Asynchronous Distributed Averaging on Communication Networks

Mortada Mehyar, Demetri Spanos, John Pongsajapan, Steven H. Low, *Senior Member, IEEE*, and Richard M. Murray

***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 are completely distributed and do not require any global coordination. In addition, they can be proven to converge under very general asynchronous timing assumptions. Our results are verified by both simulation and experiments on Planetlab, a real-world TCP/IP network. We also present some extensions that are likely to be useful in applications.**

***Index Terms*—Asynchronous computation, distributed averaging.**

I. Introduction

THIS PAPER 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 references of Bertsekas and Tsitsiklis [1] and Lynch [8]), 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. Distributed averaging, also known as the distributed consensus problem, has been studied in the context of vehicle formation control by Fax and Murray (e.g., [4], [11]). Similar algorithms have been applied to sensor fusion by Spanos, Olfati-Saber, and Murray ([14], [16]), as well as Xiao, Boyd, and Lall [20]). Gossip algorithms are algorithms with communication only between neighbors (which is also what we mean by "distributed" in this paper). Previous works in the Gossip algorithm context utilize probabilistic frameworks and analyze global behavior (see the work of Boyd *et al.* [2], Kempe *et al.* [6], [7], and references therein).

The "agreement algorithm" was proposed in the work of Tsitsiklis [17] and [18], and it is concerned with letting a distributed set of processors converge to some common value. It is worth noticing that a related algorithm for load balancing is also discussed in [1].

Manuscript received August 5, 2005; revised January 19, 2006, and March 1, 2006; approved by IEEE/ACM Transactions on Networking Editor A. Orda.

The authors are with the California Institute of Technology, Pasadena, CA 91125 USA (e-mail: morr@caltech.edu; demetri@caltech.edu; pongsaja@caltech.edu; slow@caltech.edu; murray@caltech.edu).

Digital Object Identifier 10.1109/TNET.2007.893226

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 (e.g. BitTorrent [3], Kazaa [5]) 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 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. Comparison With Other Work

Other authors have considered similar iterative mechanisms, including the work of Xiao and Boyd [19] which examines the possibility of link weight optimization for maximizing the convergence rate of the algorithm. The issues of asynchronism have also been studied. An asynchronous time model is analyzed in Boyd *et al.* [2] which assumes that each node has a clock ticking at the times of a Poisson process. In comparison, the asynchronous model we will use in this paper does not assume any stochastic properties. Also, we do *not* assume that neighboring nodes can simultaneously exchange values, as is implicit in Boyd *et al.* [2].

The Push-Sum algorithm (discussed in Kempe *et al.* [6]) is a Gossip algorithm that can be used to calculate sums and averages on a network. In the synchronous settings of Push-Sum, some probabilistic characterization of the convergence rate can be obtained. The convergence rate of Push-Sum is shown in [6] to depend on the logarithm of the size of the network. In comparison to Push-Sum, our algorithms do not assume a Gossip-like randomized communication scheme. Instead, we propose message passing mechanisms to enable communication among nodes. Also, the convergence rate of our algorithms does not in general depend on the size of the network, but only on the algebraic connectivity of the network. These points will be explained in detail in the later sections.

III. 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 $\sum_{i=1}^n z_i/n$. 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 of Olfati-Saber and Murray [11] 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 (see, e.g. [10].)

The Laplacian matrix L is defined as

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

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:

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

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.

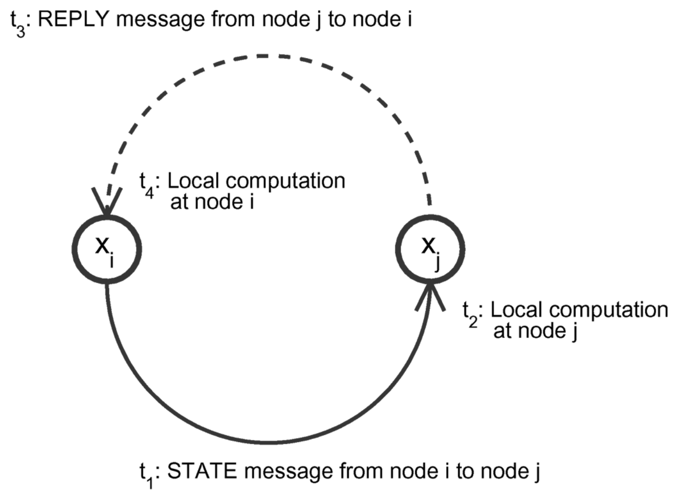

t_3: REPLY message from node j to node i
t_4: Local computation at node i
x_i
x_j
t_2: Local computation at node j
t_1: STATE message from node i to node j

Fig. 1. Illustration of the message-passing scheme.

These results implicitly assume 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 present our algorithms along with their message passing schemes and convergence analysis.

IV. Algorithm A1

In this section we will introduce our first algorithm A1. 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 basic "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 (Fig. 1) 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 sends a reply message containing the numerical value $\gamma_j(x_i - x_j)$ to node i, and then implements $x_j \leftarrow x_j + \gamma_j(x_i - x_j)$.

PW3: 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 at the end of a pairwise 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 VI. 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. We will show that any algorithm satisfying the Eventual Update Assumption and implementing the interaction (with blocking) described in Section IV must converge to the average.

V. 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 at the conclusion of each pairwise update 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 pairwise updates enumerated in discrete time $T = \{0, 1, 2, \ldots\}$, and there is only one update at each time instant. We will do so in the analysis to follow. The evolution of each state x_i under A1 can therefore be understood by considering the following update equations:

$$\begin{cases} x_i(t+1) = (1-\gamma_j)x_i(t) + \gamma_j x_j(t) \\ x_j(t+1) = \gamma_j x_i(t) + (1-\gamma_j)x_j(t) \\ x_k(t+1) = x_k(t), \forall k \neq i, j, \end{cases} \tag{2}$$

where x_j is the receiver and thus its local γ_j is used instead of γ_i.

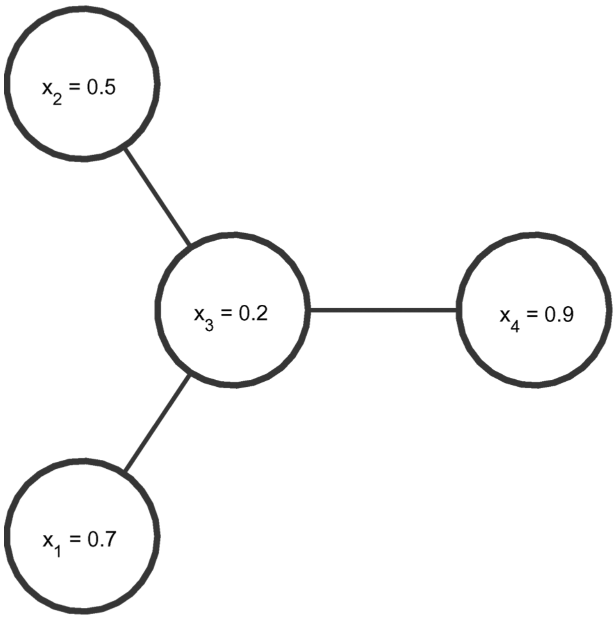


Fig. 2. An example network consisting of four nodes in a "star" topology.

Theorem 1: If the Eventual Update Assumption is satisfied, the A1 algorithm guarantees that

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n}\sum_{i=1}^{n} z_i \quad \forall i \in \{1, 2, \ldots, n\} \tag{3}$$

i.e., all node states converge to the average of the initial states of the network.

Similar convergence results can be found in [18] in the context of agreement algorithms. Our proof of Theorem 1 will make use of the following "potential" function:

$$P(t) = \sum_{\forall (i,j)} |x_i(t) - x_j(t)| \tag{4}$$

where the sum is over all $n(n-1)/2$ possible pairs (i, j). For instance, the potential function for the network in Fig. 2 is

$$|x_1 - x_2| + |x_1 - x_3| + |x_1 - x_4| + |x_2 - x_3| + |x_2 - x_4| + |x_3 - x_4|.$$

It can be shown that the potential function decreases in the following manner.

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

$$P(t+1) \le P(t) - 2\min\{\gamma_j, 1-\gamma_j\}\, |x_i(t) - x_j(t)|. \tag{5}$$

Proof: We can see from (2) that 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. We also have

$$|x_i(t+1) - x_j(t+1)| = |(1-2\gamma_j)|\, |x_i(t) - x_j(t)|. \tag{6}$$

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 in between x_i and x_j *or* it is not. Therefore, as long as $0 < \gamma_i < 1$, it is clear geometrically in both cases that we have

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

Therefore, together with (4) and (6) we have

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

■

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,

$$P(t') \leq \left(1 - \frac{8\gamma^*}{n^2}\right) P(t) \tag{7}$$

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

Proof: Without loss of generality, suppose at time t we have $x_1(t) \leq x_2(t) \leq \ldots \leq x_n(t)$. We call the $n-1$ terms of the form $|x_i(t) - x_{i+1}(t)|$, $i \in \{1, 2, \ldots, n-1\}$, *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:

$$P(t) = \sum_{i=1}^{n-1} (n-i)i\,|x_i(t) - x_{i+1}(t)|. \tag{8}$$

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 Fig. 2, the segments are $|x_3 - x_2|$, $|x_2 - x_1|$, and $|x_1 - x_4|$, as shown in Fig. 3. 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 (5)). Referring to Fig. 3, 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

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

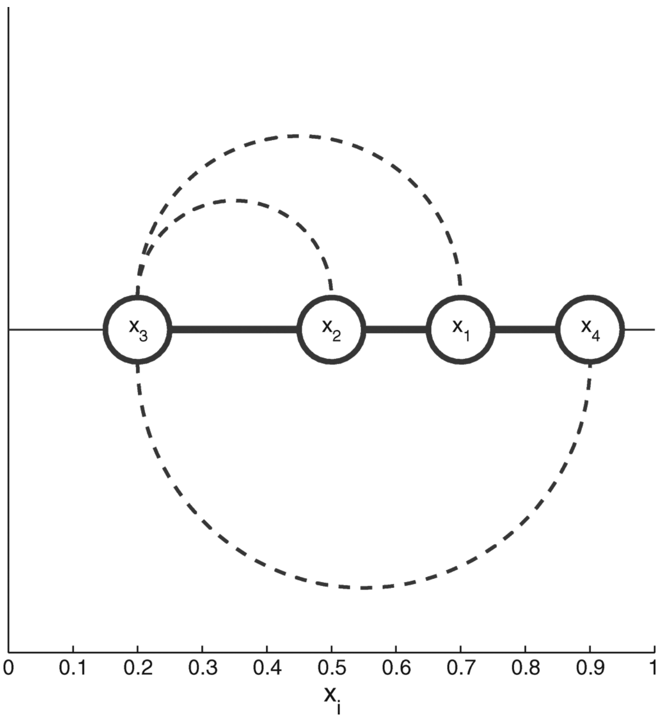


Fig. 3. The four node network embedded on the real line according to node value x_i. The bold lines indicate *segments*, i.e. intervals on the real line separating two adjacent values. The dashed curves indicate the communication topology from Fig. 2. Thus, an update on the link between node 1 and node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

It follows that

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

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

Proof: (of Theorem 1): Repeatedly applying Lemma 2, we see that

$$\lim_{t\to\infty} P(t) = 0. \tag{9}$$

Therefore

$$\lim_{t\to\infty} |x_i(t) - x_j(t)| = 0 \quad \forall i, j. \tag{10}$$

Now by the conservation property (which can be derived from (2))

$$\sum_{i=1}^{n} x_i(t) = \sum_{i=1}^{n} z_i \quad \forall t \tag{11}$$

we see that

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n}\sum_{i=1}^{n} z_i. \tag{12}$$

■

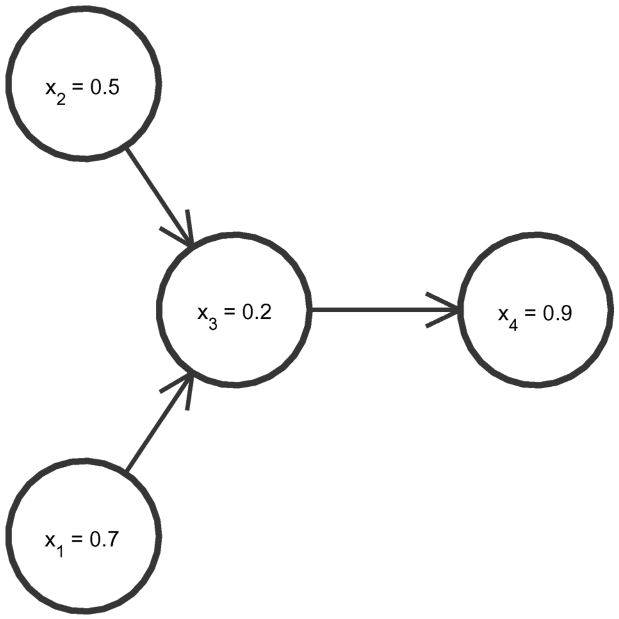


Fig. 4. The graph H for the example network, where the node indices are taken as the UIDs.

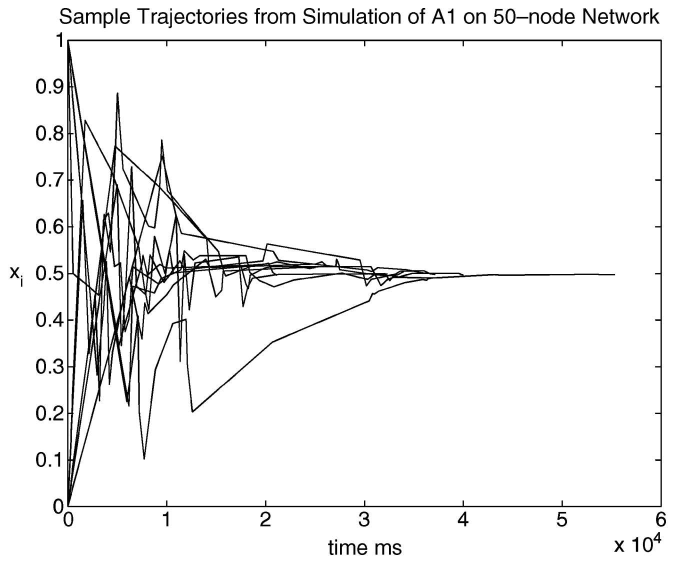


Fig. 5. State histories from a simulation of algorithm A1 on a fifty-node network. Round-trip delays on each link were assigned randomly, between 40 (ms) and 1000 (ms). Note that all states converge towards the average value 0.5.

VI. 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 a unique identifier (UID) for each node in the network. The UIDs must be orderable between different nodes. The UIDs can be obtained, for example, by mapping the IP addresses of the nodes to unique natural numbers. 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.

An example is illustrated for our four-node network in Fig. 4. This graph essentially defines a *sender-receiver relation on each link*.

Our proposed initiation scheme is as follows:

RR1: A node will wait to receive a STATE message from *all* of its inbound edges.

RR2: After having received at least one message from all its inbound edges, the node will then sequentially send a STATE message to 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 link ij, with i being the sender, and an interval $[t, \infty)$ during which this link does not carry any message. Then, node i must be waiting for one of its inbound edges to send, 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 send. 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* send to all of its outbound edges at some point in $[t, \infty)$. This is a contradiction, and proves the desired result.

VII. 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 0.5 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 0.5. The results of this simulation are shown in Fig. 5.

VIII. Dynamic Topology: Joining and Leaving of Nodes

The A1 algorithm we have described is general enough to accommodate various extensions. In this section we discuss how to handle dynamic network membership (dynamic network topology), where new nodes can join the averaging network, and current nodes can decide to leave or fail gracefully.

On a peer-to-peer network, one may wish to apply the averaging scheme that allows nodes to join and leave at various points in time. A simple mechanism for doing so is for each node to maintain an additional variable associated with each neighbor, denoted by δ_{ij}, which accounts for all the changes made on behalf of that neighbor. This idea is described in [15].

Specifically, each time node i and j interact, the net change in i's state is added to the variable δ_{ij}. Then, if a node leaves the network, all its neighbors subtract δ_{ij} from their current states. It can be shown that this ensures the following conservation property: at any given time, the sum of the states x_i over any connected component of the network is precisely equal to the sum of the initial values z_i. Thus, after a topology change, the iterative algorithm again begins converging toward the appropriate average quantity over the new network. Note that this serves as a reactive mechanism for node failures, since the failing node's neighbors can detect its failure and compensate by subtracting the associated δ_{ij} terms.

One promising application of the averaging algorithm with dynamic membership, is counting the number of nodes on a peer-to-peer network. It is known that due to the transient nature of peer-to-peer nodes, it is often hard to obtain a good estimate of the total number of active nodes on the network. Suppose it can be ensured that one and only one node has set $z_i = 1$, while all others have set $z_i = 0$. If *all* nodes are completely identical, this coordination would be hard to achieve. This is possible, however, on peer-to-peer networks with a bootstrapping server, since the bootstrapping server can be the special node with $z_i = 1$. Then the averaging algorithm combined with the aforementioned dynamic membership handling, can track the average state of all active nodes $1/n$, the reciprocal of the number of nodes. Each node can therefore have a running estimate of how large the active network size is without any additional action from the bootstrapping server.

IX. 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.

In A2, each node i makes use of the additional variables δ_{ij} as described in Section VIII. As described earlier, if there is a link between nodes i and j, there will be variables δ_{ij} and δ_{ji} stored locally with node i and node j, respectively.

We will denote the set of all neighbors of node i to be N_i. The algorithm A2 is specified mathematically in terms of the x_i's and the δ_{ij}'s as follows in the synchronous environment:

$$\begin{cases} x_i(t+1) = x_i(t) + \gamma_i \left[\sum_{j \in N_i} \delta_{ij}(t) + z_i - x_i(t) \right] \\ \delta_{ij}(t+1) = \delta_{ij}(t) + \phi_{ij} \left[x_j(t) - x_i(t) \right] \end{cases} \tag{13}$$

where we introduce the additional parameters ϕ_{ij}, which are local stepsizes similar to γ_i.

Algorithmically, the above update rules require additional specifications. First of all, each x_i is initialized to z_i as in algorithm 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. (We will see that one can also just set all ϕ's on the network to some constant value.)

Second, in order to guarantee convergence to the correct average, we require the following messaging rules. On each link ij, we impose a *sender-receiver* relationship on the variables δ_{ij} and δ_{ji}. One can use UIDs to obtain this, as described in Section VI.

MR1: Every node i sends to every neighbor a STATE message that contains its current state value x_i from time to time. Each node also, from time to time, executes the update rule (first equation in (13)) with the information it has about other state values.

MR2: On link ij, if δ_{ij} is the sender, it executes the update rule (second equation in (13)) from time to time. Whenever δ_{ij} executes the update rule, it also sends to its receiver δ_{ji} a REPLY message that contains the value of the *c*hange it has made in the value of δ_{ij}. δ_{ij} will not execute the update rule again until the TCP ACK of this REPLY message comes back.

MR3: If δ_{ji} is the receiver on link ij, it waits for REPLY messages from δ_{ij} and subtracts the value in the message from the value of δ_{ji}. (Note that the REPLY message does not directly change the value of x_j.)

Notice that the second equation in (13) is general enough to cover the execution required in MR3. Also, since the δ_{ij} variables are employed, A2 is automatically equipped with the ability to handle dynamic topologies (as discussed in Section VIII). All node i needs to do is to reset δ_{ij} to 0 if node j leaves the system.

We can now obtain the following property for δ_{ij}, which is important for A2's convergence to the correct average.

Lemma 4: For any pair of variables δ_{ij} and δ_{ji}, at any time t, there exists a later time $t' > t$ such that

$$\delta_{ij}(t') + \delta_{ji}(t') = 0. \tag{14}$$

Proof: Initially all δ_{ij}'s are set to 0. According to the messaging rules MR2 and MR3, if δ_{ij} executes the update rule (second equation in (13)) and changes its value by some amount at time t, the opposite change will be made by δ_{ji} at some later time t'. Therefore, their sum becomes 0 again. ■

If we consider the vector consisting of all the values of x_i and δ_{ij}, (13) can be thought of as an affine mapping on the vector of states. We will show that the 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 (2). A2 does not impose any blocking constraint and thus it does not have this property for simple analysis. In order to prove convergence of A2, we will make use of a general and powerful framework of asynchronous computation in [1].

We enumerate all these message-passing events in the set $T = \{0, 1, 2, \ldots\}$, 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 (13) become

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

where $0 \le \tau_i^{ij}, \tau_j^{ij}, \tau_{ij}^i \le t$ 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

$$\tau_i^{ij}(t) \ge t_1, \tau_{ij}^i(t) \ge t_1 \quad \forall i,j, \text{ and } t \ge t_2. \tag{15}$$

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:

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

under A2 with total asynchronism, provided the stepsizes satisfy

$$\begin{cases} 0 < \gamma_i < \frac{1}{d_i+1} \\ 0 < \phi_{ij} < \frac{1}{2}. \end{cases} \tag{16}$$

Proof: It can be shown that given the stepsize constraints, the synchronous equations are a *contraction mapping* with respect to the infinity norm. To illustrate this, consider a simple case with a two-node network. The synchronous update equations are

$$\begin{cases} x_1(t+1) = x_1(t) + \gamma_1\left[\delta_{12}(t) + z_1 - x_1(t)\right] \\ x_2(t+1) = x_2(t) + \gamma_2\left[\delta_{21}(t) + z_2 - x_2(t)\right] \\ \delta_{12}(t+1) = \delta_{12}(t) + \phi_{12}\left[x_2(t) - x_1(t)\right] \\ \delta_{21}(t+1) = \delta_{21}(t) + \phi_{21}\left[x_1(t) - x_2(t)\right]. \end{cases} \tag{17}$$

The linear part of the mapping is therefore

$$\begin{pmatrix} 1-\gamma_1 & 0 & \gamma_1 & 0 \\ 0 & 1-\gamma_2 & 0 & \gamma_2 \\ -\phi_{12} & \phi_{12} & 1 & 0 \\ \phi_{21} & -\phi_{21} & 0 & 1 \end{pmatrix}. \tag{18}$$

If the stepsize bounds (16) are satisfied, this linear mapping is strictly diagonally dominant (namely, the magnitude of every diagonal entry is strictly larger than the sum of the magnitudes of all the other entries in its row.) In the general case, the x_i row has a diagonal entry of $1-\gamma_i$, d_i entries of value γ_i, and entries of value 0 otherwise. The δ_{ij} row has a diagonal entry of 1, an entry of value ϕ_{ij}, an entry of value $-\phi_{ij}$, and 0 otherwise. Therefore, it can be seen that diagonal dominance is true in general, provided (16) is satisfied. Using Proposition 2.1 of Section 6.2 in [1], A2 converges under total asynchronism.

Now denote x_i^*, δ_{ij}^* to be the limit points of x_i and δ_{ij}. We see from the update equations that

$$\begin{cases} x_i^* = z_i + \sum_{j\in N_i} \delta_{ij}^* \quad \forall i, \\ x_i^* = x_j^* \quad \forall i,j. \end{cases}$$

Therefore

$$\begin{aligned} x_i^* &= \frac{1}{n}\sum_{i=1}^n x_i^* \\ &= \frac{1}{n}\sum_{i=1}^n z_i + \frac{1}{n}\sum_{i=1}^n \sum_{j\in N_i} \delta_{ij}^* \\ &= \frac{1}{n}\sum_{i=1}^n z_i \quad \forall i \end{aligned}$$

where in the last step we use Lemma 4 to cancel out all the δ_{ij}^* terms. ■

X. Convergence Rate

Analytical characterization of the convergence rate of our algorithms is difficult to obtain due to their general assumptions. In the case of algorithm A1, the potential function presented in Section V can serve as a metric for convergence rate. It is shown in the proof of theorem 1 that this potential function decreases exponentially in time.

For algorithm A2, however, we do not have any analytical results for the convergence rate in general. It is worth noting that the convergence rate of the synchronous algorithm (1) (as described in Section III) can be characterized analytically. It is shown in [11] that the convergence rate is related to the second smallest eigenvalue of the Laplacian matrix. This value is also known as the Fiedler eigenvalue or the algebraic connectivity of the graph. We have observed empirically that the convergence rate of the asynchronous algorithm A2 is similar to that of the synchronous algorithm (1) with the same average delays.

XI. Experimental Results

We developed an implementation of A2 in a C socket program and deployed it on the PlanetLab network [12]. 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 Fig. 6.

In this experiment, an overlay network of 100 nodes on Planetlab was chosen. Half of the nodes started with initial

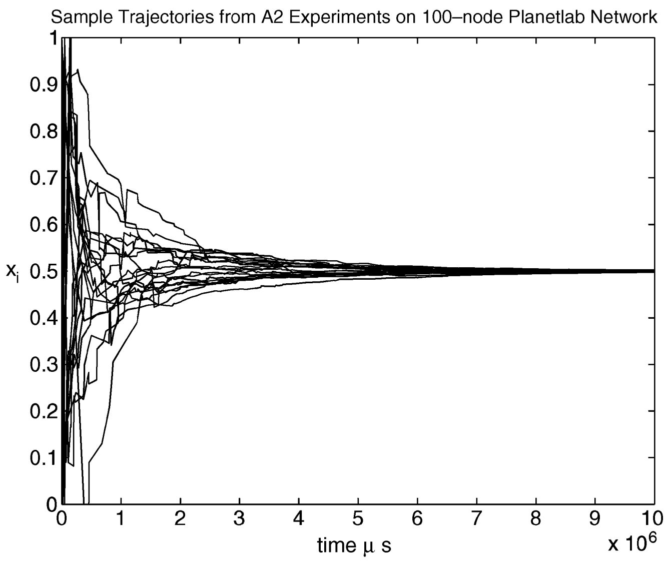


Fig. 6. Sample histories from an experiment on the PlanetLab network, using one-hundred nodes and algorithm A2. Round trip times on this network ranged between tens of milliseconds to approximately one second. Note the rapid convergence of the estimates.

value $z_i = 0$, and the other half with $z_i = 1$; therefore the target average value was 0.5. Each node was connected to five random neighbors in the overlay network. The round-trip delays (RTT) on this global overlay network ranged between tens of milliseconds to roughly half a second. It is worth noticing how rapid the convergence is: in 10 seconds all state values converged to within 1% of the average. We believe this is very promising in many applications.

XII. Simulation of A2

Experiments with more than 100 nodes were not feasible for us on the PlanetLab network. Therefore, we have written a discrete event simulator in Java and simulated algorithm A2 on larger network sizes.

The network sizes tested were: 50, 100, 250, 500, and 1000 nodes. For each size, the network topology was chosen such that each node connected to 5 randomly selected neighbors. The one-way delay on each link from node i to node j was chosen uniformly randomly between 20 ms and 500 ms. For each network size, there were five runs of simulation with the same connectivity but possibly different link delays in each run. All simulation started with half the nodes having initial states 0 and the other half having initial states 1.

In order to see the dependence of the convergence rate on the network size, we kept track of the amount of time it took for all node states to converge to within 1% of the target average. We will call this the "convergence time" for the sake of comparison. The convergence time (in units of seconds) obtained from our simulation for each run and each network size is reported in Table I.

The convergence time is therefore roughly constant regardless of the size of the network. Notice that this is consistent with our discussion on the convergence rate of the synchronous algorithm in Section X. The convergence rate in general does not depend on the network size but only on the algebraic connectivity or Fiedler eigenvalue of the topology. The simulation results suggest that our asynchronous algorithms still perform very well when the network size is large.

TABLE I
Convergence Time Versus Network Size

Network Size	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Fiedler Eigen-value
50	7.37	7.34	7.20	7.35	7.01	7.25	1.038
100	7.55	8.42	7.48	7.71	7.69	7.77	1.058
250	8.38	8.41	8.26	7.62	7.96	8.12	1.046
500	8.46	8.08	7.97	8.03	8.40	8.18	1.048
1000	7.81	8.29	8.60	8.39	8.10	8.24	1.020

XIII. Extensions

In addition to the "counting" application we have described in Section VIII, we will present a few more extensions of the averaging algorithms in this section.

So far we have assumed that the quantities being averaged (z_i) are static during the executions of our averaging algorithms. It turns out that one can readily adapt the algorithms to handle time-varying local variables, $z_i(t)$ (details can be found in [15]). All one need do is to modify the local state x_i each time z_i changes. Specifically, every time the local variable changes, according to

$$z_i \leftarrow z_i + \Delta z$$

then the local state is modified according to

$$x_i \leftarrow x_i + \Delta z.$$

This ensures that at any given time, the sum of the node states is equal to the sum of the local variables. Each time one of the z_i changes, the iterative algorithm adapts and converges to the appropriate value.

Instead of just calculating the average of node states, one can also readily adapt the algorithms to calculate the variance or any other moment of the distribution of node states. To get the variance, for example, one simply needs to run another averaging process on the quantities z_i^2, and the variance can be obtained.

XIV. Summary, Conclusion, and Future Work

We have presented a class of practically implementable distributed averaging algorithms that are suitable for communication networks such as the Internet. Our algorithms do not rely on synchronization, knowledge of the global topology, or coordination of parameter values.

Our analytical results for A1 show that under a mild timing assumption, the asynchronous message-passing algorithms can achieve exponential convergence. For the case when the blocking behavior of A1 is undesirable, we have introduced the algorithm A2, which is free of the blocking requirement and is also provably convergent under very general asynchronous timing. The iterative nature of the algorithm renders it robust to changes in topology.

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.

Acknowledgment

The first author would like to thank Prof. J. Tsitsiklis and his student A. Olshevsky for pointing out mistakes and offering useful suggestions.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis*, Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: Design, analysis and applications," in *Proc. IEEE INFOCOM*, 2005.

[3] B. Cohen, Incentives build robustness in Bittorrent. [Online]. Available: http://bitconjurer.org/BitTorrent/bittorrentecon.pdf

[4] A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Trans. Autom. Contr.*, vol. 49, pp. 1465–1476, Sep. 2004.

[5] KaZaA. [Online]. Available: http://www.kazaa.com

[6] D. Kempe, A. Dobra, and J. Gehrke, "Computing aggregate information using Gossip," in *Proc. FOCS*, 2003.

[7] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analaysis," in *Proc. STOC*, 2004.

[8] N. Lynch*, Distributed Algorithms*. San Mateo, CA: Morgan Kaufmann, 1997.

[9] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray, "Distributed averaging on asynchronous communication networks," in *Proc. IEEE Conf. Decision and Control*, 2005.

[10] R. Merris, "Laplacian matrices of a graph: A survey," *Linear Algebra and Its Applications*, vol. 197, pp. 143–176, 1994.

[11] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. Autom. Contr.*, vol. 49, no. 9, pp. 1520–1533, Sep. 2004.

[12] PlanetLab. [Online]. Available: http://www.planet-lab.org

[13] D. S. Scherber and H. C. Papadopoulos, "Distributed computation of averages over ad hoc networks," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 4, pp. 776–787, Apr. 2005.

[14] D. Spanos, R. Olfati-Saber, and R. M. Murray, "Distributed sensor fusion using dynamic consensus," in *Proc. IFAC*, 2005.

[15] D. P. Spanos, R. Olfati-Saber, and R. M. Murray, "Dynamic consensus on mobile networks," in *Proc. IFAC*, 2005.

[16] D. Spanos, R. Olfati-Saber, and R. M. Murray, "Distributed Kalman filtering in sensor networks with quantifiable performance," in *Proc. IPSN*, 2005.

[17] J. N. Tsitsiklis, "Problems in decentralized decision making and computation" Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, 1984 [Online]. Available: http://web.mit.edu/jnt/www/PhD-84-jnt.pdf

[18] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. Autom. Contr.*, vol. 31, no. 9, pp. 803–812, 1986.

[19] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," in *Proc. Conf. Decision and Control*, 2003.

[20] L. Xiao, S. Boyd, and S. Lall, "A scheme for asynchronous distributed sensor fusion based on average consensus," in *Proc. IPSN*, 2005.

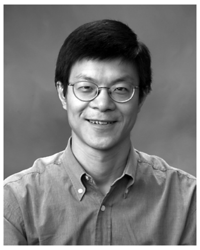
Mortada Mehyar received the B.S. degree in mathematics from National Taiwan University, Taipei, Taiwan, R.O.C., and the M.S. degree in applied physics from the California Institute of Technology, Pasadena. He is currently working toward the Ph.D. degree in electrical engineering at the California Institute of Technology.

His research focuses on control and optimization of communication networks.

Demetri Spanos received the B.A. degree in computational and applied mathematics and the B.S. degree in mechanical engineering from Rice University, Houston, TX. He is currently working toward the Ph.D. degree in control and dynamical systems at the California Institute of Technology, Pasadena.

His research focuses on control and dynamics of distributed and networked systems.

John Pongsajapan received the B.S. degree in computer science and mathematics from the University of California, Los Angeles. He is currently working toward the Ph.D. degree in computer science at the California Institute of Technology, Pasadena.



Steven H. Low (M'92–SM'99) received the B.S. degree from Cornell University, Ithaca, NY, and the Ph.D. degree from the University of California at Berkeley, all in electrical engineering.

He is an Associate Professor at Caltech, where he leads the FAST Project, and a Senior Fellow of the University of Melbourne, Australia. He was with AT&T Bell Laboratories, Murray Hill, NJ, from 1992 to 1996, and with the University of Melbourne from 1996 to 2000. His research interests are in the control and optimization of networks and protocols.

Dr. Low was a co-recipient of the IEEE William R. Bennett Prize Paper Award in 1997 and the 1996 R&D 100 Award. He is on the editorial boards of IEEE/ACM Transactions on Networking, *ACM Computing Surveys*, *Computer Networks Journal*, *NOW Foundations*, and *Trends in Networking*, and is a Senior Editor of the IEEE Journal on Selected Areas in Communications.

Richard M. Murray received the B.S. degree in electrical engineering from the California Institute of Technology, Pasadena, in 1985, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1988 and 1991, respectively.

He is currently a Professor of control and dynamical systems at the California Institute of Technology. His research is in the application of feedback and control to mechanical, information, and biological systems.