Computations on Time-Varying Sensor Networks

K. Mani Chandy Michel Charpentier* Concetta Pilotto

Computer Science Department
California Institute of Technology MC 256-80
Pasadena, CA 91125
{mani,charpov,pilotto}@cs.caltech.edu

ABSTRACT

This paper presents a unified methodology for designing sensor networks that are analog computing systems, time-stepped synchronous systems, shared-memory asynchronous systems, and reliable or unreliable message-passing systems. The dynamics of the systems may by specified by differential equations, difference equations or temporal logic. Communication and computational resources in the network vary with time due to environmental factors such as noise interference or attacks by adversaries. Computations proceed by opportunistically employing the resources available at each point, progressing rapidly when more resources are available and slowing down when resources become unavailable. The paper proposes mathematical models for a variety of sensor network types and suggests a uniform framework for designing algorithms for them. The impact of the persistence of network connectivity on the design of algorithms is explored: if network connectivity remains unchanged for long periods then algorithms that compute global parameters of the network are efficient. If, by contrast, the network changes frequently then estimates of global parameters are likely to be incorrect and atomic operations across the entire network are likely to be aborted; in this case algorithms in which agents use local information are more efficient. The paper develops design techniques and performance analysis of systems in which agents use information with different degrees of locality ranging from immediate neighbors to global parameters.

General Terms

Algorithms, Performance, Design

Keywords

Wireless Sensor Networks, Distributed Systems, Distributed Control

*On leave from the University of New Hampshire (2006/07).

To be submitted for publication. This research is supported in part by AFOSR MURI award FA9550-06-1-0303.

1. INTRODUCTION

We present a unified methodology for designing algorithms, simulation results and performance analysis of algorithms that compute functions on sensor fields for different types of networks including distributed analog computing systems, centralized digital computing systems, synchronous and asynchronous distributed message-passing systems and mobile agent systems. The resources available to the algorithm may vary over time due to environmental causes, such as noise, or due to attacks by adversaries. We develop algorithms that proceed opportunistically using the resources available at the time.

We specify problems that are abstractions of real problems that occur in design. The relationship between this abstraction and the process of design is discussed later in this section. Next, we introduce the problem specification and then present two simple examples.

1.1 Problem Specification

Let s, s', $s^{(k)}$ and $s^{(t)}$ be system states. For discrete-step computations we use the notation $s^{(k)}$ for the k-th value of S in a computation for integer $k \geq 0$. For analog computations in which the state varies continuously over time we use $s^{(t)}$ for the state at time t, for real values $t \geq 0$. The specification consists of a progress property and a set of safety properties.

Progress Property: Stable Termination

The progress property is characterized by a function f from states to states. For systems in which the state takes on discrete values the progress property is that for any computation initiated in any state $s^{(0)}$, there is a point in the computation from which the state of the system remains $f(s^{(0)})$ forever. (This progress property can be defined using the "to-always" operator [5, 4, 9].) Let $s^* = f(s^{(0)})$. Thus s^* is the desired end state.

For systems in which the state space is continuous we extend the definition of the progress property from the discrete case to the continuous case in the usual way: The state $s^{(t)}$ at time t tends to s^* as t tends to infinity. We introduce a "distance function" D between states that we use in defining the meaning of $s^{(t)}$ *tends to* s^*. $D(s, s')$ is a measure of the distance between the states s and s'. The meaning of $s^{(t)}$ *tends to* s^* is that the distance between $s^{(t)}$ and s^* gets arbitrarily small as t tends to infinity. Thus the progress property is that for any positive ϵ there is a point τ in the computation such that $D(s^{(t)}, s^*) < \epsilon$ for $t > \tau$.

$$\forall \epsilon > 0 : \exists \tau : \forall t > \tau : D(s^{(t)}, s^*) < \epsilon \qquad (1)$$

Safety Properties

In this paper a safety property is specified by a function e from states to reals where $e(s^{(t)})$ is nondecreasing with t:

$$t > r \Rightarrow e(s^{(t)}) \leq e(s^{(r)}) \qquad (2)$$

We call $e(s^{(t)})$ an "error at time t" because in most problems it decreases as the distance between the current state $s^{(t)}$ and the desired state s^* decreases. A specification may have more than one function e.

1.2 Examples

Formation of Mobile Agents

Agents indexed j for $0 \leq j \leq N$ are placed arbitrarily in a 2-dimensional space. Agents 0 and N do not move. Agents $1 \ldots N-1$ are required to move such that all agents form a straight line with equidistant spacing between agents where agent j is adjacent to agent $j+1$ (for $0 \leq j < N$) (see Fig. 1). The state S is an array indexed j where S_j is a tuple $[j, x_j, y_j]$ where j is the id of the agent and $[x_j, y_j]$ are its coordinates in 2-D space. The desired state s^* is an array of the same type as S, where all points $[x_j, y_j]$ lie on a line and where the distance between adjacent points $[x_j, y_j]$ and $[x_{j+1}, y_{j+1}]$ is the same for all j. Examples of safety properties in the specification are: *max* x and *max* y do not increase with t, and *min* x and *min* y do not decrease with t.

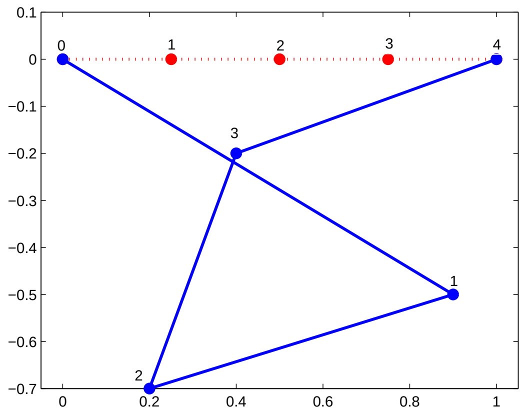

Figure 1: The initial and final configurations for the Formation of Mobile Agents example when $N = 4$. Blue and red circles are respectively the initial and final positions of the agents. The blue edges represent the connections between agents.

Computing the Average

Let S be a bag $[S_0, \ldots, S_N]$ where S_j is a variable in agent j of a sensor network. (A bag is a multiset.) Then the desired state s^* is a bag each of whose elements is the average of the values in the initial bag $s^{(0)}$. An example of a safety property in the specification is that the gap between the largest and smallest value in S is nonincreasing with time.

1.3 Abstract Problem and Actual Design

The problem as specified is that the computation must reach a point after which the state becomes and remains arbitrarily close to the desired state s^* forever, where $s^* = f(s^{(0)})$ and $s^{(0)}$ is the initial state. In practice sensor values change, and a more appropriate specification is that the network must compute a function $f(s)$ where s is the current value of the state; as the value s of the current state changes so does $f(s)$. For the network to compute $f(s)$ precisely, from instant to instant, may not be possible given inherent delays in communication and computation. The problem we address in this paper deals with the situation where sensor values have ceased changing. Our problem is an abstraction of designs of systems in which agents sample sensor values from time to time and then compute functions on the sampled values; the sampled values are treated as static until the next sample is taken. This paper does not deal with the problem of ensuring that the desired value is reached before the next sample is taken.

2. EARLIER WORK

Self-organizing, *ad hoc* networks with limited sensing, computation and communication capabilities have been widely studied [32, 2, 3]. Distributed routing and fusion algorithms that are scalable, fault-tolerant and robust to network changes are presented in [19, 31, 24, 13, 33, 11]. Pioneering studies and surveys of consensus algorithms include [7, 6, 14, 20]. Alanyali et al. in [1] consider a scenario of multiple distributed noisy sensors observing a single event where sensors need to reach consensus about the event after exchanging messages through a network. Vicsek in [27] provides simulation results which demonstrate that local rules can cause all agents to eventually move in the same direction despite the absence of centralized coordination. These problems find use in a variety of applications, such as source detection and localization [18], vehicle formation [8] and robot synchronization [21].

Algorithms for distributed computation of averages of the node data over networks are presented in [12, 22, 23, 25, 29, 30]. In [17] the algorithm is extended to the case when the connectivity of the network varies with time and in [16] an asynchronous version of the algorithm is proposed. In [28] the average problem is investigated in the case when the network is a digraph; this result takes into account the fact that sensors may have a limited field of view which is not always symmetric. Recently stochastic variants of the problems have been investigated in [10] and [26].

This paper extends earlier research in three ways:

1. We present models of dynamic continuous-transition, discrete-transition, asynchronous shared-memory and message-passing systems in which agents and channels can be disabled and enabled in an arbitrary fashion over the short term and with only weak constraints over long-term behavior.

2. We propose a unified methodology for designing algorithms for dynamic networks in which algorithms are specified by differential equations, difference equations or temporal logic.

3. We carry out performance analysis evaluating the impact of locality for different problems.

3. DYNAMIC SENSOR NETWORKS

3.1 Features Common to All Networks

First we discuss features common to all types of sensor networks and then discuss additional features appropriate for each network type.

Connectivity Graphs and the Environment

The connectivity of the network is specified by a directed or undirected graph, called the *connectivity graph*, in which vertices represent enabled computation agents and edges represent enabled communication channels. When the environment disables an agent or channel the corresponding vertex or edge is removed from the graph, and when an agent or channel is re-enabled the corresponding vertex or edge is added to the graph. In analog systems agents send and receive analog signals, and a directed edge (u, v) indicates that v receives signals generated by u. In synchronous time-stepped networks and in asynchronous message-passing networks, an edge represents a message channel. In shared-memory networks, an undirected edge $\{u, v\}$ indicates that agents u and v can access each other's memory.

Constraints on the Environment

We consider systems in which there are no constraints on what the environment can do in a single step: the environment can disable and enable agents and communication arbitrarily. Without any constraints, the environment can permanently disable all agents in which case the state never changes. Therefore we consider constraints on the long-term behavior of the environment though there are no constraints on short-term behavior. Thus, constraints on the environment are progress properties and not safety properties.

3.2 Analog Sensor Networks

The dynamics of analog networks are specified by a set of differential equations where each equation defines the rate of change of variables at a node as a function of the values of variables in neighboring nodes. Let x_v be a variable at agent v. Let $x_v^{(t)}$ be the value of x_v at time t for $t \geq 0$. Each agent has a sensor, and the value read by the sensor at v is $x_v^{(0)}$. After the value of the sensor at v is copied into $x_v^{(0)}$ the sensor is not read again until $f(x^{(0)})$ has been computed to the desired degree of accuracy. Thus the sensor readings are the *initial* values of variables.

The rate of change of x_v is a function of x_v itself and the values x_u of neighboring agents u with edges (u, v) in the connectivity graph. Let E be the set of edges of the connectivity graph. The dynamics of an enabled agent v is specified by:

$$\dot{x}_v = g(x_v, \{x_u | (u, v) \in E\}) \tag{3}$$

where $\dot{x}_v = dx_v/dt$. One of the tasks in designing a network is to determine function g. A change in the edges of the connectivity graph may modify the second argument of g, and thus change the dynamics. The dynamics of a disabled agent v is given by $\dot{x}_v = 0$ since a disabled agent does not change state.

Let x be the vector with elements x_v and let $G(x, E)$ be a vector whose v-th element is $g(x_v, \{x_u | (u, v) \in E\})$. Then the dynamics of the system are given by:

$$\dot{x} = G(x, E) \tag{4}$$

All the equations dealing with $G(x, E)$ are (implicitly) quantified over all E because we allow the environment to change E arbitrarily; therefore the formulae apply regardless of E. A steady state solution is one in which $\dot{x} = 0$ and therefore $G(x, E) = 0$. We require that the desired end state x^* is a steady state solution: $G(x^*, E) = 0$ for all E.

3.3 Synchronous Time-Stepped Networks

A synchronous time-stepped system is one in which agents exchange messages at prespecified instants. The dynamics are specified by difference equations similar to the differential equations given for analog systems. The change in value of a variable x_v is a function of its current value and the values of its neighbors u for which edge (u, v) exists. Difference equations have the form:

$$x_v^{(k+1)} - x_v^{(k)} = g(x_v^k, \{x_u^k | (u, v) \in E\}) \tag{5}$$

In vector terms the equation is

$$x^{(k+1)} - x^{(k)} = G(x^{(k)}, E) \tag{6}$$

where g and G are similar to the case of analog systems. Since time-stepped systems are similar to analog systems we don't discuss them further.

3.4 Asynchronous Shared-Memory Systems

Shared-memory models are appropriate for systems in which agents can access and modify the states of other agents in atomic actions. At each point in a computation the environment partitions the set of agents into *groups* where agents in a group cannot access or modify state of agents in other groups. An agent can access and modify its own state. An agent in a group may, in addition, be able to execute atomic operations that access and modify the states of some of the other agents in its group. The environment may re-partition the set of agents into groups, in an arbitrary manner, at each step of a computation. The implementation of atomic operations by groups is not discussed here, but the influence of aborting operations is studied in section 7.

Shared-memory models are appropriate for certain types of mobile agent systems. As agents move, communication between certain sets of agents may be disrupted, while communication capabilities between other sets of agents are enhanced. Sets of agents with reliable inter-agent communication (if only for a limited duration) form groups that can execute atomic operations on states of some of the agents in the group. Communication capacity between agents can change in unexpected ways, as agents move, due to noise and geographical features.

Algorithms for shared-memory systems are generalizations of algorithms for analog computing networks and synchronous time-stepped networks. We obtain the models for analog and synchronous networks by treating each connected component of the (undirected) connectivity graph as a group.

3.5 Asynchronous Message-Passing Systems

In this model, agents use communication channels to send and receive discrete messages asynchronously. Agents cannot send messages on a disabled channel and a disabled channel does not deliver messages to agents. A disabled agent does not change state and therefore can no longer send or receive messages.

The environment can enable and disable agents and channels arbitrarily at each step. Different types of long-term environmental constraints are appropriate for different systems. In the case of reliable channels a constraint on environmental behavior is that every message sent on a reliable channel is delivered in the order in which it is sent. In the case of unreliable channels a constraint on the environment is that if an agent sends the same message infinitely often along a channel then at least one copy of the message is delivered. The specifications of message-passing systems can be given in temporal logic [15].

4. METHODOLOGY

We develop the methodology in two steps: we determine necessary properties of all algorithms that solve the problem and then we refine these properties to obtain classes of algorithms for the different types of sensor networks described in the previous section.

4.1 Necessary Properties

Let s and s' be states of the system where there exists a transition from s to s'. In a discrete transition system, a computation in state s can make an atomic transition to state s'. For continuous transition systems there exists a computation in which the state is s' some time after the state is s. Next we determine relations between s and s'. Since errors do not increase (2):

$$e(s') \leq e(s) \tag{7}$$

From the specification, if the system is started in any state s then the system will eventually become and remain $f(s)$. Consider a system initiated in state $s^{(0)}$ and let $s^{(t)}$ be a later state of this system. This system must eventually reach the state $f(s^{(0)})$. But, for times $t' \geq t$, this system does not differ from a system initiated in state $s^{(t)}$, and therefore it must eventually reach and remain in the state $f(s^{(t)})$. It follows that $f(s^{(0)}) = f(s^{(t)})$ and hence that $f(s^{(t)})$ is the same for all t. Since $f(s^{(t)})$ is unchanged over all time t we call the following the **conservation law** of f.

$$f(s') = f(s) \tag{8}$$

As the computation proceeds the state gets closer to the desired end state s^*. Therefore in our design we attempt to identify a Lyapunov function, a measure h of the distance between the current state and the desired end state s^*, that *i)* does not increase as the computation proceeds and *ii)* converges eventually to a value h_{min}, where *iii)* if h converges to h_{min} then s converges to s^*.

$$h(s') \leq h(s) \tag{9}$$

Specifications and proofs of convergence are discussed later.

Example: Computing the Average

The state is a vector x of values where x_u is the value of agent u. Assume there exists a transition from x to x'. Equation (7) for this example is:

$$\mathit{max}\, x' \leq \mathit{max}\, x \qquad\qquad \mathit{min}\, x' \geq \mathit{min}\, x$$

Equation (8) is: $\mathit{avg}\ x' = \mathit{avg}\ x$ which is equivalent to: $\sum_u x'_u = \sum_u x_u$

We propose $h = \sum_u x_u^2$. The proof that it takes its minimum exactly when $x_u = \mathit{avg}\, x$ for all u is trivial. Equation (9) is: $\sum_u (x'_u)^2 \leq \sum_u x_u^2$. Therefore we permit any state transition from x to x' that satisfies the above equations. A possible transition from a state $x = [0, 1, 3, 27, 29, 30]$ is to $x' = [0, 0, 14, 16, 30, 30]$ illustrating that values can move away from the average becoming minima or maxima: 1 decreases to 0, and 29 increases to 30 though the mean is 15. Properties (7), (8) and (9) guarantee only that the constraints are satisfied; we prove later that minimum value of h, subject to the constraints, is reached eventually.

Example: Gaussian Mean

The previous example can be generalized, in a straightforward way, to compute the Gaussian mean in which each agent j has a vector x_j of dimension M and a positive-definite symmetric "weight" $M \times M$ matrix W_j, and the problem is to reach an end state at which the value of each agent j is $x_j = (\sum_j W_j)^{-1} \cdot \sum_j W_j.x_j$.

4.2 Properties of Analog Networks

THEOREM 1. *Properties (7), (8) and (9), are equivalent, respectively, to the following in order.*

$$\nabla e(x) \cdot G(x, E) \leq 0 \tag{10}$$

$$\nabla f(x) \cdot G(x, E) = 0 \tag{11}$$

$$\nabla h(x) \cdot G(x, E) \leq 0 \tag{12}$$

PROOF. We give the proof for (12); the proofs for the other equations are similar. From the chain rule, $\dot{h} = \nabla h(x) \cdot \dot{x}$. From (4), substitute $G(x, E)$ for $\dot{x}$ to get $\dot{h} = \nabla h(x) \cdot G(x, E)$. The result follows since $\dot{h}$ is nonpositive from (9). $\square$

The following 4-step method helps in developing algorithms that are independent of the structure and size of the connectivity graph:

1. Propose a function h.
2. Obtain differential equations that satisfy (10), (11) and (12) for a 2-vertex graph.
3. Determine the differential equation for the agent at the center of a star network where the center has k vertices, for arbitrary k, assuming that the equations governing the k agents at the periphery are those determined for the 2-vertex graph in the previous step.
4. Verify that the equations (10) - (12) hold for any arbitrary graph where the dynamics of each agent with k neighbors are those determined for the center of a star with k neighbors.

We apply the method to the average example as follows:

1. As before we propose $h = \sum_u x_u^2$.
2. The equations (11) and (12) for a 2-vertex graph with vertices u and v and an edge between them imply:
$$\dot{x}_u + \dot{x}_v = 0$$
$$2x_u.\dot{x}_u + 2x_v.\dot{x}_v \leq 0$$
Subtracting $2x_u$ times the first equation from the inequality and simplifying gives:
$$(x_v - x_u).\dot{x}_v \leq 0$$
This inequality suggests $\dot{x}_v = -\alpha(x_v - x_u)$ where α is a positive constant, since squares of real numbers of nonnegative.
3. Given g for a vertex with a single neighbor, the equations for the star network yield:
$$\dot{x}_v = \alpha.\big(\sum_{u \in N(v)} x_u - |N(v)|.x_v\big)$$
where $N(v)$ is the set of neighbors of v.
4. Show that the functions g computed in the previous steps satisfy (10), (11) and (12).

4.3 Application to Synchronous Networks

The problem for time-stepped synchronous systems in which connectivity is specified by a graph is similar.

THEOREM 2. *Properties (7), (8) and (9), are equivalent, respectively, to the following in order.*

$$e(x + G(x, E)) \leq e(x) \tag{13}$$
$$f(x + G(x, E)) = f(x) \tag{14}$$
$$h(x + G(x, E)) \leq h(x) \tag{15}$$

We apply the 4-step method given for analog networks to design an algorithm for computing averages in synchronous networks. The first step is unchanged: h is a sum of squares. The second step suggests that $g(x_u, \{x_v\}) = \alpha.(x_v - x_u)$ for any α where $0 < \alpha < 1$. The third step shows that only α satisfying $0 < \alpha < 1/n$, where n is the degree of the center, reduces h. In the fourth step we verify that this difference equation satisfies properties (13) - (15) for any graph.

4.4 Application to Shared-Memory Systems

Let s and s' be states of a group B of agents such that B can transit from s to s'. We restrict attention to algorithms in which all groups are self-similar: they behave in the same way regardless of their size and identities. Therefore there is a transition from s to s' of group B exactly when s and s' satisfy the necessary properties: (7), (8) and (9).

Consider a system consisting of two groups B and D of agents, and let r be a state of D. A transition of B's state from s to s' while D's state remains unchanged at r is also a transition of the state of the total system consisting of B and D from $s \cup r$ to $s' \cup r$; hence the necessary properties (7), (8) and (9) must hold for this transition of the total system as well. Let $S = s \cup r$ and let $S' = s' \cup r$. Then we have the obligation to prove the following property called the *local-global property*: any transition of system B that satisfies the necessary properties is also a transition of the total system, composed of B and D; therefore, this transition must also satisfies the necessary properties. This property is called the *local-global property* because it relates local state transitions to global ones.

Local-Global Property:

$$(e(s') \leq e(s)) \wedge (f(s') = f(s)) \wedge (h(s') < h(s))$$
$$\Rightarrow (e(S') \leq e(S)) \wedge (f(S') = f(S)) \wedge (h(S') < h(S)) \tag{16}$$

We can often prove local-global properties by proving the following simpler inequalities:

$$e(s') \leq e(s) \;\Rightarrow\; e(S') \leq e(S) \tag{17}$$
$$f(s') = f(s) \;\Rightarrow\; f(S') = f(S) \tag{18}$$
$$h(s') \leq h(s) \;\Rightarrow\; h(S') \leq h(S) \tag{19}$$

We prove the above equations for a group D consisting of a single agent and then prove the equations for groups of arbitrary size by induction. For many problems we can simplify proofs of these equations even further by recognizing that e, f and h can be expressed in terms of binary, commutative, associative and monotone operators — such as *max*, *min* or *sum* — on states of individual agents; in this case no proof of the local-global property is required because it follows directly from the properties of the operator.

4.5 Application to Message-Passing

Message-passing systems in which agents and channels can be enabled and disabled can be modeled as asynchronous shared-memory systems in which channels are represented as agents. A process u can send a message — and thus change the state — of a channel (u, v) at a point in a computation if and only if the agents corresponding to both the process u and the channel (u, v) are enabled and are both in the same group. The applications of the necessary properties, (7), (8) and (9), are similar for asynchronous message-passing and shared-memory systems.

5. EXAMPLES

We present very brief descriptions of applications of the methodology to develop solutions to the group formation problem which was given earlier and is reviewed next. Agents indexed j for $0 \leq j \leq N$ are placed arbitrarily in a 2-dimensional space, and agents $1 \ldots N-1$ are required to move such that all agents form a straight line with equidistant spacing between agents where agent j is adjacent to agent $j+1$. Agents 0 and N do not move. The safety properties of the specification are that *max* x^t and *max* y^t are nonincreasing with t, and likewise *min* x^t and *min* y^t are nondecreasing with t. Applying the methodology we derive the conservation law and the propose function h:

1. **Conservation Law**: All problems with the same number of agents and the same positions of the agents at the two ends — i.e., agents indexed 0 and N — yield the same result regardless of the initial conditions of the other agents. Therefore, the conservation law is that the number of agents and the positions of agents at both ends are unchanging.

2. **Function** h: One possible function h is the sum of squares of the distances between neighboring points:

$$h = \sum_{j>0} [(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2]$$

Application to Analog Networks

Applying the methodology directly gives the following equation governing the movement of an enabled agent j which has enabled edges from agents $j-1$ and $j+1$.

$$\dot{x}_j = \alpha.(x_{j-1} + x_{j+1} - 2x_j)$$
$$\dot{y}_j = \alpha.(y_{j-1} + y_{j+1} - 2y_j)$$

where α is any (positive) constant of proportionality. If agent j is disabled or if edges from its neighbors don't exist then agent j does not change state.

Application to Synchronous Time-Stepped Networks

The results are the same as for analog networks where the step size from one step to the next is $\alpha = 1/2$.

Application to Shared Memory Systems

A group of agents takes an action if and only if the group includes all agents j in a contiguous range $i \le j \le k$ for some $i < k$. Thus a group of agents indexed $3, 4, 5, 6, 7$ takes a joint action whereas a group consisting of agents 3, 5 and 7 takes no action. Since all groups take self-similar actions, a group consisting of all agents j with indices in the range $[i, k]$ move to positions on the straight line joining the points (x_i, y_i) and (x_k, y_k) which sets h for the group to its minimum value. Proofs of (7) - (9) and (17) - (19) and the local-global property are trivial since *max*, *min*, and + are associative, commutative and monotone in their arguments.

Application to Asynchronous Message-Passing Networks

We consider unreliable channels in which messages can be lost. The sequence of messages delivered on a channel is a ragged subsequence of the sequence of messages sent on the channel. (A ragged subsequence of a sequence z is obtained by deleting, but not permuting, arbitrary elements of z.) Message delays are arbitrary. A message that is sent infinitely often is delivered eventually.

When an agent moves to a location it periodically sends a message containing its current location. Thus, if an agent remains at the same location after time t then it sends the same message repeatedly from t onwards. Each agent j keeps a record of the last message received from each of its neighbors: agents $j-1$ and $j+1$. Each message is a point in the plane; let the points in the last messages that agent j has from agents $j-1$ and $j+1$ be $P_j[j-1]$ and $P_j[j+1]$ respectively. Agent j moves to midway between these points. Since agent j sends messages periodically, it will send a message containing its new location or a later message with an even more recent location.

The state of the system consists of a set of points where each point represents either the current location of an agent or a message in transit along a channel. When a message is lost in the channel the point corresponding to the message is deleted from the state. Likewise, when a message is consumed by the agent the message is deleted from the state.

The measure h that we use is the area of the convex hull of points (x_j, y_j). From properties of convex sets, the convex set never expands to include new points. Next we show that this convex set shrinks eventually. For each point, consider the line consisting of the locations of that point and an agent at one ends, say agent 0. Identify a point k such that this line is a supporting hyperplane: all points in the state lie entirely on one side of the line. From the description of the algorithm, this point k will eventually move (if it represents an agent) or disappear (if it represents a message) and thus the convex set will shrink.

Other Examples

The methodology has been applied to other problems that are not described here due to lack of space. Another mobile agent problem is to design algorithms in which each agent j has its x-coordinate fixed and moves in the y direction to form a geometric shape uniquely defined by the locations of the end points; for example, agent j is required to be at location $b.e^{c \cdot j}$ where b and c are constants determined by the location of the agents 0 and N. Exactly the same methodology can be used to develop algorithms for this case. Additional examples include computing the convex hull of a set of points, the shortest paths between every pair of points, and order-statistics.

When the Methodology Doesn't Work

The methodology doesn't work in all cases; however, when it doesn't work the exercise of applying the method sometimes suggests a solution — compute the desired value as a function of values that *can* be computed using the method. Consider the problem where each agent j has a pair of values x_j, y_j and the problem is to compute the linear regression of y given x. The direct application of the methodology doesn't work for this problem, but it suggests a solution: apply the methodology to compute sums, and then use the computed values to determine the desired value. Likewise, direct application of the methodology to compute the minimum circumscribing circle of a set of points fails; however, it suggests that we apply the methodology to compute the convex hull, and then use the convex hull to compute the circumscribing circle.

6. CONVERGENCE AND TERMINATION DETECTION

There are two aspects to the proof of convergence in dynamic systems: (i) Classical proofs of convergence of differential or difference equations, as for example in proving that e^{-ct} converges to 0 for any positive constant c. (ii) Proving convergence regardless of how the environment changes the connectivity graph in the short term and with only weak constraints over the environment in the long term. For example, consider the computation of the average of $[0, 2, 4]$ in an asynchronous shared-memory system with agents indexed $0, 1, 2$ where the environment alternately disables agents 2 and 0. A possible sequence of values of agents in this case is: $[1, 1, 4]$, $[1, 2.5, 2.5]$, $[1.75, 1.75, 2.5]$ and so on. We have to identify weak constraints on the environment for which we can prove that the values converge to the mean of 2.

The general method is as follows. We prove that for any positive ϵ, there exists a point in all computations after which, for every agent j, the absolute difference between the agent's value x_j and the desired final value x_j^* remains less than ϵ. For every positive value of ϵ we determine a positive e such that if there exists an agent j such that $|x_j - x_j^*| \geq \epsilon$ then $h - h_{min} > e$. We then show that eventually $h - h_{min} \leq e$.

Next, we apply this method to the problem of computing averages with the following constraint on the environment. Let the agents be indexed j for $0 \leq j \leq N$. For every agent $j < N$, groups that include the pair of agents $(j, j+1)$ execute actions infinitely often. This constraint is weak because it only requires groups consisting of pairs of consecutively-indexed agents to execute atomic actions. Examples of stronger constraints are those that require larger groups to act, or that require more groups — such as all pairs of agents and not merely all consecutively-indexed pairs — to act.

It is straightforward to show that if there exists at least an agent j such that $x_j - C \geq \epsilon$ then $h - h_{min} > \epsilon^2$ and there exists a pair $(i, i+1)$ of agents such that x_i and x_{i+1} differ by at least ϵ/N. When any group that contains agents i and $i+1$ replaces its values by the average value of the group, h decreases by at least $(\epsilon/2n)^2$. So, there can only be a finite number of such steps. Hence, eventually, all the x_j are within tolerance ϵ of the desired value.

Detecting Termination of Computations

Our problem is to design a system that samples sensors so that computations on a set of sensor values terminates before the sensors are sampled again. Given no constraints on how long the environment can disable all agents the only solution is to use a termination detection algorithm. There is a substantial literature on termination detection and we refer to [14]. For some problems we are given constraints on the environment such as that at most k agents and m channels are disabled at a time and are disabled at most for period T. In such cases we can compute an interval τ such that after time τ has elapsed, even in the worst case, the value of the function computed by the system becomes and remains within a specified tolerance of the desired solution.

7. EXPERIMENTAL RESULTS

7.1 Overview

The previous sections developed a methodology for designing algorithms for different types of dynamic networks. The algorithms are designed to work even if the environment disables and enables agents and channels arbitrarily over the short term. Next we explore the question of determining how much of the connectivity offered by the environment should be exploited optimally.

We explore two questions about how group formation impacts rates of convergence. *i)* Should agents tend to form large groups or small groups? Intuition may suggest that algorithms that use large groups converge faster; however, an atomic group operation on a large group is more likely to be aborted than an operation on a small group because a component of the group gets disabled while the operation is in progress. *ii)* How does the connectivity of the graph impact convergence? Intuition suggests that algorithms on densely connected graphs will converge faster than algorithms on sparse graphs because more information is fused at each step in densely connected graphs; we present experiments that help evaluate whether this intuition is correct.

Consider these questions for synchronous time-stepped algorithms. For example, the optimal stepsize for the problem of computing the average of a bag is $1/n$ where n is the maximum degree of a vertex in the connectivity graph. Since the environment may change the graph at each step the agents do not know n; therefore, a safe step size is $1/N$ where N is the maximum number of enabled vertices (which is a bound on the maximum degree). Consider a system in which agents can choose *not* to use all the edges of the graph. For example a pair of agents with an undirected edge between them chooses to form a group consisting of just these two agents and treats all other edges as nonexistent. The step size for a group consisting of a pair of agents is 1. What is more efficient: Larger step sizes on smaller groups or smaller step sizes on larger groups? The problem becomes more complex when the probability of operations being aborted is taken into account. These questions are evaluated by experiments on the problem of computing the average of a bag of values.

7.2 Impact of Group Size on Performance

We first consider the impact of group size assuming that operations are atomic and are never aborted. We evaluate the number of time steps for an algorithm to converge as function of sizes of groups performing atomic operations. At each time step one group of size m is selected randomly from a set of n values; the value of each member of the group is replaced by the average of that group. Fig. 2 shows the number of time steps as a function of group size m for different values of the total size n. As intuition suggests, the number of time steps drops dramatically with group size. When multiple mutually-exclusive groups can execute atomic steps *concurrently* the number of time steps still decreases with group size, but not as dramatically. For example, if a system with 100 agents is partitioned randomly at each step into 50 groups of size 2 each, and all 50 groups replace their values by their averages, the number of steps is over an order of magnitude lower than if only group changes value while the other 98 agents remain unchanged.

Fig. 3 shows how the total number of operations required for the algorithm to converge varies with group size. The number of operations to compute the average of a group is linear with the size of the group. Therefore, the impact of group size is less pronounced when the y axis is the number of operations than when it is the number of time steps.

7.3 Impact of Abortion Probabilities on Performance

The probability that an atomic operation on a group succeeds is a complex function of the size of the group, the connectivity of agents in the group, and the number of primitive operations (such as sending/receiving a single message, and addition or multiplication) required to complete the group operation. Next we focus exclusively on group size; for these experiments we assume that abortion rates are independent of connectivity within a group and the number of primitive

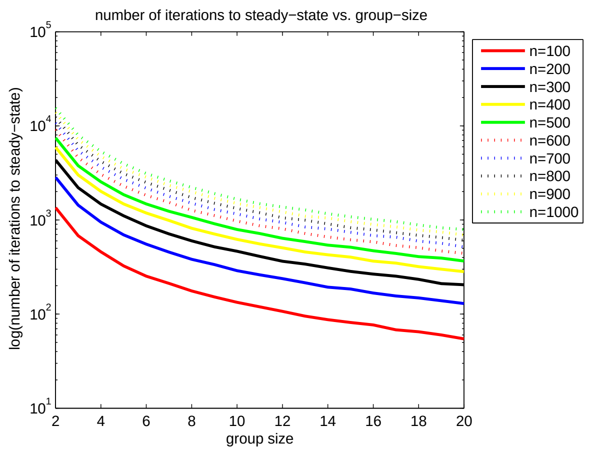

Figure 2: The logarithm of the number of iterations vs. the group size for network size n varying between 100 and 1000

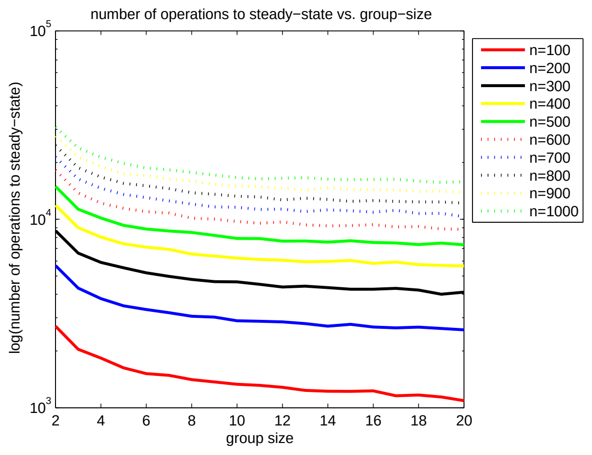

Figure 3: The logarithm of the number of total operations vs. the group size for network size n varying between 100 and 1000

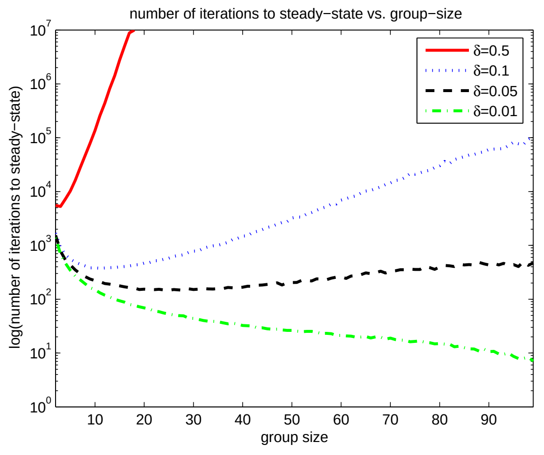

Figure 4: The logarithm of the number of iterations vs. the group size for network size $n = 100$ with δ ranging between 0.5 and 0.01

operations. Let δ be the probability that an agent in a group fails during a group operation. Then the probability that an operation on a group of size n fails is $p(n) = 1 - (1 - \delta)^n$. Figs. 4 and 5 show how the number of time steps and the number of operations varies as a function of δ and group size for a 100-node network. As expected when δ is small (0.01) large group sizes result in fewer time steps. As δ increases to 0.05, a range of group sizes from about 10 to 50 have approximately minimum numbers of time steps, and when δ is large (0.5) any increase in group size is deleterious. These experiments support the intuition that even when the environment forms large groups, greater efficiency can be obtained by agents forming smaller subgroups if the probability of abortion of atomic operations increases with group size.

7.4 Impact of Group Locality on Performance

We study the impact of locality by comparing algorithms on completely connected graphs in which each agent has n neighbors with algorithms on linear graphs in which each agent has at most 2 neighbors. In these experiments we assumed that group operations did not abort. For the completely-connected graph, groups of agents are picked uniformly from the set of agents. For the linear graph two sets of simulations were performed: (a) groups consisting of linear segments of the graph were picked randomly and uniformly at each step, and (b) groups were picked deterministically according to the following rule. Each group of agents is a window — a consecutive sequence of agents — of size n; after a group is selected, the window is moved one position to the right. For example, if the group size is 2, then after a group of agents indexed (0, 1) is picked, a group consisting of agents (1, 2) is picked. As intuition suggests, the number of time steps and the number of operations are dramatically lower for the completely connected graph than for the sparse graph as shown in figs. 6 and 7. What is surprising is that the deterministic algorithm is better than the probabilistic one for the linear graph for small group sizes but worse for large group sizes. Though we have not investigated this phenomenon in depth a possible explanation is as follows.

For the linear graph assume that agent i is connected to agents $i - 1$ and $i + 1$, and consider the case where agent i has value i, for $1 \leq i \leq 10$. Consider the sliding window algorithm with a window size of 2. It sets the value of agents 0, 1 to 0.5 each; then it changes values of agents 1 and 2 from 0.5 and 2 to 1.25 each; then agents 2 and 3 change their values from 1.25 and 3 to 2.125 each, and so on. At each succeeding step the values of agents in the group will differ by more than 1. Consider the random algorithm that picks pairs say 1, 2 and then 8, 9 and then 5, 6; in each of these steps the values in each group differ by 1. Thus, at least initially, the value of the Lyapunov function h decreases more rapidly for the sliding window case. If, by contrast, the number of elements is large, say 1000, and the window size is also large, say 100, then sliding the window by 1 at each step reduces h by less, on the average, than picking a random window of the same size.

8. CONCLUSION

In this paper we have presented models of different types of dynamic sensor networks including those described in terms of differential equations, difference equations and temporal

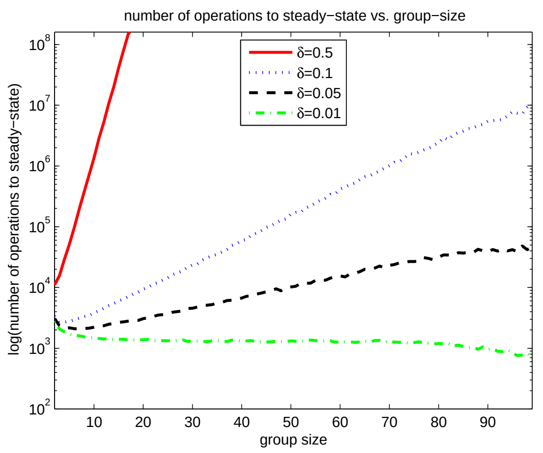

Figure 5: The logarithm of the number of total operations vs. the group size for network size $n = 100$ with δ ranging between 0.5 and 0.01

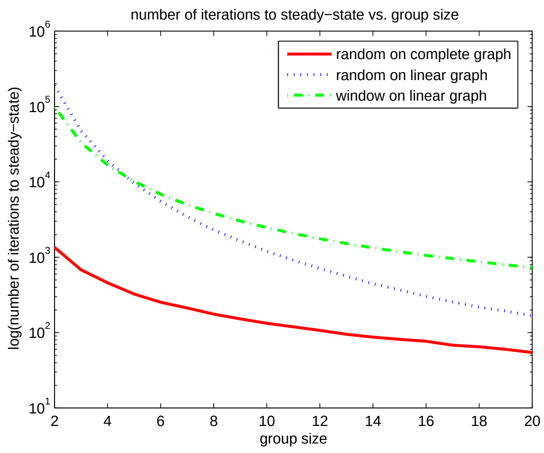

Figure 6: The logarithm of the number of iterations vs. the group size for network size $n = 100$ for several algorithms

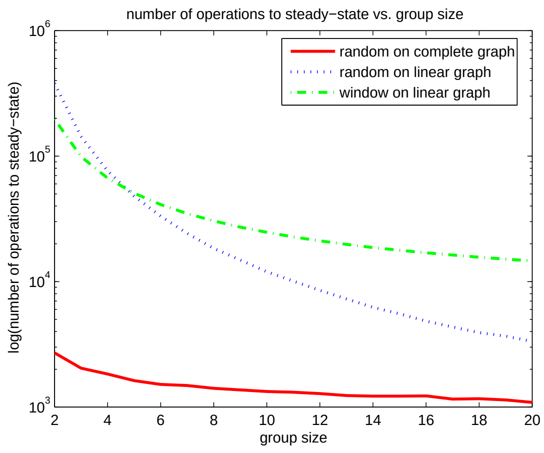

Figure 7: The logarithm of the number of total operations vs. the group size for network size $n = 100$ for several algorithms

logic. We presented a unified methodology for designing algorithms for the range of network types. We also reported on simulation results of performance as a function of different parameters of dynamic networks including group size, probabilities of aborting operations, and locality.

A great deal of further work remains including research in the following directions:

1. The model of the environment is extreme because the environment allows to disable agents and channels arbitrarily, with absolutely no constraints on what the environment does over the short term. Algorithms designed for this extreme environment are robust; however, less extreme environments, constrained to behave in more reasonable ways, may allow for the development of more efficient algorithms. Models of such environments have yet to be explored.

2. Much work remains to be done on analytic and simulation models of performance of sensor networks under different dynamic conditions.

3. Asymmetric game theory in which the environment and agent collection are treated as adversaries allows for the study of systems in which a sensor infrastructure is attacked by opponents.

9. REFERENCES

[1] M. Alanyali, S. Venkatesh, O. Savas, and S. Aeron. Distributed bayesian hypothesis testing in sensor networks. In *Proceedings of the American Control conference*, pages 5369–5374, 2003.

[2] I. Alylidiz, W. Su, Y. Dankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.

[3] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. In *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pages 20–41, 2001.

[4] K. Chandy. Properties of concurrent programs. *Formal Aspects of computing*, 6(6):607–619, 1994.

[5] K. Chandy and B. Sanders. Predicate transformers for reasoning about concurrent programs. *Science of Computer Programming*, 24(2):129–148, 1995.

[6] S. Chatterjee and E. Seneta. Towards consensus: some convergence theorems on repeated averaging. *Journal of Applied Probability*, 14(1):89–97, 1977.

[7] M. DeGroot. Reaching a consensus. *Journal of American Statistical Association*, 69(345):116–121, 1974.

[8] J. Fax and R. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49:1465–1476, 2004.

[9] D. Goldschlag. *Mechanically Verifying Concurrent Programs*. PhD thesis, University of Texas at Austin, 1992.

[10] Y. Hatano and M. Mesbahi. Agreement over random networks. *IEEE Transactions on Automatic Control*, 50(11):1867–1872, 2005.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor network. In *Proceedings of the sixth International Conference on Mobile Computing and Networking*, pages 56–67, 2000.

[12] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomuos agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[13] S. Kumar, D. Shepherd, and F. Zhao. Collaborative signal and information processing in micro-sensor networks. *IEEE Signal Processing Magazine*, 19(2):13–14, 2002.

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

[15] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[16] M. Mehyar, D. Spanos, J. Pongsajapan, S. Low, and R. Murray. Distributed averaging on asynchronous communication networks. In *Proceedings of the forty-fourth IEEE Conference on Decision and Control*, pages 7446–7451, 2005.

[17] R. Olfati-Saber and R. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, September 2004.

[18] T. Pham, D. Scherber, and H. Papadopoulos. Distributed source localization algorithm for acoustic ad hoc sensor networks. *Sensor Array Multichannel Signal Process Workshop*, June 2004.

[19] H. Qi, P. Kuruganti, and Y. Xu. The development of localized algorithms in wireless sensor networks. *Sensors*, pages 286–293, 2002.

[20] W. Ren, R. Beard, and E. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005 American Control Conference*, pages 1859–1864, 2005.

[21] A. Rodriguez-Angeles. Cooperative synchronization of robots via estimated state feedback. In *Proceedings of the fourty-second IEEE Conference on Decision and Control*, pages 1514–1519, 2003.

[22] D. Scherber and H. Papadopoulos. Locally constructed algorithms for distributed computations in ad hoc networks. In *Proceedings of the third international symposium on Information Processing on Sensor Networks*, pages 11–19, 2004.

[23] D. Scherber and H. Papadopoulos. Distributed computation of averages over ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 23(4):776–787, April 2005.

[24] K. Soharabi, J. Gao, V. Ailawadho, and G. Pottie. Protocols for self organization of a wireless sensor network. *IEEE Personal Communication Magazine*, 7(5):16–27, 2000.

[25] D. Spanos, R. Olfati-Saber, and R. Murray. Dynamic consensus on mobile networks. In *Proceedings of the sixteenth IFAC world congress*, 2005.

[26] A. Tahbaz-Salehi and A. Jadbabaie. Consensus over random networks. *Submitted to IEEE Transactions on Automatic Control*, 2006.

[27] T. Vicsek, A. Czirok, E. B. Jacob, I. Cohen, and O. Schochet. Novel type of phase transitions in a system of self-driven particles. *Physical Review Letters*, 75:1226–1229, 1995.

[28] C. Wu. Synchronization and convergence of linear dynamics in random directed networks. *IEEE Transactions on Automatic Control*, 51(7):1207–1210, 2006.

[29] L. Xiao and S. Boyd. Fast linear iterations for distributed averaging. In *Proceeding of the forty-second IEEE Conference on Decision and Control*, pages 4997–5002, 2003.

[30] L. Xiao, S. Boyd, and S. Lall. A scheme for asynchronous distributed sensor fusion based on average consensus. In *Proceeding of the fourth international conference on Information Processing in Sensor Network*, pages 63–70, 2005.

[31] Y. Xu, H. Qi, and P. Kuruganti. Distributed computing paradigms for collaborative processing in sensor networks. *IEEE Globecom*, pages 3531–3535, 2003.

[32] F. Zhao and L. Guibas. *Wireless sensor networks: an information processing approach*. Morgan Kaufmann Publishers, 2004.

[33] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.