiating communication, the system can drive itself into a deadlock.

Below we present one particular implementation based on a round-robin initiation pattern, which provably prevents deadlock and satisfies the updating assumption. This is by no means the only way to carry this out, but it has the advantage of being simple and easy to implement.

Our implementation will be based on some unique identifiers (UID), e.g. IP address. Based on these UIDs, we impose an additional directed graph $H = (V, F)$, in which an edge points from i to j if and only if node j has a higher UID than node i. This graph has two important properties:

H1: H has at least one root, i.e. a node with no inbound edges.

H2: H is acyclic.

This graph is illustrated for our four-node example network in Figure 3.

Our proposed initiation scheme is as follows:

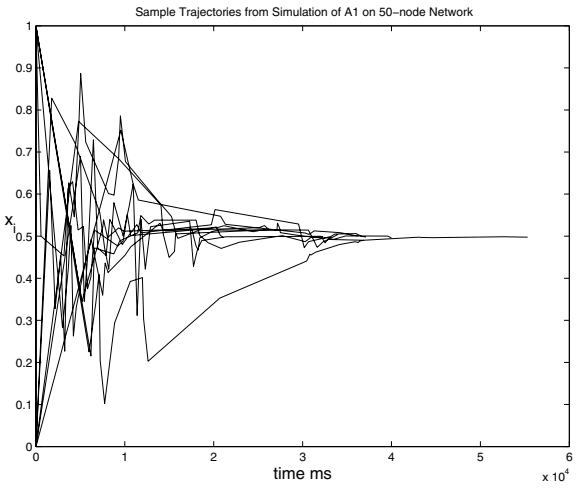

Fig. 4. State histories from a simulation of the provably convergent algorithm A1 on a fifty-node network. Round-trip delays on each link were assigned randomly, between 40 (ms) and 1000 (ms). Note that this represents forty seconds of simulation time; this is clear motivation for the more aggressive algorithm A2.

RR1: A node will wait to observe updates from *all* of its inbound edges.

RR2: The node will then sequentially initiate communication with each of its outbound edges, ordered by UID.

RR3: Upon completion, it repeats, waiting for all of its inbound edges and so on.

Lemma 3: The above procedure guarantees that the Eventual Update Assumption is satisfied.

We will prove this by contradiction. Suppose there is a a link ij and an interval $[t, \infty)$ during which this link does not update. Then, node i must be waiting for one of its inbound edges to be activated, implying the existence of a node k with a UID lower than that of i, which is also waiting for one of its inbound edges to be activated. Repeating this argument, and utilizing the fact that H is acyclic, we can find a path of inactive edges beginning at a root. However, a root has no inbound edges, and hence *must* initiate communication on all of its outbound edges at some point in $[t, \infty)$. This is a contradiction, and proves the desired result.

VI. Simulation of A1

We have written a discrete event simulator in Java and simulated algorithm A1. Below, we present a simulation of A1 with 50 nodes on a random topology with maximum degree 5. The stepsizes were chosen to be .5 for all nodes and the round-trip delays on the links were uniformly randomly distributed from 40(ms) to 1000(ms). Half of the nodes started with initial states 0 and the others with 1; the target average was therefore .5. The results of this simulation are shown in Figure 4.

VII. Algorithm A2

The blocking behavior for algorithm A1 requires occasional dropping of packets, which may not be desirable when node power is a scarce resource. Moreover, it constitutes most of the coding complexity in the implementation of A1. As an alternative, we will propose another algorithm, denoted by A2.

We will denote the set of all neighbors of node i to be N_i. In A2, each node i has an additional set of dynamic variables δ_{ij}, for all $j \in N_i$ (namely, one for each neighbor). In other words, if there is a link connecting nodes i and j, there will a dynamic variable δ_{ij} stored with node i, and another dynamic variable δ_{ji} stored with node j. The algorithm A2 is specified in terms of the x_i's and the δ_{ij}'s as follows in the synchronous environment:

$$\begin{cases} x_i(t+1) = x_i(t) + \gamma_i \left[\sum_{j \in N_i} \delta_{ij}(t) + z_i - x_i(t) \right] \\ \delta_{ij}(t+1) = \delta_{ij}(t) + \phi_{ij} \left[x_j(t) - x_i(t) \right] \end{cases} \quad (12)$$

Under A2, each x_i is initialized to z_i as in A1, and each δ_{ij} is initialized to 0. If there is a link between i and j, the parameters ϕ_{ij} and ϕ_{ji} are set to be equal. The variables δ_{ij} and δ_{ji} on each link ij can be arranged to conserve their sum (i.e., 0) in the same way A1 conserves the sum of states.

It can be shown that the above mapping is a contraction mapping, provided the stepsizes γ_i and ϕ_{ij} satisfy

$$\begin{cases} 0 < \gamma_i < \frac{1}{d_i + 1} \\ 0 < \phi_{ij} < \frac{1}{2} \end{cases}$$

Notice that the stepsize constraints are local: each node only needs to know the local degree d_i to determine the above stepsize bounds.

In the convergence proof of A1, we have used the fact that there are no overlapping updates on adjacent links. Therefore we can ignore the message-passing details and just consider each complete pairwise update in a sequence of discrete time instants as in (4). A2 does not impose the blocking constraint and thus it does not enjoy this nice property for analysis.

In particular, under A2, after node i sends out a state message to node j, node i is allowed to immediately send out another state message to some other neighbor k regardless of when node j's reply arrives. While waiting for a reply from node j, node i can also accept any incoming reply messages, and any state messages from all neighbors (including node j).

We enumerate all these message-passing events in the set $T = \{0, 1, 2, ...\}$, and let $T^{ij} \subset T$ be the set of times when δ_{ij} updates its value, and $T^i \subset T$ be the set of times when x_i updates its value. Equations (12) become

$$\begin{cases} x_i(t+1) = x_i(t) + \gamma_i \left[\sum_{j \in N_i} \delta_{ij}(\tau_{ij}^i) + z_i - x_i(t) \right], \text{ if } t \in T^i \\ x_i(t+1) = x_i(t), \text{ if } t \notin T^i \\ \delta_{ij}(t+1) = \delta_{ij}(t) + \phi_{ij} \left[x_j(\tau_j^{ij}) - x_i(\tau_i^{ij}) \right], \text{ if } t \in T^{ij} \\ \delta_{ij}(t+1) = \delta_{ij}(t), \text{ if } t \notin T^{ij} \end{cases}$$

where $0 \leq \tau_i^{ij}, \tau_j^{ij}, \tau_{ij}^i \leq t$ indicate possibly "old" copies of the variables involved in the update equations. (see [1] for more details.)

It can be shown that the following asynchronous timing assumption guarantees convergence of all the states to the desired average value:

Total asynchronism: (as defined in [1]) Given any time t_1, there exists a later time $t_2 > t_1$ such that

$$\tau_i^{ij}(t) \geq t_1, \tau_{ij}^i(t) \geq t_1, \forall i, j, \text{ and } t \geq t_2 \qquad (13)$$

This is in spirit similar to the Eventual Update Assumption for A1. In general, we have the following asynchronous convergence theorem for A2:

Theorem 2:

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n}\sum_{i=1}^{n} z_i, \forall i$$

under A2 with total asynchronism, provided the stepsizes satisfy

$$\begin{cases} 0 < \phi_i < \frac{1}{d_i+1} \\ 0 < \gamma_{ij} < \frac{1}{2} \end{cases}$$

Proof: Due to space limitations we will only sketch the proof. It can be shown that given the stepsize constraints, the synchronous equations are a contraction mapping with respect to infinity norm. Also, the linear part of the mapping satisfies the diagonal dominance property. Using Proposition 2.1 of Section 6.2 in [1], A2 converges under total asynchronism. ∎

VIII. Experimental Results

We developed an implementation of A2 in a C socket program and deployed it on the PlanetLab network [10]. We performed several runs of the algorithm, each time randomly choosing 50 to 100 nodes. Round-trip delays on this network ranged between tens of milliseconds and one second. Various overlay topologies were tested, with consistent convergence on the order of a few seconds.

In each experimental run, every node obtained a list of neighbors from a central server and established TCP connections to its neighbors. After the topology-formation phase was completed, the nodes were each sent a message instructing them to begin the iterative computation with their neighbors. One sample of these experimental results is shown in Figure 5. Note that despite comparable round-trip times to those used in the simulation of A1, the experimental results for A2 show much more rapid convergence.

IX. Summary, Conclusion, and Future Work

We have presented a class of practically implementable distributed iterative averaging algorithms, inspired by the Laplacian algorithm of [7]. Our algorithms do not rely on synchronization, knowledge of the global topology, or coordination of parameter values. The iterative nature of the algorithm renders it robust to changes in topology.

Our analytical results for A1 show that under a mild timing assumption, the asynchronous message-passing algorithms can achieve exponential convergence. Despite this, the blocking behavior of A1 is unduly conservative, and so we have introduced the significantly more aggressive algorithm A2, which is also provably convergent under very general asynchronous timing.

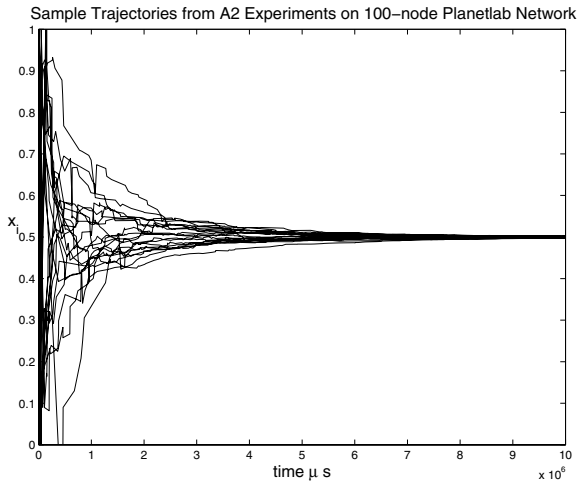

Fig. 5. Sample histories from an experiment on the PlanetLab network, using one-hundred nodes and the completely asynchronous algorithm A2. Round trip times on this network ranged between tens of milliseconds to approximately half a second. Note the rapid convergence of the estimates.

We have presented simulations, as well as experimental results from a real-world TCP/IP network. These results demonstrate the desired convergence behavior, and show that the algorithms proposed can be implemented robustly in a practical network.

References

[1] D.P. Bertsekas and J.N. Tsitsiklis, "Parallel and distributed computation: Numerical methods," *Prentice Hall*, 1989.

[2] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analaysis," *Proceedings of STOC*, 2004.

[3] D. Kempe, A. Dobra, and J. Gehrke, "Computing aggregate information using gossip," *Proceedings of FOCS*, 2003.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip and mixing times of random walks on random graphs," *Proceedings of STOC*, 2004.

[5] M. Jelasity, W. Kowalczyk, and M. van Steen, "An approach to massively distributed aggregate computing on peer-to-peer networks," *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004.

[6] N. Lynch, "Distributed algorithms," *Morgan Kaufmann Publishers*, 1997.

[7] R. Olfati-Saber and R. Murray, "Consensus Problems in Networks of Agents with Switching Topology and Time-Delays," *IEEE Trans. on Automatic Control*, vol. 49, no. 9, September 2004.

[8] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Proceedings of the Conference on Decision and Control*, 2003.

[9] D.P. Spanos, R. Olfati-Saber, and R.M. Murray, "Dynamic consensus on mobile networks," *Preprint (Submitted to IFAC 05)*, 2005.

[10] PlanetLab http://www.planet lab.org, ," .