2006 Conference on Decision and Control (CDC)
http://www.cds.caltech.edu/~murray/papers/2006n_glm06-cdc.html

When are Distributed Algorithms Robust?

Vijay Gupta, Cedric Langbort and Richard M. Murray

Abstract— **In recent years, numerous distributed algorithms have been proposed which, when executed by a team of dynamic agents, result in the completion of a joint task. However, for any such algorithm to be practical, one should be able to guarantee that the task is still satisfactorily executed even when agents fail to communicate with others or to perform their designated actions correctly. In this paper, we present a concept of robustness which is well-suited for general distributed algorithms for teams of dynamic agents. Our definition extends a similar notion introduced in the distributed computation literature for consensus problems. We illustrate the definition by considering a variety of algorithms.**

I. Introduction

Many algorithms have been presented in the last few years to solve problems as varied as average consensus [3], rendezvous [2] and sensor coverage [4] when there are multiple cooperating agents present. Some of the problems (such as average consensus) do not even make sense without many agents being present. Even so, in all such applications, the hope is that as more and more agents are introduced, the underlying problem (say computing the average temperature across a factory floor) is solved to a greater accuracy.

For any such algorithm to be practical, the failure of one agent to perform its designated duties should not imperil the joint task. In this note, we introduce the notion of robustness to agent failure which has been largely ignored in the control community. In any application involving huge numbers of agents, a typical component will be cheap and off-the-shelf. Such components will have high failure rates and the algorithm should be able to deal with such failures. We take a first step towards defining this concept and study some common algorithms for their robustness properties. Our definition ties in with a similar concept studied in the distributed computation literature (see, e.g., [6] for a good overview). We will show that distributedness in algorithms does not inherently lead to robustness. As an example, the recently proposed average consensus algorithm [3] is non-robust in the sense that a single agent failing to update its values according to the algorithm will lead to no agent converging to the desired mean value. To make such algorithms robust, in general, we need to ensure that agents receive enough information from their neighbors to be able to detect and isolate faulty agents. This point is of interest while designing multi-agent systems and algorithms.

The paper is organized as follows. We begin by setting up a framework for studying algorithms executed by teams of dynamic agents. We then define the notions of agent failure and robustness and illustrate the definitions by studying various algorithms. We compare some algorithms that fulfill the same task yet display different robustness properties and identify possible means of making an algorithm robust. We end with some possible avenues of future work.

Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA {vijay@cds.,clangbort@ist.,murray@}caltech.edu.
Research supported in part by the AFOSR grant F49620-01-1-0460 and in part by NSF grant CCR-0326554 for the first author.

II. Basic Framework

For the rest of the discussion, we will concentrate only on discrete-time algorithms and synchronous networks.

Definition 1: (A Controlled Agent:) We define an agent as a collection of 4 quantities $(X,\, U,\, X_0,\, f)$.

1) $x(k) \in X$ is the *state*; X represents the state space.
2) $u(k) \in U$ is the *control input*; U is the input space.
3) $x(0) \in X(0)$ is the *initial condition*, where $X(0) \subset X$ is the set of allowable initial states.
4) $f : X \times U \to X$ is a map that defines the *dynamics* of the agent.

In words, an agent has a state $x(k)$ at time k. Given a control input $u(k)$, the state evolves according to the dynamics f, i.e., $x(k+1) = f\left(x(k), u(k)\right)$. As an example, the agent has linear dynamics if $f\left(x(k), u(k)\right)$ is of the form $A(k)x(k)+B(k)u(k)$ where $A(k)$ and $B(k)$ are given. The state space X can in general be a continuous space (such as $\mathbf{R}^n$) or a discrete space (such as nodes of a graph).

Definition 2: (Network of Controlled Agents) (following [1]): We define a network of N agents using three quantities $(I, \mathcal{A}, \mathcal{G}_{comm})$.

1) $I = \{1, \cdots, N\}$ is the set of unique identifiers for each of the N agents.
2) $\mathcal{A} = \{A_i\}_{i \in I}$ is the set of controlled agents. Each agent A_i is in turn defined as in definition 1. We will refer to the state of the i-th agent at time k as $x_i(k)$, the control input as $u_i(k)$ and the corresponding sets of allowed values as X_i and U_i respectively.
3) $\mathcal{G}_{comm}$ is the set of allowed communication graphs. At each time step k, the communication graph $\mathcal{E}_{comm}(k)$ over N nodes is an element of $\mathcal{G}_{comm}$. Every node is identified with the identifier i corresponding to a unique physical agent. The edges in the graph represent communication edges in the network. Thus, if the pair (i, j) is an edge in $\mathcal{E}_{comm}(k)$, the agents with identifier j can communicate with the agent with identifier i at time step k .

We will assume *undirected* graphs. Agent i is a neighbor of agent j if the two can communicate. Note that the word communication includes any means of gathering information about the state of another agent (e.g., through sensing). To fully characterize a networked system, we need to also define communication and control laws according to which

the agents choose the messages transmitted to the neighbors and the control inputs for their own dynamics. However, for our present purpose, the above two definitions suffice.

There might be additional variables involved in the problem specification, which we refer to collectively as *environmental variables* and denote by the set V. For example, these can pertain to the locations of obstacles when the agents are robots moving in an area. Similarly for algorithms which assume a fixed and given communication graph, the graph is an environmental variable.

We now define a cooperative task to be carried out by the agents. We will define a task in terms of a cost function.

Definition 3: (Cooperative Task) A cooperative task is a cost function C that depends on the state trajectories of all the agents, the control inputs applied by them, their initial conditions and possibly some environmental variables.

$$C : \prod_i \{x_i(k)\}_{k=0}^{\infty} \times \prod_i \{u_i(k)\}_{k=0}^{\infty} \times \prod_i x_i(0) \times V \mapsto \mathcal{R}^+.$$

Note that for a task that is informally described in words, say 'rendezvous', there might exist many choices of possible cost functions. We will associate a separate task with each cost function. The aim of any algorithm that carries out the task is to minimize the cost function.

Definition 4: (Cooperative Algorithm) A cooperative algorithm is a choice of communication and control laws for every agent.

Note that the cost of the underlying cooperative task the algorithm is trying to solve and the cost function that the algorithm is actually minimizing may be different. Also, there can be constraints on the control and communication laws that an algorithm must satisfy. As an example, for robotic agents moving in physical space, it might be the case that only a specific function of the state of the neighbors can be sensed (output-measurable). Hence, the messages and the control inputs have to depend on that function.

Examples:

1) Average Consensus [3]: Consider N agents, each of which is provided a scalar value. The arithmetic mean of the values across the agents is m. The task is to ensure that on termination, each agent has the value m. The i-th agent has scalar dynamics of the form

$$x_i(k+1) = x_i(k) + u_i(k),$$

with $x_i(0)$ given. There are many cost functions that can be considered. We will consider the cost function

$$C = \lim_{k\to\infty} \left(\sum_{\text{all nodes } i} \left(x_i(k) - \frac{1}{N} \sum x_i(0) \right)^2 \right).$$

This is clearly a function of the state values of the agents and their initial conditions and thus fits in our framework. The communication graph is fixed and given. The only requirement on the edge set is that the graph be connected. Let h be a small positive number. Then, the i-th agent applies the control input

$$u_i(k) = -h \sum_{j:j \text{ is a neighbor of } i} (x_i(k) - x_j(k)).$$

Note that the algorithm minimizes C by minimizing the cost function

$$C_{algo} = \lim_{k\to\infty} \left(\sum_{(i,j) \text{ being neighbors}} (x_i(k) - x_j(k))^2 \right),$$

with the constraint

$$\sum_i x_i(k) = \sum_i x_i(0).$$

If all the agents are functional, it can be proven that minimizing C_{algo} yields the solution that minimizes the task cost C as well. This algorithm is similar to the rendezvous algorithm without connectivity constraint proposed in [2] and is related to works based on Vicsek's model (see [7] for an overview).

2) Sensor Deployment: This problem and its solution have been widely studied. We adopt the algorithm that is described in [4]. The basic problem is for N agents to position themselves in a convex region Q such that the total distance from each point in the region to the nearest agent (possibly weighted by a non-negative density function) is minimized. The agents once again have first order dynamics. They are assumed to move in the convex region Q. Thus, $X_i = Q$ for every agent i. The cost function is defined as

$$C = \lim_{k\to\infty} \int_Q \min_i |q - x_i(k)|_2^2 \phi(q) dq,$$

where $\phi(x)$ is a density function that has a non-negative value at all points x in Q. In addition to the state values, the cost also depends on the region Q and the function $\phi(x)$, which are given environmental variables. The network considered in [4] is the Delaunay graph. The control input is designed so that each agent moves towards the centroid of its Voronoi cell. The details are given in section III-B of [4].

Failure Modes and Robustness: One way to characterize an algorithm is through the value of the task cost function C that it achieves. Denote the performance cost achieved by the algorithm by $\mathcal{PC}$. Since the cost function C can be a function of the initial conditions $x_i(0)$ and the values of the environmental variables, so can be $\mathcal{PC}$. To characterize the algorithm, we can get rid of this dependence in two ways.

1) We can consider the *average* cost, $\mathcal{PC}_{avg}$ obtained by averaging $\mathcal{PC}$ as the initial conditions and values of the environmental variables are chosen from a given set S using a given probability distribution function.
2) We can consider the *worst case* cost $\mathcal{PC}_{wc}$ which is obtained by computing the supremum of the $\mathcal{PC}$ as the initial conditions and values of the environmental variables are varied across a set S.

In general, the performance cost will also depend on the number of agents. To show this dependence explicitly, we will sometimes denote the performance cost by $\mathcal{PC}_{avg}(N)$ or $\mathcal{PC}_{wc}(N)$ if N agents are present.

Before defining the key property of robustness, we need to define an agent failure. During the execution of an algorithm, an agent may stop functioning in many ways. When an agent fails, it alters the control law and the communication law that it follows. We can define some failure modes as follows:

1) Failure mode 1: An agent may fail by simply ceasing to communicate with other agents. This is the most popular agent failure model considered in the literature. In the language of [6], this is similar to saying that the process suffers from a stopping failure.
2) Failure mode 2: An agent fails by setting its state value $x_i(k)$ to a constant. Thus, the control input $u_i(k)$ that ensures $x_i(k+1) = x_i(k)$ is used at every time step k. The constant state value can be any value in the set X_i. The messages that a failed agent transmits to its neighbors also assume constant values for all time k.
3) Failure mode 3: The agent alters the control input to set its state at every time step k to an arbitrary value in the set X_i. The sequence of the values can be chosen maliciously so that the other agents are hindered in the pursuit of the cooperative task. The messages a failed agent transmits are also chosen arbitrarily. This is akin to the way agents fail as described in [5] and is referred to as the Byzantine failure mode in [6].

In the language of [5], the assumption that any communication from a failed agent is also affected according to the failure mode means that agents communicate "orally" and not through "signed messages". This list is not exhaustive and other modes of failure can readily be thought of.

When a given number p out of a total of N agents executing a certain algorithm fail according to a certain mode, the situation is as if the p agents follow a new control and communication law while the remaining $N-p$ agents follow the original laws. We can calculate the performance of this new algorithm. We now define robustness of an algorithm with respect to a particular agent failure model.

Definition 5: (Robustness of an Algorithm): Consider an algorithm being executed on a system of N agents out of which p agents fail according to a particular failure model. Denote the worst-case performance cost achieved through the remaining $N-p$ agents as $\mathcal{PC}_{wc}(N,p)$ where the supremum is also taken over all groups of p agents that can fail. An algorithm is said to be worst-case robust to a particular failure mode up to p agents if

$$\mathcal{PC}_{wc}(N,p) = O\left(\mathcal{PC}_{wc}(N-p)\right).$$

If $\mathcal{PC}_{wc}(N,p) = \Omega\left(\mathcal{PC}_{wc}(N-p)\right)$ but $\mathcal{PC}_{wc}(N,p) \neq \Theta\left(\mathcal{PC}_{wc}(N-p)\right)$, the algorithm is said to be worst-case non-robust[1].

[1]We say $f(x) = O(g(x))$ iff $\exists$ numbers x_0 and $M > 0$ such that $|f(x)| \leq M|g(x)|$ for $x > x_0$. $f(x) = \Omega(g(x))$ iff $\exists$ numbers x_0 and $M > 0$ such that $|f(x)| \geq M|g(x)|$ for $x > x_0$. Finally, iff $\exists$ x_0, $M_0 > 0$ and $M_1 > 0$ such that $M_1 g(x) \geq f(x) \geq M_0 g(x)$ for $x > x_0$ then $f(x) = \Theta(g(x))$

- If instead of the worst case performance costs, we consider the average performance costs $\mathcal{PC}_{ave}(.)$ (however, while still taking the supremum over the p agents that fail), we obtain the definition of average case robustness. While the worst case robustness tells us if the algorithm will perform correctly for *any* set of initial conditions (similar to the case in robust control), average case robustness guarantees that the algorithm will perform correctly *on an average*. We can also talk about almost sure (a.s.) robustness when the algorithm is worst case robust as the initial conditions and values of the environmental variables are varied across a set S, except on a region with measure zero.
- Strictly speaking, the definitions given above pertain to the robustness over the set S over which the initial values and the environmental variables vary.
- The basic intuition behind the definition is that a distributed algorithm should lead to better performance as the number of agents increases. We can expect a hit in the performance some agents fail. However, if we calculate the performance loss in two situations:
  - N agents were present to begin with but p of them failed, and
  - Only $N-p$ agents were present to begin with, (Equivalently, N agents were present and p failed but they were detected and removed)

  then the rate at which adding functional agents decreases the cost should not be adversely affected. In other words, the impact due to agents failing should not increase as more functional agents are added.

We note the following properties that follow from the definitions. We present the proofs for worst-case robustness. The proofs for average-case robustness are similar.

Proposition 1: If an algorithm is non-robust for p failed agents to failure mode 2, it is non-robust to p failed agents to mode 3. Similarly an algorithm robust for p failed agents to failure mode 3 is robust for p failed agents to mode 2.

Proof: Let the control inputs used in the calculation of the performance cost for failure mode 3 be given by $\{u_i(k)\}_3$ for agent i and the messages sent be given by $\{m_i(k)\}_3$. Similarly, let the control inputs used in the calculation of the performance cost for failure mode 2 be given by $\{u_i(k)\}_2$ for agent i and the messages sent by $\{m_i(k)\}_2$. Consider the choice of the control inputs. The set in which the control inputs are allowed to vary for mode 3 also contains as a particular element $\{u_i(k)\}_2$. Since, by definition, the cost in mode 3 is maximized by $\{u_i(k)\}_3$; in particular, the cost achieved by using $\{u_i(k)\}_2$ is not more than when $\{u_i(k)\}_3$ is used. But the cost achieved when $\{u_i(k)\}_2$ is used is the cost in failure mode 2. Thus,

$$\mathcal{PC}_{wc}(N,p)_{\text{ failure mode 3}} \geq \mathcal{PC}_{wc}(N,p)_{\text{ failure mode 2}}.$$

If the algorithm is non-robust to failure mode 2, there exists a constant c such that

$$\mathcal{PC}_{wc}(N,p)_{\text{ failure mode 2}} \geq c\mathcal{PC}_{wc}(N).$$

The above two equations together prove that the algorithm is non-robust to failure mode 3 as well. The second part can be proved similarly. ■

However, a similar statement cannot be said for failure modes 1 and 2. Even if an algorithm is non-robust to failure mode 1, it can be robust to failure mode 2.

Proposition 2: If an algorithm is non-robust to failure of p agents in failure mode 3, it is also non-robust to failure of t agents in failure mode 3 where $t \geq p$. Similarly if the algorithm is robust to failure of t agents in failure mode 3, it is also robust to p failures where $p \leq t$.

Proof: Consider the case when p agents fail. Consider the choice of initial conditions, control inputs and messages for the failed agents that corresponds to the worst case of the performance cost. Choose an arbitrary set S of $t-p$ functional agents. For this choice denote the control input that the agent i in the set S of $t-p$ functional agents applies by $\{u_i(k)\}$ and the messages it transmits by $\{m_i(k)\}$. Now, consider the case when t agents can fail. Choose the same initial conditions as the previous case. Let the t agents that fail be chosen such that they consist of the p agents that failed in the previous case and the $t-p$ agents in the set S. Also, let the p agents apply the same control inputs and transmit the same messages as the previous case. Let the i-th agent in set S apply control input $\{u_i(k)\}$ and transmit messages $\{m_i(k)\}$. Thus, the evolution of the system will be identical to the case when only p agents failed. Hence, $\mathcal{PC}_{wc}(N,t) \geq \mathcal{PC}_{wc}(N,p)$ and the result follows. The second part can be proved along the same lines. ■

Proposition 3: Suppose $\mathcal{PC}_{wc}(N) = \Theta(\mathcal{PC}_{ave}(N))$. Then, if the algorithm is worst-case robust to failure of p agents to a particular failure mode, it is also average-case robust to failure of p agents to that failure mode. Similarly of the algorithm is average-case non-robust to failure of p agents to a particular failure mode, it is also worst-case non-robust to failure of p agents to that mode.

Proof: Proof follows from the definitions once we note that $\mathcal{PC}_{wc}(N,p) \geq \mathcal{PC}_{ave}(N,p)$. ■

A similar statement can also be made about the relation between worst-case robustness and a.s. robustness.

Examples: We now illustrate the above definitions using specific algorithms. If the algorithms involve agents moving in physical space, we will model the agents as point masses and ignore issues such as collision avoidance.

a) Average Consensus [3]: Since the algorithm requires connected graphs, we will assume that to be the case as long as no agents fail. For average-case robustness, we will consider the initial conditions to be chosen uniformly over the set $[-1,1]$.

- Assume that the p agents that fail are allowed to be chosen so that the graph of the remaining $N-p$ agents is disconnected. Then, the algorithm is worst-case non-robust to failure mode 1. If the graph remains connected, then the algorithm is worst-case, average-case and a.s. robust to failure mode 1.

  Proof: First consider the case when we allow the graph of the remaining agents to be potentially disconnected. Consider the case when $p = 1$ and let $N = 2m+1$. Choose the graph of N agents as a line. Let the agent i fail such that two distinct connected sub-groups of agents are formed, each with m agents. Also, suppose the initial conditions are chosen such that every agent in the first sub-group has value 1 and every agent in the second sub-group has value -1. Thus, the algorithm will converge with each agent retaining its value, as against converging to the correct mean for the $N-1$ agents, which is 0. Thus,

  $$\mathcal{PC}_{wc}(N,1) \geq \sum_{i=1}^{m} (1)^2 + \sum_{i=1}^{m} (-1)^2 = N-1.$$

  If N agents were present, they would have all converged to the mean as long as the graph was connected. Thus, $\mathcal{PC}_{wc}(N) = 0$. Thus, the algorithm is worst-case non-robust. If the graph remains connected, $\mathcal{PC}_{wc}(N,p) = \mathcal{PC}_{wc}(N) = 0$. Hence, the algorithm is worst-case robust. A similar argument shows that the algorithm is average-case robust. ■

  In a similar manner, it can be proven that the algorithm, if it runs on a l-connected graph, is robust to the failure of $l-1$ agents. A random $G(n,p)$ graph is almost surely (a.s.) l-connected for $p \geq p_l = (\log(n)+(l-1)\log\log(n))/n$, and a.s. not l-connected otherwise [8]. Thus, if the set over which the graph (which is an environmental variable) is allowed to vary is the set of random graphs, the algorithm is average case robust to l failures in failure mode 1 for $p \geq p_l$ and non-robust otherwise.
- The algorithm is worst-case, average-case and a.s. non-robust to failure mode 2.

  Proof: Consider the case when $p = 1$. Let the initial conditions be such that the non-faulty $N-1$ agents have values 0 while the faulty agent has value 1. Thus, the algorithm will converge with each agent achieving the value 1, as against converging to the correct mean for the $N-1$ agents, which is 0. Thus,

  $$\mathcal{PC}_{wc}(N,1) \leq \sum_{i=1}^{N} (1-0)^2 = N.$$

  Since $\mathcal{PC}_{wc}(N) = 0$, the algorithm is non-robust. A similar argument holds for average case robustness. ■

b) Sensor Deployment [4]: Assume the density function $\phi(x)$ to be a constant. For the statements below, we consider the communication graph to be fully connected.

- The algorithm is worst-case robust to failure mode 1 but not to mode 2.

  Proof: Clearly, if p agents fail according to mode 1, the remaining $N-p$ agents will perform exactly as if there were only $N-p$ agents to begin with. Robustness to failure mode 1 is thus obvious. For failure mode 2, let the agents move in the set Q which is a line

segment of unit length. Let all the agents be stationed at one of the ends and the agent closest to the other end fail. Thus, $\mathcal{PC}_{wc}(N,1) \geq \int_{x=0}^{1} x^2 dx = \frac{1}{3}$. When we have N agents present, the cost is obtained by a Voronoi partition of a unit length segment by N agents. This can be shown to be $\mathcal{PC}_{wc}(N) = \frac{1}{12N^2}$. Thus, the algorithm is non-robust. ■

- The algorithm is a.s. robust to failure modes 1 and 2 for p agents failing, where p is any constant.

  Proof: Robustness to failure mode 1 follows from the worst-case robustness. For failure mode 2, consider the case when the agents are deployed along a straight line of unit length. Consider also the case when only one agent fails. Suppose that the failing agent is at the position x and there are N_1 agents in the region $[0, x)$ and $N - N_1 - 1$ in the region $(x, 1]$. Easy algebra shows that given x and N_1 $\mathcal{PC}(N,1) \approx \frac{x^3}{12N_1^2} + \frac{(1-x)^3}{3(2N-2N_1-2)^2}$. Thus, if the agents are deployed according to a uniform distribution, we can show that each typical event will have $\mathcal{PC}(N,1) = \frac{1}{12N^2}$. When N is large, $\mathcal{PC}_{wc}(N,1) \approx \frac{1}{12N^2}$ with high probability. Since $\mathcal{PC}_{ave}(N) = \frac{1}{12N^2}$ as well, the robustness is obvious. For general sets Q, the proof is similar. ■

c) Multi-Sensor Fusion through a Central Node: Let us now consider an example in which there is a central data processing node. Suppose N nodes measure the value of a random variable v with some additive measurement noise. To obtain the global estimate, every node i transmits its local estimate $\hat{x}_i$ with error covariance P_i to a central node. The central node fuses the estimates to obtain the global estimate $\hat{x}$ with the error covariance P given by $P^{-1} = \sum_i (P_i)^{-1}$ and transmits it back to every node. The cost function is $\sum_i \text{trace}(P_{f,i})$ where $P_{f,i}$ is the final error covariance of the i-th node. For testing the robustness we assume a star topology with node 1 being the central node. The algorithm is not worst-case, average-case or a.s. robust to either failure mode 1 or to mode 2.

Proof: We give the proof for failure mode 1 for worst-case robustness. The proof for other cases is similar. First we note that if the error covariance for the local estimate is given by $P_i = P$, then the error covariance for the global estimate will be given by $\frac{P}{N}$ if N agents are present. Thus,

$$\mathcal{PC}_{wc}(N) = (N) \times \text{trace}\left(\frac{P}{N}\right) = \text{trace}(P).$$

Now, consider the case when the agent 1 fails. Then, $P_{f,i} = P_i$. Thus, $\mathcal{PC}_{wc}(N,1) \leq \sum_i \text{trace}\,(P_i) = (N-1)\text{trace}\,(P)$and the algorithm is non-robust. ■ This is a sanity check since the algorithm is intuitively non-robust to failure of the central node.

Discussion: How to make Algorithms Robust: In this section, we give some ideas about how to make algorithms robust. As a case study, we will consider the classical Byzantine Generals problem in which a General needs to transmit a value v to N commanders such that when the algorithm terminates

1) All the functional (or loyal) commanders make the same decision about the value. We are not concerned with the final values of the non-loyal commanders.
2) If the General is functional, all functional commanders receive the correct value.

For ease of exposition, we will also assume that the General is functional. Consider the cost $C = \lim_{k\to\infty} \sum_{i=1}^{N} (x_i(k) - v)^2$, where x_i is the final decision of the i-th loyal commander and N is the number of loyal commanders. We will study the robustness properties of three algorithms that solve the problem. The first algorithm is similar to the average consensus algorithm discussed above. The general is assumed to be node 1. Its state remains at a constant value v that it needs to communicate to others. Every other agent updates its state as

$$x_i(k+1) = x_i(k) - h \sum_{j \neq i} (x_i(k) - x_j(k)),$$

where h is a positive constant designed to make the algorithm converge. It can be easily shown that any initial condition for the agent states is driven to a consensus vector in which every node has the value v. Thus, the algorithm solves the problem provided all the agents are functional. However, let us consider the case when p agents fail according to failure mode 2. In that case it can be shown [9] that, in general, as long as a node has a path from the failed agent that does not include the general, it does not converge to the value v. Thus, the algorithm is seen to be both worst-case and average-case non-robust for any non-zero value of p. Since the algorithm is non-robust to failure mode 2, it is also non-robust to failure mode 3.

The second algorithm was proposed by Lamport et al [5]. They demonstrated that if one-third or more agents fail according to the failure model 3, then no algorithm that solves the above problem exists. For the case of less than one-third agents failing, they gave an algorithm which successfully solves the problem. In the simplest version of the algorithm, the communication graph is assumed to be fully connected. We will also make that assumption. We illustrate Lamport's algorithm with a simple example of one General and three commanders. The algorithm proceeds as:

1) At time step 1, the General transmits its value v to all the commanders.
2) At time step 2, every commander transmits its estimate of what the General transmitted to every other commander.
3) At time step 3, every commander calculates a majority of the messages it has heard, so far, and outputs its estimate of the decision.

If at most one commander can fail, it can be proven that the cost C is still 0. The algorithm can be extended to N commanders and can be studied under slightly less restricted communication requirements than a fully connected graph, e.g., a 3-regular graph. We note that the algorithm fits in our framework and that it is both worst-case and average-case robust to failure mode 3.

The third algorithm involves including a fault-detection step in algorithm 1. For node i, let $\mathcal{N}_i$ be the neighbor set of i and N_i be its cardinality. In this algorithm, when agent i communicates with agent j at time k, it transmits four quantities: $x_i(k)$ (denoted by $a_i(k)$), $x_i(k-1)$ (denoted by $b_i(k)$), $\sum_{l\in\mathcal{N}_i} x_l(k-1)$ (denoted by $d_i(k)$) and N_i. Given these quantities, each node carries out the following checks:

1) It checks if $a_i(k-1) = b_i(k)$.
2) It checks if $a_i(k) = (1 - hN_i)b_i(k) + hN_i d_i(k)$.

If both these checks are successful, it carries out the same step as the average consensus algorithm, otherwise it identifies the node i as faulty and disregards it from that time on. We will consider the case of one agent failing in failure mode 2. If the failing node disconnects the network into 2 parts, there is no hope for an algorithm to be robust. We will assume the network is at least 2-connected. The general is again node 1. Without loss of generality, let node 2 fail. The following can easily be proved.

1) If $\forall k$, node 2 transmits $a_i(k) = a$, $b_i(k) = b$, $d_i(k) = d$, and $N_i = N$ then to avoid detection, $a = b$.
2) Moreover, to avoid detection by some node j_0, $x_{j_0}(k) = N_i a - d$. Thus, unless two non-faulty nodes have the same state value at all times, at least one will be able to detect the fault in node 2.
3) To ensure that $x_{j_0}(k)$ remains constant, it must be true that $\sum_{l\in\mathcal{N}_{j_0}, l\neq i} x_l(k) = N_{j_0}(Na - d) - a$.

Note that the last two conditions define two surfaces parameterized by the values of $\{x_l(k)\}_{l\neq i}$ in the (a, d, N) space where the faulty values transmitted must lie for the failing agent to go detected. Similarly the condition that the sum of the state values of all neighbors of j_0 remains constant places an algebraic condition on the state values of all other nodes of the network and defines another surface. Now, it is certainly possible to come up with initial conditions that satisfy the above constraints. As an example consider the topology in which the edges (1,2), (1,5), (2,4), (3,4), (3,5) and (4,5) are present. Node 5 is the general with value $4m-n+1$ while node 1 fails by transmitting values $a = m$, $d = n$ and $N_1 = 2$. The initial values of nodes 2, 3 and 4 are $3m - 1$, $2m - 3n + 1$ and $2m - n$ respectively. Then, the nodes 3 and 5 will be able to detect that node 1 is faulty but not node 4. Also, the nodes will never agree on the value of node 1. We can add any number of nodes such that they transmit only to nodes 3 and 5 with the same initial conditions as node 3. Thus, they too will never reach node 1's value and the algorithm is worst-case non-robust. On the other hand, if we choose the initial conditions randomly from a uniform distribution over the interval $(0, 1)$, the probability that the three surfaces will intersect is small. Thus, the probability that valid values of a, d and N_i will exist is also small. Hence, with high probability, all nodes will be able to detect that node 1 is faulty and disregard it. Once the nodes disregard it, the algorithm will run on a connected graph of $N - 1$ functional agents and hence will terminate successfully. The algorithm is thus a.s. robust.

The common feature of the two robust algorithms is that every node received enough information to be able to recover from the effects of the faulty agents. In the second algorithm, enough number of edges were present for every node to obtain multiple copies of the same value. Thus, it could apply majority rule and obtain the correct value. In the third algorithm, information was being transmitted with sufficient redundancy that each node could check if its neighbor was transmitting inconsistently and hence was faulty. This *robustness through information redundancy* is different in nature from the *robustness through cost function* that was exhibited by the sensor coverage algorithm. Whether any algorithm can be made robust through such information redundancy is an open question.

III. Conclusions and Future Work

In this note, we considered networks of distributed agents. We introduced a framework for defining a distributed task and an algorithm. We then defined the key property of robustness and considered some common algorithms for their properties. We saw that distributed algorithms may not be robust and discussed how one might design them to be robust.

This work is but a first step towards a theory of robust distributed algorithms. We are currently working on identifying the properties that the cost function must satisfy for (say a gradient descent type) algorithm to be inherently robust. It will also be nice to have an analytic tool to determine the robustness properties of algorithms and to synthesize robust algorithms systematically.

References

[1] 'On Synchronous Robotic Networks - Part I: Models, Tasks and Complexity Notions,' S. Martinez, F. Bullo, J. Cortes and E. Frazzoli, IEEE Transactions on Automatic Control, Submitted.

[2] 'On Synchronous Robotic Networks - Part II: Time Complexity of Rendezvous and Deployment Algorithms,' S. Martinez, F. Bullo, J. Cortes and E. Frazzoli, IEEE Transactions on Automatic Control, Submitted.

[3] 'Consensus Problems in Networks of Agents with Switching Topology and Time-Delays,' R. O. Saber and R. M. Murray, IEEE Transactions on Automatic Control, 49(9):1520-1533, 2004

[4] 'Coverage Control for Mobile Sensing Networks,' J. Cortes, S. Martinez, T. Karatas and F. Bullo, IEEE Transactions on Robotics and Automation, 20(2):243-255, 2004.

[5] 'The Byzantine Generals Problem,' L. Lamport, R. Shostak and M. Pease, ACM Transactions on Programming Languages and Systems, 4(3):382-401, 1982.

[6] 'Distributed Algorithms,' N. Lynch, Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[7] 'Consensus Seeking, Formation Keeping, and Trajectory Tracking in Multiple Vehicle Cooperative Control,' W. Ren, Doctoral Dissertation, Electrical and Computer Engineering Department, Brigham Young University, August 2004.

[8] 'Random Graphs,' B. Bollobas, Academic Press, London, England, 1985.

[9] 'Decentralized Control of Networks of Dynamic Agents with Invariant Nodes: A Probabilistic View,' P. Hovareshti and J. S. Baras, Technical Report: ISR TR 2004-39, Institute of Systems Research, University of MAryland, College Park.