Distributed Anonymous Discrete Function Computation

Julien M. Hendrickx, Alex Olshevsky, and John N. Tsitsiklis, *Fellow, IEEE*

***Abstract*—We propose a model for deterministic distributed function computation by a network of identical and anonymous nodes. In this model, each node has bounded computation and storage capabilities that do not grow with the network size. Furthermore, each node only knows its neighbors, not the entire graph. Our goal is to characterize the class of functions that can be computed within this model. In our main result, we provide a necessary condition for computability which we show to be nearly sufficient, in the sense that every function that violates this condition can at least be approximated. The problem of computing (suitably rounded) averages in a distributed manner plays a central role in our development; we provide an algorithm that solves it in time that grows quadratically with the size of the network.**

***Index Terms*—Averaging algorithms, distributed computing, distributed control.**

I. Introduction

THE goal of many multi-agent systems, distributed computation algorithms, and decentralized data fusion methods is to have a set of nodes compute a common value based on initial values or observations at each node. Towards this purpose, the nodes, which we will sometimes refer to as agents, perform some internal computations and repeatedly communicate with each other. The objective of this paper is to understand the fundamental limitations and capabilities of such systems and algorithms when the available information and computational resources at each node are limited.

A. Motivation

The model that we will employ is a natural one for many different settings, including the case of wireless sensor networks. However, before describing the model, we start with a few examples that motivate the questions that we address.

Manuscript received April 01, 2010; revised March 31, 2011; accepted June 30, 2011. Date of publication August 08, 2011; date of current version October 05, 2011. This paper was presented in part at the Forty-Seventh Annual Allerton Conference on Communication, Control, and Computing, Oct.2009. This work was supported by the National Science Foundation under a graduate fellowship and grant ECCS-0701623, and by postdoctoral fellowships from the Belgian Fund for Scientific Research (F.R.S.-FNRS) and the Belgian American Education Foundation (B.A.E.F.), and was conducted while J. Hendrickx and A. Olshevsky were at M.I.T. Recommend by Associate Editor I. Paschalidis.

J.M. Hendrickx is with the Université catholique de Louvain, Louvain-la-Neuve B-1348, Belgium (e-mail: julien.hendrickx@uclouvain.be).

A. Olshevsky is with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: aolshevs@princeton.edu).

J. N. Tsitsiklis is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: jnt@mit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2011.2163874

(a) **Quantized consensus:** Suppose that each node begins with an integer value $x_i(0) \in \{0,\ldots,K\}$. We would like the nodes to end up, at some later time, with values y_i that are almost equal, i.e., $|y_i - y_j| \leq 1$, for all i, j, while preserving the sum of the values, i.e., $\sum_{i=1}^n x_i(0) = \sum_{i=1}^n y_i$. This is the so-called quantized averaging problem, which has received considerable attention recently; see, e.g., [3], [7], [13], [21], [25], [36]. It may be viewed as the problem of computing the function $(1/n)\sum_{i=1}^n x_i$, rounded to an integer value.

(b) **Distributed hypothesis testing** and **majority voting:** Consider n sensors interested in deciding between two hypotheses, H_0 and H_1. Each sensor collects measurements and makes a preliminary decision $x_i \in \{0,1\}$ in favor of one of the hypotheses. The sensors would like to make a final decision by majority vote, in which case they need to compute the indicator function of the event $\sum_{i=1}^n x_i \geq n/2$, in a distributed way. Alternatively, in a weighted majority vote, they may be interested in computing the indicator function of an event such as $\sum_{i=1}^n x_i \geq 3n/4$. A variation of this problem involves the possibility that some sensors abstain from the vote, perhaps due to their inability to gather sufficiently reliable information.

(c) **Direction coordination on a ring:** Consider n vehicles placed on a ring, each with some arbitrarily chosen direction of motion (clockwise or counterclockwise). We would like the n vehicles to agree on a single direction of motion. A variation of this problem was considered in [30], where, however, additional requirements on the vehicles were imposed which we do not consider here. The solution provided in [30] was semi-centralized in the sense that vehicles had unique numerical identifiers, and the final direction of most vehicles was set to the direction of the vehicle with the largest identifier. We wonder whether the direction coordination problem can be solved in a completely decentralized way. Furthermore, we would like the final direction of motion to correspond to the initial direction of the majority of the vehicles: if, say, 90% of the vehicles are moving counterclockwise, we would like the other 10% to turn around. If we define x_i to be 1 when the ith vehicle is initially oriented clockwise, and 0 if it is oriented counterclockwise, then, coordinating on a direction involves the distributed computation of the indicator function of the event $\sum_{i=1}^n x_i \geq n/2$.

(d) **Solitude verification:** This is the problem of checking whether exactly one node has a given state. This problem is of interest if we want to avoid simultaneous transmissions over a common channel [17], or if we want to main-

0018-9286/$26.00 © 2011 IEEE

tain a single leader (as in motion coordination—see for example [20]) Given states $x_i \in \{0, 1, \ldots, K\}$, solitude verification is equivalent to the problem of computing the binary function which is equal to 1 if and only if $|\{i : x_i = 0\}| = 1$.

There are numerous methods that have been proposed for solving problems such as the above; see for example the vast and growing literature on consensus and averaging methods, or the distribute robotics literature [9]. Oftentimes, different algorithms involve different computational capabilities on the part of the nodes, which makes it hard to talk about a "best" algorithm. At the same time, simple algorithms (such as setting up a spanning tree and aggregating information by progressive summations over the tree) are often considered undesirable because they require too much coordination or global information. It should be clear that a sound discussion of such issues requires the specification of a precise model of computation, followed by a systematic analysis of fundamental limitations under a given model. This is precisely the objective of this paper: to propose a particular model, and to characterize the class of functions computable under this model.

B. The Features of Our Model

Our model provides an abstraction for common requirements for distributed algorithms in the wireless sensor network literature. We model the nodes as interacting deterministic finite automata that exchange messages on a fixed bidirectional network, with no time delays or unreliable transmissions. Some important qualitative features of our model are the following.

Identical nodes: Any two nodes with the same number of neighbors must run the same algorithm. Note that this assumption is equivalent to assuming that the nodes are exactly identical. Any algorithm that works in this setting will also work if the nodes are not all identical, since the nodes can still run the same algorithm.

Anonymity: A node can distinguish its neighbors using its own, private, local identifiers. However, nodes do not have global identifiers. In other words, a node receiving a message from one of its neighbors can send an answer to precisely that neighbor, or recognize that a later message comes from this same neighbor. On the other hand, nodes do not *a priori* have a unique signature that can be recognized by every other node.

Determinism: Randomization is not allowed. This restriction is imposed in order to preclude essentially centralized solutions that rely on randomly generated distinct identifiers and thus bypass the anonymity requirement. Clearly, developing algorithms is much harder, and sometimes impossible, when randomization is disallowed.

Limited memory: We focus on the case where the nodes can be described by finite automata, and pay special attention to the required memory size. Ideally, the number of memory bits required at each node should be bounded above by a slowly growing function of the degree of a node.

Absence of global information: Nodes have no global information, and do not even have an upper bound on the total number of nodes. Accordingly, the algorithm that each node is running is independent of the network size and topology.

Convergence requirements: Nodes hold an estimated output that must converge to a desired value which is a function of all nodes' initial observations or values. In particular, for the case of discrete outputs, all nodes must eventually settle on the desired value. On the other hand, the nodes do not need to become aware of such termination, which is anyway impossible in the absence of any global information [6].

In this paper, we only consider the special case of **fixed graph topologies**, where the underlying (and unknown) interconnection graph does not change with time. Developing a meaningful model for the time-varying case and extending our algorithms to that case is an interesting topic, but outside the scope of this paper.

C. Literature Review

There is a very large literature on distributed function computation in related models of computation [5], [8], [26]. This literature can be broadly divided into two strands, although the separation is not sharp: works that address general computability issues for various models, and works that focus on the computation of specific functions, such as the majority function or the average. We start by discussing the first strand.

A common model in the distributed computing literature involves the requirement that all processes terminate once the desired output is produced and that nodes become aware that termination has occurred. A consequence of the termination requirement is that nodes typically need to know the network size n (or an upper bound on n) to compute non-trivial functions. We refer the reader to [1], [4], [6], [22], [32], [37] for some fundamental results in this setting, and to [14] for a comprehensive summary of known results. Closest to our work is [11] which provides an impossibility result very similar to our Theorem III.1, for a closely related model computation.

The biologically-inspired "population algorithm" model of distributed computation has some features in common with our model, namely, anonymous, bounded-resource nodes, and no requirement of termination awareness; see [2] for an overview of available results. However, this model involves a different type of node interactions from the ones we consider; in particular, nodes interact pairwise at times that may be chosen adversarially.

Regarding the computation of specific functions, [27] shows the impossibility of majority voting if the nodes are limited to a binary state. Some experimental memoryless algorithms (which are not guaranteed to always converge to the correct answer) have been proposed in the physics literature [16]. Several papers have quantified the performance of simple heuristics for computing specific functions, typically in randomized settings. We refer the reader to [19], which studied simple heuristics for computing the majority function, and to [35], which provides a heuristic that has guarantees only for the case of complete graphs.

The large literature on quantized averaging often tends to involve themes similar to those addressed in this paper [3], [10], [13], [21], [25], [34]. However, the underlying models of computation are typically more powerful than ours, as they allow for randomization and unbounded memory. Closer to the current paper, [33] develops an algorithm with $O(n^2)$ convergence

time for a variant of the quantized averaging problem, but requires unbounded memory. Reference [7] provides an algorithm for the particular quantized averaging problem that we consider in Section IV (called in [7] the "interval consensus problem"), which uses randomization but only bounded memory (a total of two bits at each node). An upper bound on its expected convergence time is provided in [12] as a function of n and a spectral quantity related to the network. A precise convergence time bound, as a function of n, is not given. Similarly, the algorithm in [38] runs in $O(n^5)$ time for the case of fixed graphs. (However, we note that [38] also addresses an asynchronous model involving time-varying graphs.) Roughly speaking, the algorithms in [7], [38] work by having positive and negative "load tokens" circulate randomly in the network until they meet and annihilate each other. Our algorithm involves a similar idea. However, at the cost of some algorithmic complexity, our algorithm is deterministic. This allows for fast progress, in contrast to the slow progress of algorithms that need to wait until the coalescence time of two independent random walks. Finally, a deterministic algorithm for computing the majority function (and some more general functions) was proposed in [29]. However, the algorithm appears to rely on the computation of shortest path lengths, and thus requires unbounded memory at each node.

Semi-centralized versions of the problem, in which the nodes ultimately transmit to a fusion center, have often been considered in the literature, e.g., for distributed statistical inference [31] or detection [24]. The papers [15], [23], and [28] consider the complexity of computing a function and communicating its value to a sink node. We refer the reader to the references therein for an overview of existing results in such semi-centralized settings. However, the underlying model is fundamentally different from ours, because the presence of a fusion center violates our anonymity assumption.

Broadly speaking, our results differ from previous works in several key respects: (i) Our model, which involves totally decentralized computation, deterministic algorithms, and constraints on memory and computation resources at the nodes, but does not require the nodes to know when the computation is over, is different from that considered in almost all of the relevant literature. (ii) Our focus is on identifying computable and non-computable functions under our model, and we achieve a nearly tight separation, as evidenced by a comparison between Theorem III.1 and Corollary IV.3. (iii) Our $O(n^2)$ averaging algorithm is quite different, and significantly faster than available memory-limited algorithms.

D. Summary and Contributions

We provide a general model of decentralized anonymous computation on fixed graphs, with the features described in Section I-B, and characterize the type of functions of the initial values that can be computed.

We prove that if a function is computable under our model, then its value can only depend on the frequencies of the different possible initial values. For example, if the initial values x_i are binary, a computable function can only depend on $p_0 := |\{i : x_i = 0\}|/n$ and $p_1 := |\{i : x_i = 1\}|/n$. In particular, determining the number of nodes, or whether at least two nodes have an initial value of 1, is impossible.

Conversely, we prove that if a function only depends on the frequencies of the different possible initial values (and is measurable), then the function can be approximated with any given precision, except possibly on a set of frequency vectors of arbitrarily small volume. Moreover, if the dependence on these frequencies can be expressed through a combination of linear inequalities with rational coefficients, then the function is computable exactly. In particular, the functions involved in the quantized consensus, distributed hypothesis testing, and direction coordination examples are computable, whereas the function involved in solitude verification is not. Similarly, statistical measures such as the standard deviation of the distribution of the initial values can be approximated with arbitrary precision. Finally, we show that with infinite memory, the frequencies of the different initial values (i.e., p_0, p_1 in the binary case) are computable exactly, thus obtaining a precise characterization of the computable functions in this case.

The key to our positive results is a new algorithm for calculating the (suitably quantized) average of the initial values, which is of independent interest. The algorithm does not involve randomization, requires only $O(n^2)$ time to terminate, and the memory (number of bits) required at each node is only logarithmic in the node's degree. In contrast, existing algorithms either require unbounded memory, or are significantly slower to converge.

E. Outline

In Section II, we describe formally our model of computation. In Section III, we establish necessary conditions for a function to be computable. In Section IV, we provide sufficient conditions for a function to be computable or approximable. Our positive results rely on an algorithm that keeps track of nodes with maximal values, and an algorithm that calculates a suitably rounded average of the nodes' initial values; these are described in Sections Sections V and VI, respectively. We provide some corroborating simulations in Section VII, and we end with some concluding remarks, in Section VIII.

II. Formal Description of the Model

Under our model, a distributed computing system consists of three elements:

(a) **A network:** A network is a triple $(n, G, \mathcal{L})$, where n is the number of nodes, and $G = (V, E)$ is a *connected bidirectional* graph $G = (V, E)$ with n nodes. (By bidirectional, we mean that the graph is directed but if $(i, j) \in E$, then $(j, i) \in E$.) We define $d(i)$ as the in-degree (and also out-degree, hence "degree" for short) of node i. Finally, $\mathcal{L}$ is a *port labeling* which assigns a *port number* (a distinct integer in the set $\{0, 1, \ldots, d(i)\}$) to each outgoing edge of any node i. Note that the unique identifiers i used to refer to nodes are only introduced for the purpose of analysis, and are not part of the actual system. In particular, nodes do not know and cannot use their identifiers.

(b) **Input and Output Sets:** The input set is a finite set $X = \{0, 1, \ldots, K\}$ to which the initial value of each node belongs. The output set is a finite set Y to which the output of each node belongs.

(c) **An algorithm:** An algorithm is defined as a family of finite automata $(A_d)_{d=1,2,\ldots}$, where the automaton A_d describes the behavior of a node with degree d. The state of the automaton A_d is a tuple $[x, z, y; (m_1, \ldots, m_d)]$; we will call $x \in X$ the *initial value*, $z \in Z_d$ the *internal memory state*, $y \in Y$ the *output* or *estimated answer*, and $m_1, \ldots, m_d \in M$ the outgoing *messages.* The sets Z_d and M are assumed finite. We allow the cardinality of Z_d to increase with d. Clearly, this would be necessary for any algorithm that needs to store the messages received in the previous time step. Each automaton A_d is identified with a transition law from $X \times Z_d \times Y \times M^d$ into itself, which maps each $[x, z, y; (m_1, \ldots, m_d)]$ to some $[x, z', y'; (m'_1, \ldots, m'_d)]$. In words, at each iteration, the automaton takes x, z, y, and incoming messages into account, to create a new memory state, output, and (outgoing) messages, but does not change the initial value.

Given the above elements of a distributed computing system, an algorithm proceeds as follows. For convenience, we assume that the above defined sets Y, Z_d, and M contain a special element, denoted by $\emptyset$. Each node i begins with an initial value $x_i \in X$ and implements the automaton $A_{d(i)}$, initialized with $x = x_i$ and $z = y = m_1 = \cdots = m_d = \emptyset$. We use $S_i(t) = [x_i, y_i(t), z_i(t), m_{i,1}(t), \ldots, m_{i,d(i)}(t)]$ to denote the state of node i's automaton at time t. Consider a particular node i. Let $j_1, \ldots, j_{d(i)}$ be an enumeration of its neighbors, according to the port numbers. (Thus, j_k is the node at the other end of the kth outgoing edge at node i.) Let p_k be the port number assigned to link (j_k, i) according to the port labeling at node j_k. At each time step, node i carries out the following update:

$$\begin{aligned}&\left[x_i, z_i(t+1), y_i(t+1); m_{i,1}(t+1), \ldots, m_{i,d(i)}(t+1)\right]\\ &\quad= A_{d(i)}\left[x_i, z_i(t), y_i(t); m_{j_1,p_1}(t), \ldots, m_{j_{d(i)},p_{d(i)}}(t)\right].\end{aligned}$$

In words, the messages $m_{j_k,p_k}(t)$, $k = 1, \ldots, d(i)$, "sent" by the neighbors of i into the ports leading to i are used to transition to a new state and create new messages $m_{i,k}(t+1)$, $k = 1, \ldots, d(i)$, that i "sends" to its neighbors at time $t+1$. We say that the algorithm *terminates* if there exists some $y^* \in Y$ (called the *final output* of the algorithm) and a time t' such that $y_i(t) = y^*$ for every i and $t \geq t'$.

Consider now a family of functions $(f_n)_{n=1,2,\ldots}$, where $f_n : X^n \to Y$. We say that such a family is *computable* if there exists a family of automata $(A_d)_{d=1,2,\ldots}$ such that for any n, for any network $(n, G, \mathcal{L})$, and any set of initial conditions $x_1, \ldots, x_n$, the resulting algorithm terminates and the final output is $f_n(x_1, \ldots, x_n)$. Intuitively, a family of function s is computable if there is a bounded-memory algorithm which allows the nodes to "eventually" learn the value of the function in any connected topology and for any initial conditions.

As an exception to the above definitions, we note that although we primarily focus on the finite case, we will briefly consider in Section IV function families $(f_n)_{n=1,2,\ldots}$ *computable with infinite memory*, by which we mean that the internal memory sets Z_d and the output set Y are countably infinite, the rest of the model remaining the same.

The rest of the paper focuses on the following general question: what families of functions are computable, and how can we design a corresponding algorithm $(A_d)_{d=1,2,\ldots}$? To illustrate the nature of our model and the type of algorithms that it supports, we provide a simple example.

1) Detection Problem: In this problem, all nodes start with a binary initial value $x_i \in \{0,1\} = X$. We wish to detect whether at least one node has an initial value equal to 1. We are thus dealing with the function family $(f_n)_{n=1,2,\ldots}$, where $f_n(x_1, \ldots, x_n) = \max\{x_1, \ldots, x_n\}$. This function family is computable by a family of automata with binary messages, binary internal state, and with the following transition rule:

if $x_i = 1$ or $z_i(t) = 1$ or $\max_{j:(i,j)\in E} m_{ji}(t) = 1$ **then**

  set $z_i(t+1) = y_i(t+1) = 1$

  send $m_{ij}(t+1) = 1$ to every neighbor j of i

else

  set $z_i(t+1) = y_i(t+1) = 0$

  send $m_{ij}(t+1) = 0$ to every neighbor j of i

end if

In the above algorithm, we initialize by setting $m_{ij}(0)$, $y_i(0)$, and $z_i(0)$ to zero instead of the special symbol $\emptyset$. One can easily verify that if $x_i = 0$ for every i, then $y_i(t) = 0$ for all i and t. If on the other hand $x_k = 1$ for some k, then at each time step t, those nodes i at distance less than t from k will have $y_i(t) = 1$. Thus, for connected graphs, the algorithm will terminate within n steps, with the correct output. It is important to note, however, that because n is unknown, a node i can never know whether its current output $y_i(t)$ is the final one. In particular, if $y_i(t) = 0$, node i cannot exclude the possibility that $x_k = 1$ for some node whose distance from i is larger than t.

III. Necessary Condition for Computability

In this section we establish our main negative result, namely, that if a function family is computable, then the final output can only depend on the frequencies of the different possible initial values. Furthermore, this remains true even if we allow for infinite memory, or restrict to networks in which neighboring nodes share a common label for the edges that join them. This result is quite similar to Theorem 3 of [11], and so is the proof. Nevertheless, we provide a proof in order to keep the paper self-contained.

We first need some definitions. Recall that $X = \{0, 1, \ldots, K\}$. We let D be the *unit simplex*, that is, $D = \{(p_0, \ldots, p_K) \in [0,1]^{K+1} : \sum_{k=0}^{K} p_k = 1\}$. We say that a function $h : D \to Y$ *corresponds* to a function family $(f_n)_{n=1,2,\ldots}$ if for every n and every $x \in X^n$, we have

$$\begin{aligned}&f(x_1, \ldots, x_n)\\ &\quad= h\left(p_0(x_1, \ldots, x_n), p_1(x_1, \ldots, x_n), \ldots, p_K(x_1, \ldots, x_n)\right)\end{aligned}$$

where

$$p_k(x_1, \ldots, x_n) = \frac{|\{i \mid x_i = k\}|}{n}$$

so that $p_k(x_1, \ldots, x_n)$ is the frequency of occurrence of the initial value k. In this case, we say that the family (f_n) is *frequency-based.* Note that n is used in defining the notion of frequency, but its value is unknown to the agents, and cannot be used in the computations.

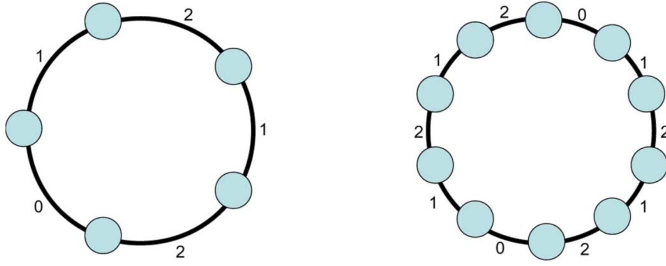

Fig. 1. Example of two situations that are algorithmically indistinguishable. The numbers next to each edge are the edge labels.

Theorem III.1: Suppose that the family (f_n) is computable with infinite memory. Then, this family is frequency-based. The result remains true even if we only require computability over edge-labeled networks.

The following are some applications of Theorem III.1.

(a) The parity function $\sum_{i=1}^{n} x_i \pmod{k}$ is not computable, for any $k > 1$.

(b) In a binary setting ($X = \{0, 1\}$), checking whether the number of nodes with $x_i = 1$ is larger than or equal to the number of nodes with $x_i = 0$ plus 10 is not computable.

(c) Solitude verification, i.e., checking whether $|i : \{x_i = 0\}| = 1$, is not computable.

(d) An aggregate difference function such as $\sum_{i<j} |x_i - x_j|$ is not computable, even if it is to be calculated modulo k.

Proof of Theorem III.1

The proof of Theorem III.1 involves a particular degree-two network (a ring), in which all port numbers take values in the set $\{0, 1, 2\}$, and in which any two edges (i, j) and (j, i) have the same port number, as represented in Fig. 1. More precisely, it relies on showing that two rings obtained by repeating, respectively, k and k' times the same sequences of nodes, edges, and initial conditions are algorithmically indistinguishable, and that any computable family of functions must thus take the same value on two such rings. The proof proceeds through a sequence of intermediate results, starting with the following lemma, which essentially reflects the fact that the node identifiers used in our analysis do not influence the course of the algorithm. It can be easily proved by induction on time, and its proof is omitted. The second lemma states that the value taken by computable (families of) functions may not depend on which particular node has which initial condition.

Lemma III.1: Suppose that $G = (\{1, \ldots, n\}, E)$ and $G' = (\{1, \ldots, n\}, E')$ are isomorphic; that is, there exists a permutation π such that $(i, j) \in E$ if and only if $(\pi(i), \pi(j)) \in E'$. Furthermore, suppose that the port label at node i for the edge leading to j in G is the same as the port label at node $\pi(i)$ for the edge leading to $\pi(j)$ in G'. Then, the state $S_i(t)$ resulting from the initial values $x_1, \ldots, x_n$ on the graph G is the same as the state $S_{\pi(i)}(t)$ resulting from the initial values $x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)}$ on the graph G'.

Lemma III.2: Suppose that the family $(f_n)_{n=1,2,\ldots}$ is computable with infinite memory on edge-labeled networks. Then, each f_i is invariant under permutations of its arguments.

Proof: Let π_{ij} be the permutation that swaps i with j (leaving the other nodes intact); with a slight abuse of notation, we also denote by π_{ij} the mapping from X^n to X^n that swaps the ith and jth elements of a vector. (Note that $\pi_{ij}^{-1} = \pi_{ij}$.) We show that for all $x \in X^n$, $f_n(x) = f_n(\pi_{ij}(x))$.

We run our distributed algorithm on the n-node complete graph with an edge labeling. Note that at least one edge labeling for the complete graph exists: for example, nodes i and j can use port number $(i + j) \bmod n$ for the edge connecting them. Consider two different sets of initial values, namely the vectors (i) x, and (ii) $\pi_{ij}(x)$. Let the port labeling in case (i) be arbitrary; in case (ii), let the port labeling be such that the conditions in Lemma III.1 are satisfied (which is easily accomplished). Since the final value is $f(x)$ in case (i) and $f(\pi_{ij}(x))$ in case (ii), we obtain $f(x) = f(\pi_{ij}(x))$. Since the permutations π_{ij} generate the group of permutations, permutation invariance follows. ■

Let $x \in X^n$. We will denote by x^2 the concatenation of x with itself, and, generally, by x^k the concatenation of k copies of x. We now prove that self-concatenation does not affect the value of a computable family of functions.

Lemma III.3: Suppose that the family $(f_n)_{n=1,2,\ldots}$ is computable with infinite memory on edge-labeled networks. Then, for every $n \geq 2$, every sequence $x \in X^n$, and every positive integer m

$$f_n(x) = f_{mn}(x^m).$$

Proof: Consider a ring of n nodes, where the ith node clockwise begins with the ith element of x; and consider a ring of mn nodes, where the nodes $i, i+n, i+2n, \ldots$ (clockwise) begin with the ith element of x. Suppose that the labels in the first ring are $0, 1, 2, 1, 2, \ldots$. That is, the label of the edge (1,2) is 0 at both nodes 1 and 2; the label of a subsequent edge $(i, i+1)$ is the same at both nodes i and $i+1$, and alternates between 1 and 2 as i increases. In the second ring, we simply repeat m times the labels in the first ring. See Fig. 1 for an example with $n = 5$, $m = 2$.

Initially, the state $S_i(t) = [x_i, y_i(t), z_i(t), m_{i,1}(t), m_{i,2}(t)]$, with $t = 0$, of node i in the first ring is exactly the same as the state of the nodes $j = i, i+n, i+2n, \ldots$ in the second ring. We show by induction that this property must hold at all times t. (To keep notation simple, we assume, without any real loss of generality, that $i \neq 1$ and $i \neq n$.)

Indeed, suppose this property holds up to time t. At time t, node i in the first ring receives a message from node $i-1$ and a message from node $i+1$; and in the second ring, node j satisfying $j \pmod{n} = i$ receives one message from $j-1$ and $j+1$. Since $j - 1 \pmod{n} = i - 1 \pmod{n}$ and $j + 1 \pmod{n} = i + 1 \pmod{n}$, the states of $j-1$ and $i-1$ are identical at time t, and similarly for $j+1$ and $i+1$. Thus, because of periodicity of the edge labels, nodes i (in the first ring) and j (in the second ring) receive identical messages through identically labeled ports at time t. Since i and j were in the same state at time t, they must be in the same state at time $t+1$. This proves that they are always in the same state. It follows that $y_i(t) = y_j(t)$ for all t, whenever $j \pmod{n} = i$, and therefore $f_n(x) = f_{mn}(x^m)$. ■

Proof of Theorem III.1: Let x and y be two sequences of n and m elements, respectively, such that $p_k(x_1, \ldots, x_n)$ and $p_k(y_1, \ldots, y_m)$ are equal to a common value $\hat{p}_k$, for $k \in X$; thus, the number of occurrences of k in x and y are $n\hat{p}_k$ and $m\hat{p}_k$, respectively. Observe that for any $k \in X$, the vectors x^m and y^n have the same number mn of elements, and both contain

$mn\hat{p}_k$ occurrences of k. The sequences y^n and x^m can thus be obtained from each other by a permutation, which by Lemma III.2 implies that $f_{nm}(x^m) = f_{nm}(y^n)$. From Lemma III.3, we have that $f_{nm}(x^m) = f_n(x)$ and $f_{mn}(y^n) = f_m(y)$. Therefore, $f_n(x) = f_m(y)$. This proves that the value of $f_n(x)$ is determined by the values of $p_k(x_1, \ldots, x_n)$, $k = 0, 1, \ldots, K$. ∎

Remark: Observe that the above proof remains valid even under the "symmetry" assumption that an edge is assigned the same label by both of the nodes that it is incident on.

IV. Reduction of Generic Functions to the Computation of Averages

In this section, we turn to positive results, aiming at a converse of Theorem III.1. The centerpiece of our development is Theorem IV.1, which states that a certain average-like function is computable. Theorem IV.2 then implies the computability of a large class of functions, yielding an approximate converse to Theorem III.1. We will then illustrate these positive results on some examples.

The average-like functions that we consider correspond to the "interval consensus" problem studied in [7]. They are defined as follows. Let $X = \{0, \ldots, K\}$. Let Y be the following set of single-point sets and intervals:

$$Y = \{\{0\}, (0,1), \{1\}, (1,2), \ldots, \{K-1\}, (K-1,K), \{K\}\}$$

(or equivalently, an indexing of this finite collection of intervals). For any n, let f_n be the function that maps $(x_1, x_2, \ldots, x_n)$ to the element of Y which contains the average $\sum_i x_i/n$. We refer to the function family $(f_n)_{n=1,2,\ldots}$ as the *interval-averaging* family. The output of this family of functions is thus the exact average when it is an integer; otherwise, it is the open interval between two integers that contains the average.

The motivation for this function family comes from the fact that the exact average $\sum_i x_i/n$ takes values in a countably infinite set, and cannot be computed when the set Y is finite. In the quantized averaging problem considered in the literature, one settles for an approximation of the average. However, such approximations do not necessarily define a single-valued function from X^n into Y. In contrast, the above defined function f_n is both single-valued and delivers an approximation with an error of size at most one. Note also that once the interval-average is computed, we can readily determine the value of the average rounded down to an integer.

Theorem IV.1: The interval-averaging function family is computable.

The proof of Theorem IV.1 (and the corresponding algorithm) is quite involved; it will be developed in Sections V and VI. In this section, we show that the computation of a broad class of functions can be reduced to interval-averaging.

Since only frequency-based function families can be computable (Theorem III.1), we can restrict attention to the corresponding functions h. We will say that a function h on the unit simplex D is *computable* if it corresponds to a frequency-based computable family (f_n). The level sets of h are defined as the sets $L(y) = \{p \in D \mid h(p) = y\}$, for $y \in Y$.

Theorem IV.2 (Sufficient Condition for Computability): Let h be a function from the unit simplex D to Y. Suppose that every level set $L(y)$ can be written as a finite union

$$L(y) = \bigcup_k C_{i,k}$$

where each $C_{i,k}$ can in turn be written as a finite intersection of linear inequalities of the form

$$\alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_K p_K \leq \alpha \qquad \text{(IV.1)}$$

or

$$\alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 + \cdots + \alpha_K p_K < \alpha$$

with rational coefficients $\alpha, \alpha_0, \alpha_1, \ldots, \alpha_K$. Then, h is computable.

Proof: Consider one such linear inequality, which we assume, for concreteness, to be of the form (IV.1). Let P be the set of indices k for which $\alpha_k \geq 0$. Since all coefficients are rational, we can clear their denominators and rewrite the inequality as

$$\sum_{k \in P} \beta_k p_k - \sum_{k \in P^c} \beta_k p_k \leq \beta \qquad \text{(IV.2)}$$

for nonnegative integers β_k and β. Let χ_k be the indicator function associated with initial value k, i.e., $\chi_k(i) = 1$ if $x_i = k$, and $\chi_k(i) = 0$ otherwise, so that $p_k = (1/n)\sum_i \chi_k(i)$. Then, (IV.2) becomes

$$\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k \in P} \beta_k \chi_k(i) + \sum_{k \in P^c} \beta_k (1 - \chi_k(i))\right) \leq \beta + \sum_{k \in P^c} \beta_k$$

or

$$\frac{1}{n}\sum_{i=1}^{n} q_i \leq q^*$$

where $q_i = \sum_{k \in P} \beta_k \chi_k(i) + \sum_{k \in P^c} \beta_k (1 - \chi_k(i))$ and $q^* = \beta + \sum_{k \in P^c} \beta_k$.

An algorithm that determines whether the last inequality is satisfied can be designed as follows. Knowing the parameters β_k and the set P, which can be made part of algorithm description as they depend on the problem and not on the data, each node can initially compute its value q_i, as well as the value of q^*. Nodes can then apply the distributed algorithm that computes the integer part of $(1/n)\sum_{i=1}^{n} q_i$; this is possible by virtue of Theorem IV.1, with K set to $\sum_k \beta_k$ (the largest possible value of q_i). It suffices then for them to constantly compare the current output of this algorithm to the integer q^*. To check any finite collection of inequalities, the nodes can perform the computations for each inequality in parallel.

To compute h, the nodes simply need to check which set $L(y)$ the frequencies $p_0, p_1, \ldots, p_K$ lie in, and this can be done by checking the inequalities defining each $L(y)$. All of these computations can be accomplished with finite automata: indeed, we do nothing more than run finitely many copies of the automata provided by Theorem IV.1, one for each inequality. The total memory used by the automata depends on the number of sets

$C_{i,k}$ and the magnitude of the coefficients β_k, but not on n, as required. ■

Theorem IV.2 shows the computability of functions h whose level-sets can be defined by linear inequalities with rational coefficients. On the other hand, it is clear that not every function h can be computable. (This can be shown by a counting argument: there are uncountably many possible functions h on the rational elements of D, but for the special case of bounded degree graphs, only countably many possible algorithms.) Still, the next result shows that the set of computable functions is rich enough, in the sense that computable functions can approximate any measurable function, everywhere except possibly on a low-volume set.

We will call a set of the form $\prod_{k=0}^{K}(a_k, b_k)$, with every a_k, b_k rational, *a rational open box*, where $\prod$ stands for Cartesian product. A function that can be written as a finite sum $\sum_i a_i 1_{B_i}$, where the B_i are rational open boxes and the 1_{B_i} are the associated indicator functions, will be referred to as a *box function*. Note that box functions are computable by Theorem IV.2.

Corollary IV.3: If every level set of a function $h : D \rightarrow Y$ on the unit simplex D is Lebesgue measurable, then, for every $\epsilon > 0$, there exists a computable box function $h_\epsilon : D \rightarrow Y$ such that the set $\{p \in D \mid h(p) \neq h_\epsilon(p)\}$ has measure at most ϵ.

Proof: (Outline) The proof relies on the following elementary result from measure theory. Given a Lebesgue measurable set $E \subseteq D$ and some $\epsilon > 0$, there exists a set E' which is a finite union of disjoint open boxes, and which satisfies

$$\mu(E \Delta E') < \epsilon$$

where μ is the Lebesgue measure and Δ is the symmetric difference operator. By a routine argument, these boxes can be taken to be rational. By applying this fact to the level sets of the function h (assumed measurable), the function h can be approximated by a box function h_ϵ. Since box functions are computable, the result follows. ■

The following corollary states that continuous functions are approximable.

Corollary IV.4: If a function $h : D \rightarrow [L, U] \subseteq \Re$ is continuous, then for every $\epsilon > 0$ there exists a computable function $h_\epsilon : D \rightarrow [L, U]$ such that $\|h - h_\epsilon\|_\infty < \epsilon$

Proof: Since D is compact, h is uniformly continuous. One can therefore partition D into a finite number of subsets, $A_1, A_2, \ldots, A_q$, that can be described by linear inequalities with rational coefficients, so that $\max_{p \in A_j} h(p) - \min_{p \in A_j} h(p) < \epsilon$ holds for all A_j. The function h_ϵ is then built by assigning to each A_j an appropriate value in $\{L, L+\epsilon, L+2\epsilon, \ldots, U\}$. ■

To illustrate these results, let us consider again some examples.

(a) Majority voting between two options is equivalent to checking whether $p_1 \leq 1/2$, with alphabet $\{0, 1\}$. This condition is clearly of the form (IV.1), and is therefore computable.

(b) Majority voting when some nodes can "abstain" amounts to checking whether $p_1 - p_0 \geq 0$, with input set $X = \{0, 1, \text{abstain}\}$. This function family is computable.

(c) We can ask for the second most popular value out of four, for example. In this case, the sets A_i can be decomposed into constituent sets defined by inequalities such as $p_2 \leq p_3 \leq p_4 \leq p_1$, each of which obviously has rational coefficients. The level sets of the function can thus clearly be expressed as unions of sets defined by a collection of linear inequalities of the type (IV.1), so that the function is computable.

(d) For any subsets I, I' of $\{0, 1, \ldots, K\}$, the indicator function of the set where $\sum_{i \in I} p_i > \sum_{i \in I'} p_i$ is computable. This is equivalent to checking whether more nodes have a value in I than do in I'.

(e) The indicator functions of the sets defined by $p_1^2 \leq 1/2$ and $p_1 \leq \pi/4$ are measurable, so they are approximable. We are unable to say whether they are computable.

(f) The indicator function of the set defined by $p_1 p_2 \leq 1/8$ is approximable, but we are unable to say whether it is computable.

Finally, we show that with infinite memory, it is possible to recover the exact frequencies p_k. (Note that this is impossible with finite memory, because n is unbounded, and the number of bits needed to represent p_k is also unbounded.) The main difficulty is that p_k is a rational number whose denominator can be arbitrarily large, depending on the unknown value of n. The idea is to run separate algorithms for each possible value of the denominator (which is possible with infinite memory), and reconcile their results.

Theorem IV.5: The vector $(p_0, p_1, \ldots, p_K)$ is computable with infinite memory.

Proof: We show that p_1 is computable exactly, which is sufficient to prove the theorem. Consider the following algorithm, to be referred to as Q_m, parametrized by a positive integer m. The input set X_m is $\{0, 1, \ldots, m\}$ and the output set Y_m is the same as in the interval-averaging problem: $Y_m = \{\{0\}, (0,1), \{1\}, (1,2), \{2\}, (2,3), \ldots, \{m-1\}, (m-1, m), \{m\}\}$. If $x_i = 1$, then node sets its initial value $x_{i,m}$ to m; else, the node sets its initial value $x_{i,m}$ to 0. The algorithm computes the function family (f_n) which maps X_m^n to the element of Y_m containing $(1/n)\sum_{i=1}^{n} x_{i,m}$, which is possible, by Theorem IV.1.

The nodes run the algorithms Q_m for every positive integer value of m, in an interleaved manner. Namely, at each time step, a node runs one step of a particular algorithm Q_m, according to the following order:

$$Q_1,\ Q_1, Q_2,\ Q_1, Q_2, Q_3,\ Q_1, Q_2, Q_3, Q_4,\ Q_1, Q_2, \ldots$$

At each time t, let $m_i(t)$ be the smallest m (if it exists) such that the output $y_{i,m}(t)$ of Q_m at node i is a singleton (not an interval). We identify this singleton with the numerical value of its single element, and we set $y_i(t) = y_{i,m_i(t)}(t)/m_i(t)$. If $m_i(t)$ is undefined, then $y_i(t)$ is set to some default value, e.g., $\emptyset$.

Let us fix a value of n. For any $m \leq n$, the definition of Q_m and Theorem IV.1 imply that there exists a time after which the outputs $y_{i,m}$ of Q_m do not change, and are equal to a common value, denoted y_m, for every i. Moreover, at least one of the algorithms $Q_1, \ldots, Q_n$ has an integer output y_m. Indeed, observe that Q_n computes $(1/n)\sum_{i=1}^{n} n 1_{x_i=1} = \sum_{i=1}^{n} 1_{x_i=1}$, which is clearly an integer. In particular, $m_i(t)$ is eventually well-defined and bounded above by n. We conclude that there exists a time

after which the output $y_i(t)$ of our overall algorithm is fixed, shared by all nodes, and different from the default value $\emptyset$.

We now argue that this value is indeed p_1. Let m^* be the smallest m for which the eventual output of Q_m is a single integer y_m. Note that y_{m^*} is the exact average of the x_{i,m^*}, i.e.

$$y_{m^*} = \frac{1}{n}\sum_{i=1}^{n} m^* 1_{x_i=1} = m^* p_1.$$

For large t, we have $m_i(t) = m^*$ and therefore $y_i(t) = y_{i,m^*}(t)/m^* = p_1$, as desired.

Finally, it remains to argue that the algorithm described here can be implemented with a sequence of infinite memory automata. All the above algorithm does is run a copy of all the automata implementing $Q_1, Q_2, \ldots$ with time-dependent transitions. This can be accomplished with an automaton whose state space is the countable set $\mathcal{N} \times \cup_{m=1}^{\infty} \prod_{i=1}^{m} \mathcal{Q}_i$, where $\mathcal{Q}_i$ is the state space of Q_i, and the set $\mathcal{N}$ of integers is used to keep track of time. ∎

V. Computing and Tracking Maximal Values

We now describe an algorithm that tracks the maximum (over all nodes) of time-varying inputs at each node. It will be used as a subroutine of the interval-averaging algorithm described in Section VI, and which is used to prove Theorem IV.1. The basic idea is the same as for the simple algorithm for the detection problem given in Section II: every node keeps track of the largest value it has heard so far, and forwards this "intermediate result" to its neighbors. However, when an input value changes, the existing intermediate results need to be invalidated, and this is done by sending "restart" messages. A complication arises because invalidated intermediate results might keep circulating in the network, always one step ahead of the restart messages. We deal with this difficulty by "slowing down" the intermediate results, so that they travel at half the speed of the restart messages. In this manner, restart messages are guaranteed to eventually catch up with and remove invalidated intermediate results.

We start by giving the specifications of the algorithm. Suppose that each node i has a time-varying input $u_i(t)$ stored in memory at time t, belonging to a finite set of numbers $\mathcal{U}$. We assume that, for each i, the sequence $u_i(t)$ must eventually stop changing, i.e., that there exists some T' such that

$$u_i(t) = u_i(T'), \quad \text{for all } i \text{ and } t \geq T'.$$

(However, node i need not ever be aware that $u_i(t)$ has reached its final value.) Our goal is to develop a distributed algorithm whose output eventually settles on the value $\max_i u_i(T')$. More precisely, each node i is to maintain a number $M_i(t)$ which must satisfy the following condition: for every network and any allowed sequences $u_i(t)$, there exists some T'' with

$$M_i(t) = \max_{j=1,\ldots,n} u_j(t), \qquad \text{for all } i \text{ and } t \geq T''.$$

Moreover, each node i must also maintain a pointer $P_i(t)$ to a neighbor or to itself. We will use the notation $P_i^2(t) = P_{P_i(t)}(t)$, $P_i^3(t) = P_{P_i^2(t)}(t)$, etc. We require the following additional property, for all t larger than T'': for each node i there exists a node j and a power K such that for all $k \geq K$ we have $P_i^k(t) = j$ and $M_i(t) = u_j(t)$. In words, by successively following the pointers $P_i(t)$, one can arrive at a node with a maximal value.

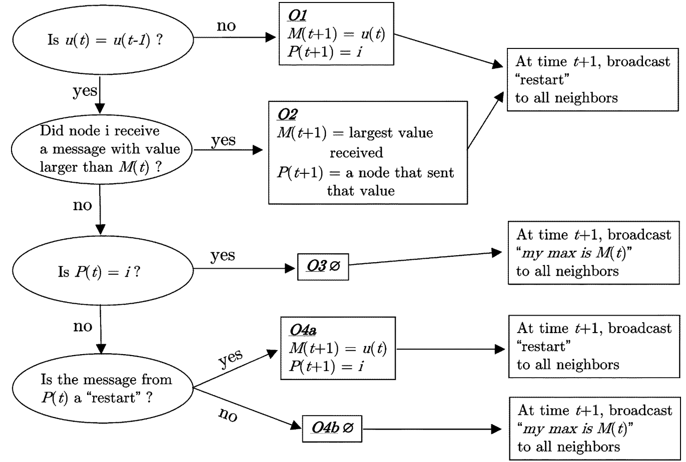

Fig. 2. Flowchart of the procedure used by node i during slot t in the maximum tracking algorithm. The subscript i is omitted, but $u(t)$, $M(t)$, and $P(t)$ should be understood as $u_i(t)$, $M_i(t)$, and $P_i(t)$. In those cases where an updated value of M or P is not indicated, it is assumed that $M(t+1) = M(t)$ and $P(t+1) = P(t)$. The symbol $\emptyset$ is used to indicate no action. Note that the various actions indicated are taken during slot t, but the messages determined by these actions are sent (and instantaneously received) at time $t+1$. Finally, observe that every node sends an identical message to all its neighbors at every time $t > 0$. We note that the apparent non-determinism in instruction O2 can be removed by picking a node with, say, the smallest port label.

We next describe the algorithm. We will use the term *slot* t to refer, loosely speaking, to the interval between times t and $t+1$. More precisely, during slot t each node processes the messages that have arrived at time t and computes the state at time $t+1$ as well as the messages it will send at time $t+1$.

The variables $M_i(t)$ and $P_i(t)$ are a complete description of the state of node i at time t. Our algorithm has only two types of messages that a node can send to its neighbors. Both are broadcasts, in the sense that the node sends them to every neighbor:

1) "Restart!"
2) "My estimate of the maximum is y," where y is some number in $\mathcal{U}$ chosen by the node.

Initially, each node sets $M_i(0) = u_i(0)$ and $P_i(0) = i$. At time $t = 0, 1, \ldots$, nodes exchange messages, which then determine their state at time $t+1$, i.e., the pair $M_i(t+1)$, $P_i(t+1)$, as well as the messages to be sent at time $t+1$. The procedure node i uses to do this is described in Fig. 2. One can verify that a memory size of $C \log |\mathcal{U}| + C \log d(i)$ at each node i suffices, where C is an absolute constant. (This is because M_i and P_i can take one of $|\mathcal{U}|$ and $d(i)$ possible values, respectively.)

The result that follows asserts the correctness of the algorithm. The idea of the proof is quite simple. Nodes maintain estimates $M_i(t)$ which track the largest among all the $u_i(t)$ in the graph; these estimates are "slowly" forwarded by the nodes to their neighbors, with many artificial delays along the way. Should some value $u_j(t)$ change, restart messages traveling without artificial delays are forwarded to any node which thought j had a maximal value, causing those nodes to start over. The possibility of cycling between restarts and forwards is avoided because restarts travel faster. Eventually, the variables $u_i(t)$ stop changing, and the algorithm settles on the correct answer. On the other hand a formal and rigorous exposition of this

simple argument is rather tedious. A formal proof is available in the technical report [18].

Theorem V.1 (Correctness of the Maximum Tracking Algorithm): Suppose that the $u_i(t)$ stop changing after some finite time. Then, for every network, there is a time after which the variables $P_i(t)$ and $M_i(t)$ stop changing and satisfy $M_i(t) = \max_i u_i(t)$; furthermore, after that time, and for every i, the node $j = P_i^n(t)$ satisfies $M_i(t) = u_j(t)$.

VI. Interval-Averaging

In this section, we present an interval-averaging algorithm and prove its correctness. We start with an informal discussion of the main idea. Imagine the integer input value x_i as represented by a number of x_i pebbles at node i. The algorithm attempts to exchange pebbles between nodes with unequal numbers so that the overall distribution becomes more even. Eventually, either all nodes will have the same number of pebbles, or some will have a certain number and others just one more. We let $u_i(t)$ be the current number of pebbles at node i; in particular, $u_i(0) = x_i$. An important property of the algorithm will be that the total number of pebbles is conserved.

To match nodes with unequal number of pebbles, we use the maximum tracking algorithm of Section V. Recall that the algorithm provides nodes with pointers which attempt to track the location of the maximal values. When a node with u_i pebbles comes to believe in this way that a node with at least $u_i + 2$ pebbles exists, it sends a request in the direction of the latter node to obtain one or more pebbles. This request follows a path to a node with a maximal number of pebbles until the request either gets denied, or gets accepted by a node with at least $u_i + 2$ pebbles.

A. The Algorithm

The algorithm uses two types of messages. Each type of message can be either *originated* at a node or *forwarded* by a node.

(a) (Request, r): This is a request for a transfer of pebbles. Here, r is an integer that represents the number of pebbles $u_i(t)$ at the node i that first originated the request, at the time t that the request was originated. (Note, however, that this request is actually sent at time $t + 1$.)

(b) (Accept, w): This corresponds to acceptance of a request, and a transfer of w pebbles towards the node that originated the request. An acceptance with a value $w = 0$ represents a request denial.

As part of the algorithm, the nodes run the maximum tracking algorithm of Section V, as well as a minimum tracking counterpart. In particular, each node i has access to the variables $M_i(t)$ and $P_i(t)$ of the maximum tracking algorithm (recall that these are, respectively, the estimated maximum and a pointer to a neighbor or to itself). Furthermore, each node maintains three additional variables.

(a) "$\mathrm{Mode}(t)$" $\in \{\mathrm{Free}, \mathrm{Blocked}\}$. Initially, the mode of every node is free. A node is blocked if it has originated or forwarded a request, and is still waiting to hear whether the request is accepted (or denied).

(b) "$\mathrm{Rin}_i(t)$" and "$\mathrm{Rout}_i(t)$" are pointers to a neighbor of i, or to i itself. The meaning of these pointers when in blocked mode are as follows. If $\mathrm{Rout}_i(t) = j$, then node i has sent (either originated or forwarded) a request to node j, and is still in blocked mode, waiting to hear whether the request is acceptd or denied. If $\mathrm{Rin}_i(t) = k$, and $k \neq i$, then node i has received a request from node k but has not yet responded to node k. If $\mathrm{Rin}_i(t) = i$, then node i has originated a request and is still in blocked mode, waiting to hear whether the request is accepted or denied.

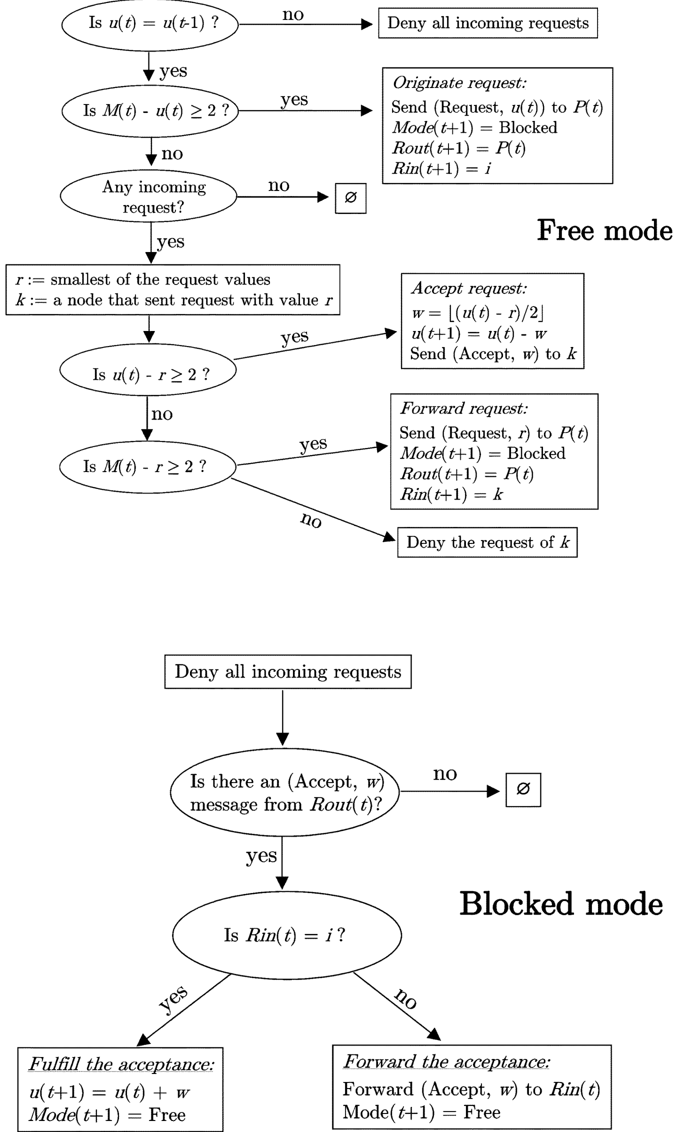

Fig. 3. Flowchart of the procedure used by node i during slot t in the interval-averaging algorithm. The subscript i is omitted from variables such as $\mathrm{Mode}(t)$, $M(t)$, etc. Variables for which an update is not explicitly indicated are assumed to remain unchanged. "Denying a request" is a shorthand for i sending a message of the form (Accept, 0) at time $t+1$ to a node from which i received a request at time t. Note also that "forward the acceptance" in the blocked mode includes the case where the answer had $w = 0$ (i.e., it was a request denial), in which case the denial is forwarded.

A precise description of the algorithm is given in Fig. 3. The proof of correctness is given in Section VI-B, thus also establishing Theorem IV.1. Furthermore, we will show that the time until the algorithm settles on the correct output is of order $O(n^2K^2)$.

Proof of Correctness

We begin by arguing that the rules of the algorithm preclude one potential obstacle; we will show that nodes will not get stuck sending requests to themselves.

Lemma VI.1: A node never sends (originates or forwards) a request to itself. More precisely, $\mathrm{Rout}_i(t) \neq i$, for all i and t.

Proof: By inspecting the first two cases for the free mode, we observe that if node i originates a request during time slot t (and sends a request message at time $t+1$), then $P_i(t) \neq i$. Indeed, to send a message, it must be true $M_i(t) > u_i(t) = u_i(t-1)$. However, any action of the maximum tracking algorithm that sets $P_i(t) = i$ also sets $M_i(t) = u_i(t-1)$, and moreover, as long as P_i does not change neither does M_i. So the recipient $P_i(t)$ of the request originated by i is different than i, and accordingly, $\mathrm{Rout}_i(t+1)$ is set to a value different than i. We argue that the same is true for the case where Rout_i is set by the "Forward request" box of the free mode. Indeed, that box is enabled only when $u_i(t) = u_i(t-1)$ and $u_i(t) - 1 \leq r < M_i(t) - 1$, so that $u_i(t-1) < M_i(t)$. As in the previous case, this implies that $P_i(t) \neq i$ and that $\mathrm{Rout}_i(t+1)$ is again set to a value other than i. We conclude that $\mathrm{Rout}_i(t) \neq i$ for all i and t. ∎

We will now analyze the evolution of the requests. A request is originated at some time τ by some originator node ℓ who sets $\mathrm{Rin}_\ell(\tau+1) = \ell$ and sends the request to some node $i = \mathrm{Rout}_\ell(\tau+1) = P_\ell(\tau)$. The recipient i of the request either accepts/denies it, in which case Rin_i remains unchanged, or forwards it while also setting $\mathrm{Rin}_i(\tau+2)$ to ℓ. The process then continues similarly. The end result is that at any given time t, a request initiated by node ℓ has resulted in a "request path of node ℓ at time t," which is a maximal sequence of nodes $\ell, i_1, \ldots, i_k$ with $\mathrm{Rin}_\ell(t) = \ell$, $\mathrm{Rin}_{i_1}(t) = \ell$, and $\mathrm{Rin}_{i_m}(t) = i_{m-1}$ for $m \leq k$.

Lemma VI.2: At any given time, different request paths cannot intersect (they involve disjoint sets of nodes). Furthermore, at any given time, a request path cannot visit the same node more than once.

Proof: For any time t, we form a graph that consists of all edges that lie on some request path. Once a node i is added to some request path, and as long as that request path includes i, node i remains in blocked mode and the value of Rin_i cannot change. This means that adding a new edge that points into i is impossible. This readily implies that cycles cannot be formed and also that two request paths cannot involve a common node. ∎

We use $p_\ell(t)$ to denote the request path of node ℓ at time t, and $s_\ell(t)$ to denote the last node on this path. We will say that a request originated by node ℓ *terminates* when node ℓ receives an (Accept, w) message, with any value w.

Lemma VI.3: Every request eventually terminates. Specifically, if node ℓ originates a request at time t' (and sends a request message at time $t'+1$), then there exists a later time $t'' \leq t'+n$ at which node $s_r(t'')$ receives an "accept request" message (perhaps with $w=0$), which is forwarded until it reaches ℓ, no later than time $t''+n$.

Proof: By the rules of our algorithm, node ℓ sends a request message to node $P_\ell(t')$ at time $t'+1$. If node $P_\ell(t')$ replies at time $t'+2$ with a "deny request" response to ℓ's request, then the claim is true; otherwise, observe that $p_\ell(t'+2)$ is nonempty and until $s_\ell(t)$ receives an "accept request" message, the length of $p_\ell(t)$ increases at each time step. Since this length cannot be larger than $n-1$, by Lemma VI.2, it follows that $s_\ell(t)$ receives an "accept request" message at most n steps after ℓ initiated the request. One can then easily show that this acceptance message is forwarded backwards along the path (and the request path keeps shrinking) until the acceptance message reaches ℓ, at most n steps later. ∎

The arguments so far had mostly to do with deadlock avoidance. The next lemma concerns the progress made by the algorithm. Recall that a central idea of the algorithm is to conserve the total number of "pebbles," but this must include both pebbles possessed by nodes and pebbles in transit. We capture the idea of "pebbles in transit" by defining a new variable. If i is the originator of some request path that is present at time t, and if the final node $s_i(t)$ of that path receives an (Accept, w) message at time t, we let $w_i(t)$ be the value w in that message. (This convention includes the special case where $w=0$, corresponding to a denial of the request). In all other cases, we set $w_i(t)=0$. Intuitively, $w_i(t)$ is the value that has already been given away by a node who answered a request originated by node i, and that will eventually be added to u_i, once the answer reaches i.

We now define

$$\hat{u}_i(t) = u_i(t) + w_i(t).$$

By the rules of our algorithm, if $w_i(t) = w > 0$, an amount w will eventually be added to u_i, once the acceptance message is forwarded back to i. The value $\hat{u}_i$ can thus be seen as a future value of u_i, that includes its present value and the value that has been sent to i but has not yet reached it.

The rules of our algorithm imply that the sum of the $\hat{u}_i$ remain constant. Let $\bar{x}$ be the average of the initial values x_i. Then

$$\frac{1}{n}\sum_{i=1}^{n} \hat{u}_i(t) = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}.$$

We define the *variance* function V as

$$V(t) = \sum_{i=1}^{n} (\hat{u}_i(t) - \bar{x})^2.$$

Lemma VI.4: The number of times that a node can send an acceptance message (Accept, w) with $w \neq 0$, is finite.

Proof: Let us first describe the idea behind the proof. Suppose that nodes could instantaneously transfer value to each other. It is easily checked that if a node i transfers an amount w^* to a node j with $u_i - u_j \geq 2$ and $1 \leq w^* \leq (1/2)(u_i - u_j)$, the variance $\sum_i (u_i - \bar{x})^2$ decreases by at least 2. Thus, there can only be a finite number of such transfers. In our model, the situation is more complicated because transfers are not immediate and involve a process of requests and acceptances. A key element of the argument is to realize that the algorithm can be interpreted as if it only involved instantaneous exchanges involving disjoint pairs of nodes.

Let us consider the difference $V(t+1) - V(t)$ at some typical time t. Changes in V are solely due to changes in the $\hat{u}_i$. Note that if a node i executes the "fulfill the acceptance" instruction at time t, node i was the originator of the request and the request path has length zero, so that it is also the final node on the path, and $s_i(t) = i$. According to our definition, $w_i(t)$ is the value w in the message received by node $s_i(t) = i$. At the next time

step, we have $w_i(t+1) = 0$ but $u_i(t+1) = u_i(t) + w$. Thus, $\hat{u}_i$ does not change, and the function V is unaffected.

By inspecting the algorithm, we see that a nonzero difference $V(t+1) - V(t)$ is possible only if some node i executes the "accept request" instruction at slot t, with some particular value $w^* \neq 0$, in which case $u_i(t+1) = u_i(t) - w^*$. For this to happen, node i received a message (Request, r) at time t from a node k for which $\mathrm{Rout}_k(t) = i$, and with $u_i(t) - r \geq 2$. That node k was the last node, $s_\ell(t)$, on the request path of some originator node ℓ. Node k receives an (Accept, w^*) message at time $t+1$ and, therefore, according to our definition, this sets $w_\ell(t+1) = w^*$.

It follows from the rules of our algorithm that ℓ had originated a request with value $r = u_\ell(t')$ at some previous time t'. Subsequently, node ℓ entered the blocked mode, preventing any modification of u_ℓ, so that $r = u_\ell(t) = u_\ell(t+1)$. Moreover, observe that $w_\ell(t)$ was 0 because by time t, no node had answered ℓ's request. Furthermore, $w_i(t+1) = w_i(t) = 0$ because having a positive w_i requires i to be in blocked mode, preventing the execution of "accept request". It follows that:

$$\hat{u}_i(t+1) = u_i(t+1) = u_i(t) - w^* = \hat{u}_i(t) - w^*,$$
$$\text{and}$$
$$\hat{u}_\ell(t+1) = r + w^* = \hat{u}_\ell(t) + w^*.$$

Using the update equation $w^* = \lfloor (u_i(t) - r)/2 \rfloor$, and the fact $u_i(t) - r \geq 2$, we obtain

$$1 \leq w^* \leq \frac{1}{2}(u_i(t) - r) = \frac{1}{2}(\hat{u}_i(t) - \hat{u}_\ell(t)).$$

Combining with the previous equalities, we have

$$\hat{u}_\ell(t) + 1 \leq \hat{u}_\ell(t+1) \leq \hat{u}_i(t+1) \leq \hat{u}_i(t) - 1.$$

Assume for a moment that node i was the only one that executed the "accept request" instruction at time t. Then, all of the variables $\hat{u}_j$, for $j \neq i$, ℓ, remain unchanged. Simple algebraic manipulations then show that V decreases by at least 2. If there was another pair of nodes, say j and k, that were involved in a transfer of value at time t, it is not hard to see that the transfer of value was related to a different request, involving a separate request path. In particular, the pairs ℓ, i and j, k do not overlap. This implies that the cumulative effect of multiple transfers on the difference $V(t+1) - V(t)$ is the sum of the effects of individual transfers. Thus, at every time for which at least one "accept request" step is executed, V decreases by at least 2. We also see that no operation can ever result in an increase of V. It follows that the instruction "accept request" can be executed only a finite number of times. ■

Proposition VI.1: There is a time t' such that $u_i(t) = u_i(t')$, for all i and all $t \geq t'$. Moreover

$$\sum_i u_i(t') = \sum_i x_i,$$
$$\max_i u_i(t') - \min_i u_i(t') \leq 1.$$

Proof: It follows from Lemma VI.4 that there is a time t' after which no more requests are accepted with $w \neq 0$. By Lemma VI.3, this implies that after at most n additional time steps, the system will never again contain any "accept request" messages with $w \neq 0$, so no node will change its value $u_i(t)$ thereafter.

We have already argued that the sum (and therefore the average) of the variables $\hat{u}_i(t)$ does not change. Once there are no more "accept request" messages in the system with $w \neq 0$, we must have $w_i(t) = 0$, for all i. Thus, at this stage the average of the $u_i(t)$ is the same as the average of the x_i.

It remains to show that once the $u_i(t)$ stop changing, the maximum and minimum $u_i(t)$ differ by at most 1. Recall (cf. Theorem V.1) that at some time after the $u_i(t)$ stop changing, all estimates $M_i(t)$ of the maximum will be equal to $M(t)$, the true maximum of the $u_i(t)$; moreover, starting at any node and following the pointers $P_i(t)$ leads to a node j whose value $u_j(t)$ is the true maximum, $M(t)$. Now let A be the set of nodes whose value at this stage is at most $\max_i u_i(t) - 2$. To derive a contradiction, let us suppose that A is nonempty.

Because only nodes in A will originate requests, and because every request eventually terminates (cf. Lemma VI.3), if we wait some finite amount of time, we will have the additional property that all requests in the system originated from A. Moreover, nodes in A originate requests every time they are in the free mode, which is infinitely often.

Consider now a request originating at a node in the set A. The value r of such a request satisfies $M(t) - r \geq 2$, which implies that every node that receives it either accepts it (contradicting the fact that no more requests are accepted after time t'), or forwards it, or denies it. But a node i will deny a request only if it is in blocked mode, that is, if it has already forwarded some other request to node $P_i(t)$. This shows that requests will keep propagating along links of the form $(i, P_i(t))$, and therefore will eventually reach a node at which $u_i(t) = M(t) \geq r + 2$, at which point they will be accepted—a contradiction. ■

We are now ready to conclude.

Proof of Theorem IV.1: Let u_i^* be the value that $u_i(t)$ eventually settles on. Proposition VI.1 readily implies that if the average $\bar{x}$ of the x_i is an integer, then $u_i(t) = u_i^* = \bar{x}$ will eventually hold for every i. If $\bar{x}$ is not an integer, then some nodes will eventually have $u_i(t) = u_i^* = \lfloor \bar{x} \rfloor$ and some other nodes $u_i(t) = u_i^* = \lceil \bar{x} \rceil$. Besides, using the maximum and minimum computation algorithm, nodes will eventually have a correct estimate of $\max u_i^*$ and $\min u_i^*$, since all $u_i(t)$ settle on the fixed values u_i^*. This allows the nodes to determine whether the average is exactly u_i^* (integer average), or whether it lies in $(u_i^*, u_i^* + 1)$ or $(u_i^* - 1, u_i^*)$ (fractional average). Thus, with some simple post-processing at each node (which can be done using finite automata), the nodes can produce the correct output for the interval-averaging problem. The proof of Theorem IV.1 is complete. ■

Next, we give a convergence time bound for the algorithms we have just described.

Theorem VI.1: Any function h satisfying the assumptions of Theorem IV.2 can be computed in $O(n^2 K^2)$ time steps.

Theorem VI.2: The functions h_ϵ whose existence is guaranteed by Corollary IV.3 or Corollary IV.4 can be computed in time which grows quadratically in n.

The general idea behind Theorems VI.1 and VI.2 is quite simple. We have just argued that the nonnegative function $V(t)$ decreases by at least 2 each time a request is accepted. It also

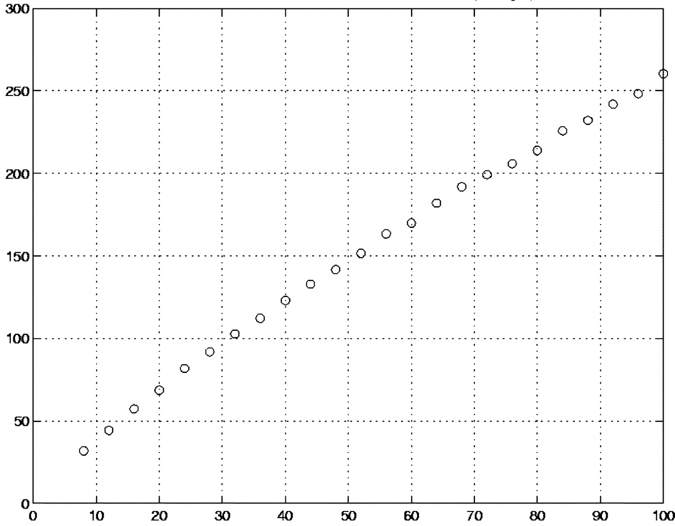

Fig. 4. Number of iterations as a function of the number of nodes for a complete graph under random initial conditions.

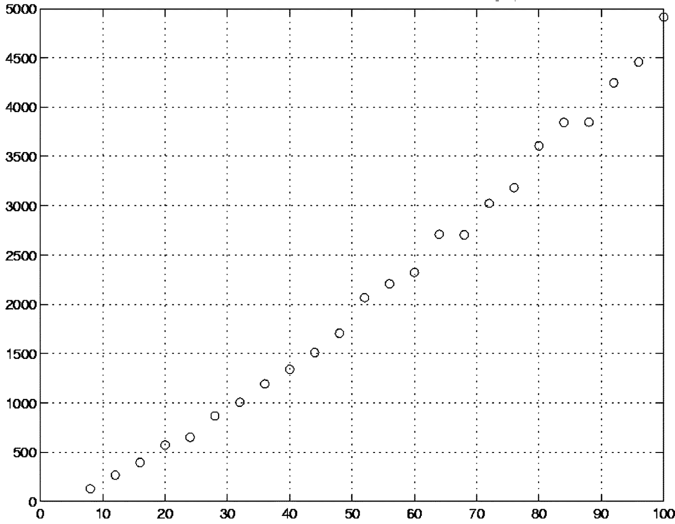

Fig. 5. Number of iterations as a function of the number of nodes for a line graph under random initial conditions.

satisfies $V(0) = O(nK^2)$. Thus there are at most $O(nK^2)$ acceptances. To prove Theorems VI.1 and VI.2, one needs to argue that if the algorithm has not terminated, there will be an acceptance within $O(n)$ time steps. This should be fairly clear from the proof of Theorem IV.1. A formal argument is given in the technical report [18]. It is also shown there that the running time of our algorithm, for many graphs, satisfies a $\Omega(n^2)$ lower bound, in the worst case over all initial conditions.

VII. Simulations

We report here on simulations involving our algorithm on several natural graphs. Figs. 4 and 5 describe the results for a complete graph and a line. Initial conditions were random integers between 1 and 30, and each data point represents the average of two hundred runs. As expected, convergence is faster on the complete graph. Moreover, convergence time in both simulations appears to be approximately linear.

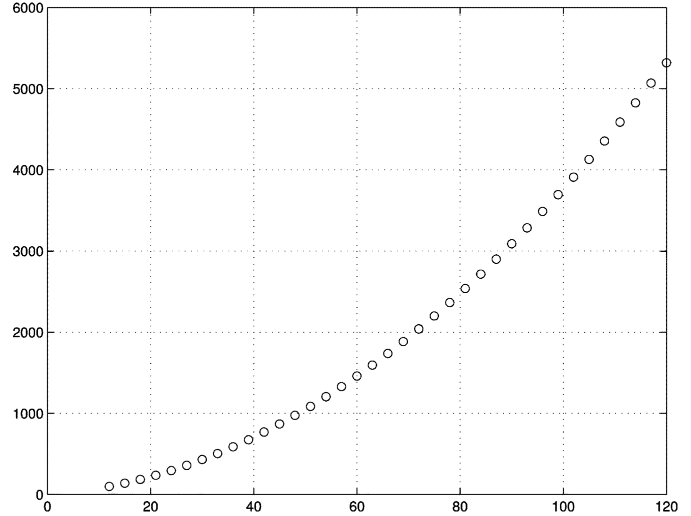

Fig. 6. The number of iterations as a function of the number of nodes for a dumbbell graph.

Finally recall that it is possible for our algorithm to take on the order of n^2 (as opposed to $O(n)$) time steps to converge. Fig. 6 shows simulation results for the dumbbell graph (two complete graphs with $n/3$ nodes, connected by a line) of length $n/3$; each node in one of the complete graphs starts with $x_i(0) = 1$, every node in the other complete graph starts with $x_i(0) = 30$. The time to converge in this case is quadratic in n.

VIII. Conclusion

We have proposed a model of deterministic anonymous distributed computation, inspired by the wireless sensor network and multi-agent control literature. We have given an almost tight characterization of the functions that are computable in our model. We have shown that computable functions must depend only on the the frequencies with which the different initial conditions appear, and that if this dependence can be expressed in term of linear inequalities with rational coefficients, the function is indeed computable. Under weaker conditions, the function can be approximated with arbitrary precision. It remains open to exactly characterize the class of computable function families.

Our positive results are proved constructively, by providing a generic algorithm for computing the desired functions. Interestingly, the finite memory requirement is not used in our negative results, which remain thus valid in the infinite memory case. In particular, we have no examples of functions that can be computed with infinite memory but are provably not computable with finite memory. We suspect though that simple examples exist; a good candidate could be the indicator function $1_{p_1 < 1/\pi}$, which checks whether the fraction of nodes with a particular initial condition is smaller than $1/\pi$.

We have shown that our generic algorithms terminate in $O(n^2)$ time. On the other hand, it is clear that the termination time cannot be faster than the graph diameter, which is of order n, in the worst case. Some problems, such as the detection problem described in Section II, admit $O(n)$ algorithms. It is an open problem whether the interval averaging problem admits an $o(n^2)$ algorithm under our model. Finally, we conjecture that the dependence on K in our $O(n^2K^2)$ complexity estimate can be improved by designing a different algorithm.

Possible extensions of this work involve variations of the model of computation. For example, the algorithm for detection problem, described in Section II, does not make full use of the flexibility allowed by our model of computation. In particular, for any given i, the messages $m_{ij}(t)$ are the same for all j, so, in some sense, messages are "broadcast" as opposed to being personalized for the different outgoing links. This raises an interesting question: do there exist computable function families that become non-computable when we restrict to algorithms that are limited to broadcast messages? We have reasons to believe that in a pure broadcast scenario where nodes located in a non-bidirectional network broadcast messages without knowing their out-degree (i.e., the size of their audience), the only computable functions are those which test whether there exists a node whose initial condition belongs to a given subset of $\{0, 1, \dots, K\}$, and combinations of such functions.

Another important direction is to consider models in which the underlying graph may vary with time. It is of interest to develop algorithms that converge to the correct answer at least when the underlying graph eventually stops changing. For the case where the graph keeps changing while maintaining some degree of connectivity, we conjecture that no deterministic algorithm with bounded memory can solve the interval-averaging problem. Finally, other extensions involve models accounting for clock asynchronism, delays in message propagation, or data loss.

REFERENCES

[1] D. Angluin, "Local and global properties in networks of processors," in *Proc. 12th Annu. ACM Symp. Theory Comp.*, 1980, pp. 82–93.

[2] J. Aspnes and E. Ruppert, "An introduction to population protocols," *Bull. Eur. Assoc. Theoretical Comp. Sci.*, vol. 93, pp. 98–117, 2007.

[3] T. C. Aysal, M. Coates, and M. Rabbat, "Distributed average consensus using probabilistic quantization," in *Proc. 14th IEEE/SP Workshop Stat. Signal Processing*, 2007, pp. 640–644.

[4] Y. Afek and Y. Matias, "Elections in anonymous networks," *Inform. Comp.*, vol. 113, no. 2, pp. 113–330, 1994.

[5] O. Ayaso, D. Shah, and M. Dahleh, "Counting bits for distributed function computation," in *Proc. IEEE Int. Symp. Inform. Theory*, 2008, pp. 652–656.

[6] H. Attiya, M. Snir, and M. K. Warmuth, "Computing on an anonymous ring," *J. ACM*, vol. 35, no. 4, pp. 845–875, 1988.

[7] F. Bénézit, P. Thiran, and M. Vetterli, "Interval consensus: From quantized gossip to voting," in *Proc. ICASSP'08.*, 2008, pp. 3661–3664.

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

[9] F. Bullo, J. Cortés, and S. Martítnez*, Distributed Control of Robotic Networks*. Princeton, NJ: Princeton Univ. Press, 2009.

[10] R. Carli and F. Bullo, "Quantized coordination algorithms for rendezvous and deployment," *SIAM J. Control Optim.*, vol. 48, no. 3, pp. 1251–1274, 2009.

[11] I. Cidon and Y. Shavitt, "Message terminating algorithms for anonymous rings of unknown size," *Inform. Processing Lett.*, vol. 54, no. 2, pp. 111–119, Apr. 1995.

[12] M. Draief and M. Vojnovic, "Convergence speed of binary interval consensus," in *Proc. Infocom'10*, 2010, pp. 1–9.

[13] P. Frasca, R. Carli, F. Fagnani, and S. Zampieri, "Average consensus on networks with quantized communication," *Int. J. Robust Nonlin. Control*, vol. 19, no. 6, pp. 1787–1816, 2009.

[14] F. Fich and E. Ruppert, "Hundreds of impossibility results for distributed computing," *Distrib. Comp.*, vol. 16, pp. 121–163, 2003.

[15] A. Giridhar and P. R. Kumar, "Computing and communicating functions over sensor networks," *IEEE J. Selected Areas Commun.*, vol. 23, no. 4, pp. 755–764, Apr. 2005.

[16] P. Gacs, G. L. Kurdyumov, and L. A. Levin, "One-dimensional uniform arrays that wash out finite islands," *Problemy Peredachi Informatsii*, vol. 14, no. 3, pp. 92–96, 1978.

[17] A. G. Greenberg, P. Flajolet, and R. Lander, "Estimating the multiplicity of conflicts to speed their resolution in multiple access channels," *J. ACM*, vol. 34, no. 2, pp. 289–325, 1987.

[18] J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis, Distributed Anonymous Discrete Function Computation and Averaging Tech. Rep., 2010 [Online]. Available: http://arxiv.org/abs/1004.2102

[19] Y. Hassin and D. Peleg, "Distributed probabilistic polling and applications to proportionate agreement," *Inform. Comp.*, vol. 171, no. 2, pp. 248–268, 2001.

[20] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Trans. Autom. Control*, vol. 48, no. 6, pp. 988–1001, Jun. 2003.

[21] A. Kashyap, T. Basar, and R. Srikant, "Quantized consensus," *Automatica*, vol. 43, no. 7, pp. 1192–1203, 2007.

[22] E. Kranakis, D. Krizanc, and J. van den Berg, "Computing boolean functions on anonymous networks," in *Proc. 17th Int. Colloq. Autom., Languages Programming*, Jul. 1990, pp. 254–267.

[23] N. Khude, A. Kumar, and A. Karnik, "Time and energy complexity of distributed computation in wireless sensor networks," in *Proc. INFOCOM*, 2005, pp. 2625–2637.

[24] N. Katenka, E. Levina, and G. Michailidis, "Local vote decision fusion for target detection in wireless sensor networks," *IEEE Trans. Signal Processing*, vol. 56, no. 1, pp. 329–338, Jan. 2008.

[25] S. Kar and J. M. Moura, "Distributed average consensus in sensor networks with quantized inter-sensor communication," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Processing*, 2008, pp. 2281–2284.

[26] N. Lynch*, Distributed Algorithms*. Waltham, MA: Morgan-Kauffman, 1996.

[27] M. Land and R. K. Belew, "No perfect two-state cellular automaton for density classification exists," *Phys. Rev. Lett.*, vol. 74, no. 25, pp. 5148–5150, Jun. 1995.

[28] Y. Lei, R. Srikant, and G. E. Dullerud, "Distributed symmetric function computation in noisy wireless sensor networks," *IEEE Trans. Inform. Theory*, vol. 53, no. 12, pp. 4826–4833, Dec. 2007.

[29] L. Liss, Y. Birk, R. Wolff, and A. Schuster, "A local algorithm for ad hoc majority voting via charge fusion," in *Proc. DISC'04*, Amsterdam, The Netherlands, Oct. 2004, pp. 275–289.

[30] S. Martinez, F. Bullo, J. Cortes, and E. Frazzoli, "On synchronous robotic networks—Part I: Models, tasks, complexity," *IEEE Trans. Robot. Autom.*, vol. 52, no. 12, pp. 2199–2213, Dec. 2007.

[31] S. Mukherjee and H. Kargupta, "Distributed probabilistic inferencing in sensor networks using variational approximation," *J. Parallel Distrib. Comp.*, vol. 68, no. 1, pp. 78–92, 2008.

[32] S. Moran and M. K. Warmuth, "Gap theorems for distributed computation," *SIAM J. Comp.*, vol. 22, no. 2, pp. 379–394, 1993.

[33] A. Nedic, A. Olshevsky, A. Ozdaglar, and J. N. Tsitsiklis, "On distributed averaging algorithms and quantization effects," *IEEE Trans. Autom. Control*, vol. 54, no. 11, pp. 2506–2517, Nov. 2009.

[34] D. Peleg, "Local majorities, coalitions and monopolies in graphs: A review," *Theoretical Comp. Sci.*, vol. 282, no. 2, pp. 231–237, 2002.

[35] E. Perron, D. Vasuvedan, and M. Vojnovic, "Using Three States for Binary Consensus on Complete Graphs," in *Proc. IEEE INFOCOM Conf.*, 2009, pp. 2527–2535.

[36] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Syst. Control Lett.*, vol. 53, pp. 65–78, 2004.

[37] M. Yamashita and T. Kameda, "Computing on an anonymous network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 1, pp. 69–89, Jan. 1996.

[38] M. Zhu and S. Martinez, "On the convergence time of asynchronous distributed quantized averaging algorithms," *IEEE Trans. Autom. Control*, vol. 56, no. 2, pp. 386–390, Feb. 2011.

Julien M. Hendrickx received the M.S. degree in applied mathematics and the Ph.D. degree in mathematical engineering from Université catholique de Louvain, Ecole Polytechnique de Louvain, Belgium, in 2004 and 2008, respectively.

He is Assistant Professor (chargé de cours) at the Université Catholique de Louvain, Ecole Polytechnique de Louvain, Belgium, since September 2010. He was a Visiting Researcher at the University of Illinois at Urbana Champaign, from 2003 to 2004, at the National ICT Australia in 2005 and 2006, and at the Massachusetts Institute of Technology in 2006 and 2008. He was a Postdoctoral Fellow at the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, in 2009 and 2010, holding Postdoctoral Fellowships of the Fund for Scientific Research (F.R.S.-FNRS) and the Belgian American Education Foundation since September 2010.

Dr. Hendrickx received the 2008 EECI Award for the best Ph.D. thesis in Europe in the field of Embedded and Networked Control, and the Alcatel-Lucent-Bell 2009 Award for a Ph.D. thesis on original new concepts or application in the domain of information or communication technologies.

Alex Olshevsky received the B.S. degree in mathematics and the B.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge.

He is a Postdoctoral Scholar with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ, until Fall 2011 when he will be an Assistant Professor with the Department of Industrial and Enterprise System Engineering, University of Illinois at Urbana-Champaign. His research interests are in control theory, optimization, and applied probability.

John N. Tsitsiklis (F'99) received the B.S. degree in mathematics and the B.S., M.S., and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, in 1980, 1980, 1981, and 1984, respectively.

He is currently a Clarence J. Lebel Professor with the Department of Electrical Engineering, MIT. He has served as a Codirector of the MIT Operations Research Center from 2002 to 2005, and in the National Council on Research and Technology in Greece (2005–2007). His research interests are in systems, optimization, communications, control, and operations research. He has coauthored four books and several journal papers in these areas.

Dr. Tsitsiklis received the Outstanding Paper Award from the IEEE Control Systems Society (1986), the M.I.T. Edgerton Faculty Achievement Award (1989), the Bodossakis Foundation Prize (1995), and the INFORMS/CSTS Prize (1997). He is a member of the National Academy of Engineering. Finally, in 2008, he was conferred the title of Doctor honoris causa, from the Université Catholique de Louvain.