Decentralized decision-making on robotic networks with hybrid performance metrics

Federico Rossi  Marco Pavone

Abstract—The past decade has witnessed a rapidly growing interest in decentralized algorithms for collective decision-making in cyber-physical networks. For a large variety of settings, control strategies are now known that either minimize time complexity (i.e., convergence time) or optimize communication complexity (i.e., number and size of exchanged messages). Yet, little attention has been paid to the problem of studying the inherent trade-off between time and communication complexity. Generally speaking, time-optimal algorithms are fast and robust, but require a large (and sometimes impractical) number of exchanged messages; in contrast, communication optimal algorithms minimize the amount of information routed through the network, but are slow and sensitive to link failures. In this paper we address this gap by focusing on a generalized version of the decentralized consensus problem (that includes voting and mediation) on undirected network topologies and in the presence of “infrequent” link failures. First, we provide fundamental limitations of performance in terms of communication complexity, which is modeled as the overall size in bytes of all messages exchanged by an algorithm before its completion. Leveraging these results, we present and rigorously analyze a tunable algorithm, where the tuning parameter allows a graceful transition from time-optimal to communication-optimal performance (hence, allowing hybrid performance metrics), and determines the algorithm’s robustness, measured as either the number of single points of failure or the time required to recover from a failure. An interesting feature of our algorithm is that it leads the decision-making agents to self-organize into a semi-hierarchical structure with variable-size clusters, within which information is flooded. Our results make use of a novel connection between the consensus problem and the theory of gamma synchronizers. Simulation experiments are presented and discussed.

I. INTRODUCTION

Decentralized decision-making in robotic networks is a ubiquitous problem, with applications as diverse as state estimation [1], formation control [2], and cooperative task allocation [3]. In particular, the consensus problem, where the nodes in a robotic network have to agree on some common value, has received significant attention in the last decade following the works in [4], [5]. Most recent efforts in the control community have primarily focused on studying the properties and fundamental limitations of average-based consensus, a subclass of the consensus problem in which nodes average their status with their neighbors at each (continuous or discrete) time step [6]. In these works, the dominant performance metric is time complexity, i.e., convergence time. In contrast, the computer science community has mainly focused the attention on the complementary notion of communication complexity, and “communication-optimal” algorithms for selected consensus problems are now known [7].

Despite the large interest in consensus problems in the last decade, little attention has been devoted to the problem of studying the inherent trade-off between time and communication complexity. Generally speaking, the optimization of time-complexity metrics leads to fast and robust decentralized algorithms; however, such algorithms often require a massive number of messages to be exchanged and, consequently, might lead to unacceptable energy requirements for message transmission. In contrast, the optimization of communication-complexity leads to decentralized algorithms that are very economical in terms of exchanged messages, but are extremely sensitive to link failures and converge too slowly in practical scenarios. To appreciate the value of the communication complexity metric, consider the following scenario. A swarm of autonomous underwater vehicles (AUV) are tasked with performing a collaborative mission such as patrolling or tracking of a moving target. Underwater ultrasonic communication has significantly higher energy demands than over-the-water radio transmissions: for instance, in [8], the authors use 18 W to maintain a 16 kbps link (sufficient to stream rich telemetry or low-quality images) with $35^\circ$ (-3dB) antennas over 6500 m. For underwater operations, omnidirectional communication is undesirable because of intersymbol interference caused by multipath propagation, a phenomenon exacerbated by widebeam transceivers [9]. Hence, underwater communication typically relies on directional antennas and communication with $n$ agents requires $n \text{ different}$ messages. Consider, now, an all-to-all communication scheme to reach consensus (as needed by time-optimal flooding algorithms and studied in [8]): in this case, communication with, say, 50 AUVs would require 900 W per AUV, which is impractical (as a comparison, electric motors on modern Remotely Operated Vehicles (ROVs) such as NOAA’s Autonomous Benthic Explorer only draw 100 W in cruise [10]). Hence, in this setting, a time-optimal algorithm such as flooding cannot be implemented. In contrast, a communication-optimal algorithm such as GHS [11] would require $O(\log_2(n))$ messages per AUV, which results in a more practical power requirement of 102 W.

Accordingly, the objective of this paper is to study algorithms for decentralized decision-making that achieve a trade-off between time complexity and communication complexity. Specifically, we focus on a generalized version of the decentralized consensus problem (henceforth referred to as convex consensus), whereby the nodes of a robotic network must agree in a finite number of steps on a common value lying within the convex hull of their initial values. This problem models decision-making problems as diverse as average consensus, leader election, collective data fusion, voting, and mediation. The underlying network topology is indirect, and link failures are infrequent. The contributions of this paper are twofold.
First, we prove tight lower bounds for the communication complexity of the convex consensus problem (in this paper communication complexity is modeled as the overall size in bytes of all messages exchanged by an algorithm before its completion). Leveraging these results, we design and rigorously analyze a tunable algorithm that solves the convex consensus problem in finite time. The tuning parameter allows a graceful transition from time-optimal to communication-optimal performance (hence, allowing hybrid performance metrics), and determines the algorithm’s robustness, measured as either the number of single points of failure or the time required to recover from a failure. The key advantage of our algorithm is that it enables a system designer (by properly “tuning” the algorithm) to achieve the desired trade-off between convergence rate, number of messages exchanged, and robustness of the network. Interestingly, our algorithm leads the nodes in a robotic network to self-organize into a semi-hierarchical structure with variable-size clusters, within which information is flooded. From a technical standpoint, our results make use of a novel connection between the consensus problem and the theory of gamma synchronizers [17].

This paper is structured as follows. In Section II we present some notions from graph theory and the theory of distributed algorithms. In Section III we rigorously state the problem we wish to solve in this paper. In Section IV we provide tight bounds for the communication complexity of the convex consensus problem. In Section V we present a tunable algorithm for the convex consensus problem, and in Section VI we analyze its time and communication complexity as a function of the tuning parameter. In Section VII we present numerical experiments on random geometric graphs that corroborate our results. Finally, in Section VIII we draw our conclusions and we discuss directions for future work.

II. Preliminaries

In this section we first discuss the network model considered in this paper and any relevant performance metrics. Then, we briefly overview the algorithms that are known to be either time- or communication-optimal.

A. Model for Asynchronous Robotic Networks

An asynchronous robotic network with \( n \) agents is modeled as a connected, undirected graph \( G = (V, E) \), where the node set \( V = \{1, \ldots, n\} \) corresponds to the \( n \) agents, and the edge set \( E \subseteq V \times V \) is a set of unordered node pairs modeling the availability of a communication channel. Henceforth, we will refer to nodes and agents interchangeably. Two nodes \( i \) and \( j \) are neighbors if \( (i, j) \in E \). The neighborhood set of node \( i \in V \), denoted by \( N_i \), is the set of nodes \( j \in V \) neighbors of node \( i \).

Each node is internally modeled as a input/output (I/O) automaton, which is essentially a labeled state transition system commonly used to model reactive systems (we refer the reader to [7, ch. 8] for a formal definition). All nodes are identical except, possibly, a unique identifier (UID). The following key assumptions characterize the time evolution of each node in the graph \( G \):

- **Fairness assumption**: the order in which transitions happen and messages are delivered is not fixed a priori. However, any enabled transition will eventually happen and any sent message will eventually be delivered.

- **Non-blocking assumption**: every transition is activated within \( l \) time units of being enabled and every message is delivered within \( d \) time units of being dispatched. Essentially, the fairness assumption states that every node will have an opportunity to perform transitions, while the non-blocking assumption gives timing guarantees (but no synchronization). We refer the interested reader to [7, ch. 8] for a detailed explanation of these assumptions.

Communication links may experience stopping failures, which are modeled as the deletion of an edge in \( G \). Communication links may go offline but not come back: messages across the link at the time of failure are dropped. Agents on both sides of the link are notified of the failure immediately.

We assume that new links cannot be added to the network during execution; hence, the network is static. Finally, we assume that each agent knows at least an upper bound \( \bar{n} \) on the number of nodes in the network, where \( n \leq \bar{n} \leq n + H \), \( H \in \mathbb{N} \). The role of the parameter \( H \) will be made clearer in Section V. Arguably, this assumption is natural in many engineering applications, where the initial number of nodes is known but a small number of nodes may fail during deployment.

B. Time and Byte Complexity

Let \( P \) be a problem to be solved by the nodes in \( G \); more formally, \( P \) represents the task of computing a computable [12] function of the initial values of the I/O automata in the network \( G \). Let \( A \) be the set of decentralized algorithms able to compute the function corresponding to a problem \( P \) on the I/O automata in \( G \). In this paper, a decentralized algorithm is an algorithm executed on each node (of course, nodes can exchange messages). The decentralized algorithm correctly computes a given function if each node outputs the correct function of a certain input, given by the union of all agents’ inputs, and terminates [7]. Let \( G \) be a set of graphs with node set \( V = \{1, \ldots, n\} \), let \( K(G) \) be the set of initial conditions for the I/O automata for a given graph \( G \in G \) (independent of algorithms), and let \( F(a, k, G) \) be the set of fair executions for an algorithm \( a \in A \), set of initial conditions \( k \in K(G) \), and graph \( G \in G \) (a fair execution is an execution of an algorithm that satisfies the fairness and non-blocking assumptions stated above).

The following definitions naturally capture the notions of time complexity and communication complexity and are widely used in the theory of distributed algorithms [7].

1) **Time complexity**: Time complexity is defined as the infimum worst-case (over initial values and fair executions) completion time of an algorithm. Rigorously, the time complexity for a given problem \( P \) with respect to the class of graphs \( G \) is

\[
TC(P, G) := \inf_{a \in A} \sup_{G \in G} \sup_{k \in K(G)} \sup_{\alpha \in F(a, k, G)} T(a, k, \alpha, G),
\]

where \( T(a, k, \alpha, G) \) is the first time when all nodes have computed the correct value for the function associated with problem \( P \) and have stopped. The order of the inf-sup operands in the above expression is naturally induced by our definitions. By dropping the leading \( \inf_{a \in A} \), one recovers the time complexity of a given algorithm \( a \) for a given problem \( P \). In our asynchronous setting, time complexity is expressed in multiples of \( l + d \), defined in section II-A (see also [7]). We will henceforth refer to \( l + d \) as a time unit.
2) **Byte complexity:** In many instances, message size plays a critical role in the energy needed for information transmission. To capture this aspect, in this paper we define communication complexity as the infimum worst-case (over initial values and fair executions) overall size (in bytes) of all messages exchanged by an algorithm before its completion. Accordingly, we will refer to such notion of communication complexity as byte complexity. Rigorously, the byte complexity for a given problem $P$ with respect to the class of graphs $\mathcal{G}$ is

$$BC(P, \mathcal{G}) := \inf_{a \in A} \sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}(G)} \sup_{a \in \mathcal{F}(a, k, G)} B(a, k, \alpha, G),$$

where $B(a, k, \alpha, G)$ is the overall size (in bytes) of all messages exchanged between the initial time and $T(a, k, \alpha, G)$.

In this paper we are specifically interested in the asymptotic growth of TC and BC; accordingly, we briefly review some useful notation for asymptotic performance. For $f, g : \mathbb{N} \rightarrow \mathbb{R}$, $f \in O(g)$ (respectively, $f \in \Omega(g)$) if there exists $N_0 \in \mathbb{N}$ and $k \in \mathbb{R}_{\geq 0}$ such that $f(N) \leq k g(N)$ for all $N \geq N_0$ (respectively, $f(N) \geq k g(N)$ for all $N \geq N_0$). If $f \in O(g)$ and $f \in \Omega(g)$, then we use the notation $f \in \Theta(g)$.

### C. Discussion

Energy consumption is a limiting factor for a variety of cyber-physical systems, for example robotic swarms and wireless sensor networks, and wireless communication is often one of the main contributors to battery depletion (see example in Section I). The notion used in this paper for communication complexity, i.e., byte complexity, is a reasonable proxy for the energy cost of message transmission in settings where:

- The energy cost of a message is independent of the receiver’s distance (although the neighborhood of the sender typically is a function of the range of the communication equipment).
- The cost of a message linearly depends on the payload size.
- The cost of sending the same piece of information to $k$ agents is $k$ times the cost of a single message (which is in general not true for broadcast communication models).

As for the second condition, linear dependence of the cost on payload size holds true for lightweight protocols whose handshakes, headers and acknowledgements are small with respect to the actual payload. As for the third condition, the exclusion of broadcast protocols is justified in settings where the nodes in the robotic network are equipped with efficient narrow-band, high-gain mechanically or electronically steerable antennas, or use of a network protocol that does not implement broadcasts. The proper modeling of “energy complexity” for broadcast models is beyond the scope of this paper and is left for future research. If the above three conditions are verified, the minimization of byte-complexity is aligned with the goal of minimizing the energy cost for message transmission. Also, the minimization of byte complexity is aligned with the goal of minimizing the likelihood of communication jamming in adversarial environments, of minimizing traceability in covert applications, and of optimizing the availability of the frequency spectrum.

Finally, we remark that, if the energy cost of a message is independent of the message size, message complexity, i.e. the overall number of messages exchanged, is a more appropriate proxy for energy cost. This may be the case whenever the cost of a message is dominated by the fixed cost to establish a connection, handshake, exchange connection parameters, and frame the payload.

### III. Problem Formulation

In this paper, we focus on finding decentralized algorithms to solve convex consensus problems on hierarchically computable functions, which fulfill hybrid time/communication performance requirements. In this section, we first rigorously define the notion of convex consensus. Then we define the class of hierarchically computable functions. Finally, we formally state the problem we wish to solve.

#### A. Convex Consensus Problem

The convex consensus problem is defined as follows:

**Definition III.1** (Convex consensus). Consider $n$ nodes indexed by $\{1, \ldots, n\}$ arranged in an undirected graph and capable of exchanging information according to the asynchronous network model presented in Section II-A. Each node is equipped with an initial value $k_i \in \mathbb{R}^+$ (representing, e.g., a local measurement of an environmental phenomenon). The goal is for all nodes to agree in a finite number of steps on a common value $k$ lying within the convex hull of the initial values $k_i$, $i = 1, \ldots, n$; in other words, $k$ can be represented as a convex combination of the initial values $k_i$’s, i.e.:

$$k := \sum_{i=1}^{n} c_i k_i, \quad \text{where } c_i \in [0, 1], \text{ and } \sum_{i=1}^{n} c_i = 1,$$

where the weights $c_i$, $i = 1, \ldots, n$, are problem-dependent. In other words, the vector of weights $[c_i]_i$ parameterizes the convex consensus problem. The weights might be unknown to the agents.

The convex consensus problem models a variety of decision-making problems of interest for robotic sensor networks. Some examples include:

- Computation of $\max_i k_i$ (equivalently, $\min_i k_i$), e.g., for leader election [7]. This problem can be represented with the weight choice (assuming there exists a unique maximum or minimum): $c_i = 1$ if $k_i = \max_j(k_j)$ (equivalently, $k_i = \min_j(k_j)$) and $c_i = 0$ otherwise.
- Average consensus, which can be employed to solve problems as diverse as distributed sensing and filtering [1], formation control [4], rendezvous [14] and coverage control [15]. This problem can be represented with the weight choice: $c_i = 1/n$.
- Weighted average consensus, which can be employed for data fusion when information about the confidence of several measurements is available. This problem can be represented with the weight choice: $c_i = 1/(\sigma_i \sum_j (1/\sigma_j))$, where $\sigma_i$ is the uncertainty of each measurement.
- Any logical operation whose outcome lies within the convex hull of the nodes’ initial “opinions” for the policy to follow. If policies are mutually exclusive, this problem can be represented with the weight choice (assuming only one agent proposes the selected policy) $c_i = 1$ if $i$ is the selected policy, $c_i = 0$ otherwise. If the problem admits
a notion of mediation between different policies, $c_i$ can assume problem-dependent values between 0 and 1.

It is important to note three key differences with respect to “standard” average-based models for consensus problems such as those in [6]: (i) convex consensus provides a generalization of average-based consensus, (ii) a solution should be provided in finite time (as opposed to asymptotically converging algorithms), and (iii) an algorithm is not restricted to evolve according to a (possibly discontinuous) differential equation, but can also perform logical operations, establish hierarchical relationships, route messages, and make complex decisions, better exploiting the possibilities offered by the on-board processing capabilities.

B. Hierarchically Computable Functions

Hierarchically computable functions (related to the sensitively decomposable functions as defined in [13]) are defined as follows:

Definition III.2 (Hierarchically computable function). A hierarchically computable function is a function obeying the following property: given the values of a function computed on several disjoint sets of nodes (as opposed to the inputs of every node), it is possible to compute the function of the union of these sets. Moreover, it is possible to store the result in a string of the same order of magnitude as the size of the string needed to represent a single argument.

Average and weighted average are examples of hierarchically computable functions: given a subset of nodes, their contribution to the consensus value can be represented by their (weighted) average and associated weight. Majority voting on a limited number of options is also hierarchically computable: it is sufficient to store the number of votes obtained by each option. Other examples of hierarchically computable functions include maximum and minimum. Applications as diverse as leader election, distributed filtering and voting can therefore be represented by this class of functions. The name is inspired by the observation that hierarchically computable functions can be computed with messages of small size on a hierarchical structure such as a tree.

C. Problem Formulation

In this paper, we wish to design a decentralized algorithm (as defined in Section II-B) that solves the convex consensus problem and achieves a trade-off between time complexity and byte complexity. Accordingly, the problem statement reads as follows:

Parameterized convex consensus problem: — Let $\mathcal{G}$ be the set of all graphs with node set $V$. Find a decentralized algorithm $a(\tau)$ parametrized by $\tau \in [0, 1]$ that solves the convex consensus problem $P$ with optimal order of growth of $TC(P, G, a(\tau)) = TC(P, G)$ for $\tau = 1$, optimal order of growth of $BC(P, G, a(\tau)) = BC(P, G)$ for $\tau = 0$, and orders of growth $TC(P, G, a(\tau)) < TC(P, G, a(\tau = 0))$ and $BC(P, G, a(\tau)) < BC(P, G, a(\tau = 1))$ for $\tau \in (0, 1)$.

That is, we wish for our algorithm to move from asymptotically time-optimal to asymptotically byte-optimal behavior, at the same time guaranteeing time performance no worse than the byte-optimal algorithm and byte performance no worse than the time-optimal algorithm.

IV. Fundamental limitations of convex consensus

In this section we present some fundamental limitations results for the convex consensus problem.

A. Time Complexity

The time complexity of the convex consensus problem is well-understood: if the consensus function depends on all nodes’ initial values, the diameter of the network $Diam(G)^1$ represents a trivial lower bound on the number of time units required by any algorithm solving the convex consensus problem. A simple flooding algorithm achieves this lower bound; in fact, one can easily show that the time complexity of flooding is $Diam(G)$ time units. Hence, flooding is time-optimal. On the other hand, one can also show that the byte complexity of a flooding algorithm is $O(|E| n \log n)$. This bound is very large and limits the applicability of flooding algorithms to limited-size networks. Note that average-consensus algorithms belong to the class of flooding algorithms.

B. Communication Complexity

In contrast to the case of time complexity, few results are known about communication complexity: these results typically concentrate on specific applications of consensus [7] or specific network topologies [16].

Of particular importance is the GHS distributed minimum spanning tree algorithm [11]: GHS allows the construction of a rooted minimum spanning tree (MST) on synchronous and asynchronous static networks in $O(n \log(n))$ time and $O((n \log(n) + |E|) \log n)$ byte complexity. Once a rooted spanning tree is in place, the root can collect information from all nodes using the tree and compute any consensus function using $O(n)$ messages. Message size depends on the nature of the consensus function under consideration; if messages carry a sender and a receiver ID, their size is $\Omega(\log n)$. Improved versions of GHS such as [13] can yield a time complexity of $O(n)$ with no degradation in byte complexity. Under mild assumptions, the improved GHS algorithm [13] can be shown to be message-optimal for MST construction; if messages carry a sender or receiver ID, byte optimality follows. Its time-complexity, however, is significantly higher than that of the time-optimal flooding algorithms; furthermore, GHS-like algorithms are very fragile with respect to link failures.

In the remainder of this section we present a tight lower bound on the byte complexity of the convex consensus problem. Collectively, this result and the aforementioned bound on time complexity justify the use of flooding and GHS as time-optimal and byte-optimal benchmark algorithms for convex consensus. We divide our analysis into two parts: first we examine dense networks with $\Theta(n^2)$ edges, then we concentrate on sparse networks with $\Theta(n)$ edges. We assume that messages carry two labels identifying the sender and the receiver respectively. Any node may have up to $n - 1$ neighbors: identifiers therefore have size $\Omega(\log n)$. It follows that exchanged messages also have size $\Omega(\log n)$.

1 The diameter of a graph $G$, denoted as $Diam(G)$, is the greatest distance between any pair of nodes; the distance between two nodes is the shortest path between them.
1) Dense networks: In [16], the authors show that any computation problem that requires use of a spanning subgraph of the network requires use of up to $|E| - 1$ edges (and therefore $\Omega(|E| - 1)$ messages) on a certain class of almost complete graphs. Furthermore, any consensus algorithm whose consensus function depends on all nodes’ initial values (which includes hierarchically computable functions) needs to use a spanning subgraph of the network for information to travel from a given node to all other nodes. Accordingly, we have the following lemma.

**Lemma IV.1.** Let $\mathcal{G}$ be the set of all graphs with node set $V$, then $BC(P, \mathcal{G}) \in \Omega(|E| \log n)$.

**Proof:** The work in [16] shows that there exist a class of “almost complete” networks such that, in order to solve the consensus problem on this network, messages of size $\Omega(\log n)$ must be sent across $|E| - 1$ edges. Therefore, there exist $G \in \mathcal{G}$ such that $BC(P, G) = \Omega(|E| \log n)$. It follows that $BC(P, \mathcal{G}) = \Omega(|E| \log n)$.

2) Sparse networks: We show that there exist network topologies with $O(n)$ edges where any asynchronous algorithm solving a convex consensus problem requires transmission of $\Omega(n \log n)$ messages and therefore $\Omega(n \log^2 n)$ bytes. The proof leverages results from [7], which shows that leader election on a ring of size $n$ requires at least $\Omega(n \log(n))$ messages. The proof for the lower bound requires two preliminary lemmas, which are omitted in the interest of brevity and are provided in the Supplementary Material. The first lemma shows that, given an infinite set of I/O automata running a convex consensus algorithm, all but at most one of them can send a message without first receiving any; the second key lemma proves that, for any natural number $r$, there exist infinite pairwise disjoint lines of $2^r$ automata with byte complexity larger than $r^2 - 2^{r-2}$.

The next lemma presents a lower bound for the growth order of byte complexity.

**Lemma IV.2** (Lower bound for byte complexity). Let $\mathcal{G}$ be the set of all graphs with node set $V$, then $BC(P, \mathcal{G}) \in \Omega(n \log^2 n)$.

**Proof:** Consider an execution of an algorithm on a ring $R \in \mathcal{G}$ in which all messages passing through a given communication channel are delayed until the rest of the network becomes silent. The ring behaves as the line in the previous lemma, which shows that at least $n/4 \log(n)$ messages are required to reach consensus: the claim follows.

We are now in a position to characterize the growth order for byte complexity.

**Lemma IV.3** (Lower bound on byte complexity). Let $\mathcal{G}$ be the set of all graphs with node set $V$. The byte complexity of the convex consensus problem is $\Omega((n \log n + |E|) \log n)$, i.e., $BC(P, \mathcal{G}) \in \Omega((n \log n + |E|) \log n)$.

**Proof:** Lemma IV.1 shows that $BC(P, \mathcal{G}) = \Omega(|E| \log n)$; Lemma IV.2 shows that $BC(P, \mathcal{G}) = \Omega(n \log^2 n)$. The claim follows.

**Theorem IV.4** (Order of growth for byte complexity). Let $\mathcal{G}$ be the set of all graphs with node set $V$. The byte complexity of the convex consensus problem is $\Theta((n \log n + |E|) \log n)$, i.e., $BC(P, \mathcal{G}) \in \Theta((n \log n + |E|) \log n)$.

**Proof:** We need to show that there exists an algorithm $a \in \mathcal{A}$ such that

$$\sup_{G \in \mathcal{G}} \sup_{k \in \mathcal{K}} \sup_{n \in \mathcal{F}(a,k)} M(a, k, d, |G|, G) = BC(P, \mathcal{G}).$$

Indeed, the GHS algorithm achieves a byte complexity of $O((n \log n + |E|) \log n)$ [11] with messages of size $\log n$. Therefore the GHS algorithm achieves the lower bound on byte complexity.

**V. A Tunable Algorithm for Convex Consensus**

In this section we present a semi-hierarchical algorithm that solves the parameterized convex consensus problem as defined in Section III-C. The algorithm “gracefully” transitions from a flooding-like, time-optimal behavior to a GHS-like, byte-optimal behavior as a function of a parameter $m \in (1, n)$. The parameter $m \in (1, n)$ is affine to the parameter $\tau \in (0, 1)$ presented in the Problem Formulation. The complexity of the algorithm is characterized in Section VI. Our algorithm is inspired by the gamma synchronizers proposed in [17].

We provide a high-level description of the algorithm in Section V-A. Then, in Section V-B we discuss the algorithm in detail and we provide a proof for its correctness. The pseudocode for the algorithm is provided in the Supplementary Material.

**A. High-Level Description**

Our algorithm operates in four nominal phases compounded by two error recovery routines.

Phase 1 starts by building a forest of minimum weight trees (shown in Figure 1a) of height $O((n/m))$, where $n$ is the number of nodes in the robotic network and $m$ is the algorithm’s tuning parameter. All nodes run a modified version of the GHS algorithm [11]. GHS builds a minimum spanning tree in stages: it grows a forest of minimum weight trees by incrementally merging clusters until they span the entire network. At each stage, nodes belonging to a cluster collectively identify the cluster’s minimum weight outgoing edge; the cluster then absorbs the edge and merges with the tree across it. The algorithm terminates when the tree root is unable to identify a minimum weight outgoing edge because all nodes belong to the same cluster: it then informs all descendants, which stop. Phase 1 of our algorithm differs from the GHS algorithm with respect to the stopping criterion. Specifically, at each stage, the root keeps track of the number of nodes in its cluster: when the cluster size exceeds $(n/m)$, the root stops the tree-building phase and informs its descendants. At this point, other smaller groups may try and join the cluster: they are allowed to do so immediately, at which point they inherit the cluster’s identity and learn that the tree-building phase is complete. When a node discovers that Phase 1 is over, it contacts all its neighbors, excluding its father and children, to inquire whether they, too, are done. When all have replied affirmatively, it starts executing Phase 2.

In Phase 2, tree height is upper-bounded by splitting clusters with too many agents while enforcing a lower bound on tree size. This phase of the algorithm starts at the leaves of each tree. Each node recursively counts the number of its
descendants moving towards the root; agents with more than \(\lfloor n/m \rfloor \) offspring create a new cluster, of which they become the root, and cut the connection with their fathers. The tree containing the original root may be left with too few nodes: the root can undo one cut to guarantee that all clusters contain a minimum number of nodes.

In Phase 3, each tree establishes a certain number of connections with neighbor clusters, as shown in Figure 1b. When a node switches to Phase 3, it contacts all neighbors except for its father and children, inquiring about their cluster UIDs. Upon reception of an inquiry, a node replies as soon as it enters Phase 3 (and is therefore sure of which cluster it belongs to). Information is then convergecast on the tree, starting from the leaves: each node informs the father about which clusters it is connected to (either directly or through its children) and how many connections per cluster are available. Roots also exploit the tree structure to compute their cluster’s consensus function after convergecasting information from their offspring; when they have received information from all offspring, they switch to Phase 4.

In Phase 4, cluster roots communicate with each other through the connections discovered in the previous stage. Conceptually, this phase of the algorithm is simply flooding across clusters. Each root sends a message containing its cluster’s consensus function to each neighbor tree through the connections built in Phase 3. Each message is replicated “a few” times as a protection against link failures. When a root learns new information, it forwards it once to its neighbor clusters (sender excluded) via the same mechanism.

If a link failure breaks one of the trees (as in Figure 1d), the two halves evaluate their size. If either of the two halves is too small, it initiates a search for its minimum weight outgoing edge (MWOE) and rejoins the cluster across it; a splitting procedure guarantees that tree height stays bounded. After failure, all nodes in the affected cluster contact their neighbors to update their routing tables.

When a link outside a tree fails, nodes on the two sides of the failure update their routing tables and notify their cluster roots. Note that, if each inter-cluster message is replicated \(c \) times, up to \(c - 1 \) simultaneous, adversarial failures can occur while the algorithm updates its routing tables without disrupting cluster flooding.

B. Detailed Description and Proof of Correctness

As discussed in Section V-A, the algorithm involves four phases, plus one phase (named Phase OF) to handle inter-tree failures. In the following, we discuss the phases in detail and prove their correctness. Proofs for Lemmas V.1 to V.10 and Lemmas V.12 to V.17 are omitted in the interest of brevity and are provided in the Supplementary Material.

1) Phase 1 (tree building): As explained in Section V-A, Phase 1 is an essentially an implementation of the GHS algorithm with a modified stopping criterion.

The proof of correctness relies on three preliminary lemmas.

Lemma V.1 (Minimum weight tree structure). At the end of Phase 1, each cluster is a tree and contains only edges belonging to the graph’s minimum spanning tree.

Lemma V.2 (Cluster size). At the end of Phase 1, all clusters contain at least \(\lfloor n/m \rfloor \) nodes.

Lemma V.3 (Participation). All nodes eventually join a cluster.

The correctness claim for Phase 1 then reads as follows.

Lemma V.4 (Termination of Phase 1). All nodes are eventually informed of the end of Phase 1.

The trees obtained in Phase 1 are guaranteed to have a height larger than \(\lfloor n/m \rfloor \). Yet this is not sufficient: we wish to bound the number of clusters and the height of each tree. The stopping criterion is a good heuristic but it does not offer worst-case guarantees: one can produce examples (in terms of network topologies and weight distributions) that give rise to a single tree spanning the whole network. This motivates the next phase of the algorithm.

2) Phase 2 (tree splitting): Before executing Phase 2 of the algorithm, each node waits to be sure that all neighbors (excluding its father and children) are in the same phase. Leaves (childless nodes) then send a message to their fathers. The algorithm proceeds recursively from here: once a node has heard from all of its children and made sure that its neighbors are in Phase 2, it can correctly compute the number of its offspring. It then sends this information to its father. If a node learns that it has more than \(\lfloor n/m \rfloor \) offspring, it cuts the connection with its father after letting it know and tentatively declares itself as root (but waits before notifying its offspring). The former father makes a local note of this. Information (i.e., the removed child’s UID, the father’s UID, and the number of removed offsprings) about the cut which removed the least number of children is relayed towards the root during the counting process.

The procedure eventually reaches the cluster’s root. If the number of remaining offspring is higher than a lower bound \(H < n/m, H = \Theta(n/m)\), the root switches to Phase 3 and informs its offspring. These, in turn, approve tentative cuts by sending a message to former children who had declared themselves roots; they then switch to Phase 3 and inform all remaining children of this. Removed children (now bona fide roots) do the same. Each child records the UID of its tree’s
root, which is used as the cluster’s identifier.

If, on the other hand, the tree containing the original root is smaller than $H$, the root asks its offspring to undo the cut that removed the least number of children (identified by the UIDs of father and removed child), then switches to Phase 3. The father of the cut to be undone asks the child to do so and switches to Phase 3. The child notifies its offspring; all other nodes behave as in the previous case.

The proof of correctness for Phase 2 relies on two preliminary lemmas.

Lemma V.5 (Cluster height). At the end of Phase 2, all trees have height lower than \((\lceil n/m \rceil + H) + 2 = \Theta(n/m)\).

Lemma V.6 (Number of clusters). At the end of Phase 2, there are at most \(n/H = O(m)\) clusters.

The correctness claim for Phase 2 then reads as follows.

Lemma V.7 (Termination of Phase 2). All nodes are eventually notified of the end of Phase 2.

3) Phase 3 (inter-cluster links): Each node asks all neighbors, excluding its father and children, to declare their Cluster ID. Each node then waits until it has heard from all children (if any) and all neighbors before informing its father. Nodes maintain two local routing tables: one (the neighbor routing table) relates non-tree neighbors and their cluster, whereas the other (the children routing table) records which clusters each child is connected to (directly or indirectly) and how many connections are available per cluster. When informing its father, a node makes no distinction between direct and children-mediated connections.

The correctness claim for Phase 3 then reads as follows.

Lemma V.8 (Termination of Phase 3). Each node is eventually informed of the correct number of neighbor clusters connected either to it or to its offspring.

4) Phase 4 (inter-cluster flooding): The root of each tree generates a message for each of its neighbors with the value of its cluster’s own consensus function. Each message is replicated \(c\) times, where \(c\) is a user-defined parameter. The root then sends as many copies of each message as possible through its direct connections, stored in its neighbor routing table. Unsent copies of the message are distributed to children proportional to the number of links available, stored in the children routing table. Children do the same: when required to forward a message to a cluster, they send as many copies as possible through their direct connections and divide the rest among their own children according to the number of connections available. When a node receives a message for its cluster, it checks whether it has already received this information. If this is not the case, it forwards the message up the tree; otherwise, it discards it. The first time a root hears new information, it broadcasts it to neighbor clusters via the same mechanism as above, forwarding \(c\) copies of a message with the new information and the origin cluster’s UID. Roots also include the number of their children in their cluster’s information. When a root has heard from \(n - H + 1\) nodes, it terminates: after forwarding new information one last time, it computes the consensus value and informs its offspring.

The proof of correctness relies on a preliminary lemma.

Lemma V.9 (Diffusion of information). In absence of failures, all clusters eventually hear from each other.

The correctness claim for Phase 4 then reads as follows.

Lemma V.10 (Termination of Phase 4). Phase 4 of the algorithm eventually terminates.

We are now ready to prove the correctness of our algorithm.

Theorem V.11 (Correctness of the algorithm). Every node correctly computes the consensus function.

Proof: By Lemma V.7, the network is partitioned in rooted trees: it is easy to see that a convergecast allows the root to correctly compute its cluster’s consensus function. Lemmas V.9 and V.10 show that every root is eventually informed of all clusters’ consensus functions. It follows that, at the end of Phase 4, every root is able to correctly compute the consensus function on the initial values of all nodes. Every nonroot node is then informed of the result with an inter-cluster broadcast.

5) Phase F (recovery from in-tree failure): Upon being notified of a severed connection with a child, a node notifies its root. Conversely, a node losing a connection with its father declares itself a root. If either root has fewer than \(H\) offspring, it sends them a unique cut identifier and initiates a search for the cluster’s minimum weight outgoing edge. If the number of offspring is high enough, the root just sends to the offspring the cut identifier, which includes the old cluster UID and the UIDs of the two nodes immediately upstream and downstream of the failure.

The presence of a failure complicates the search for a minimum weight outgoing edge: nodes notified of the cut at different times may mistakenly believe they belong to different clusters and create cycles within a cluster. To avoid this, nodes disclose the unique cut identifier when looking for the minimum weight outgoing edge: if a node sports the old Cluster ID but is not aware of the cut, it delays the reply until it is informed of it. Once a small cluster finds its minimum weight outgoing edge, it rejoins the cluster on the other side. A splitting procedure, akin to the one outlined in Phase 2, is then initiated to maintain bounded tree height. The procedure is initiated by the node rejoining the cluster and proceeds up to the root: nodes outside this path pass no change in the number of their offspring.

As nodes learn their final Cluster ID, they inform all noncluster neighbors. Neighbors, in turn, update their routing tables and inform their fathers, as in Phase 3. When an unaffected node is contacted by a non-cluster neighbor, it does not immediately notify its father to avoid a multitude of expensive incremental updates: the node waits to hear from all offspring that were connected to the affected cluster (recorded in the children routing table) before updating its father. This way, routing tables are updated in a convergecast. When a root learns about a variation in the topology of neighbor clusters (either because a new cluster is formed or because the number of connections to an existing cluster decreases) it crafts a message with all information it holds and sends it to the modified clusters as in Phase 4. The roots of clusters born or modified after the cut collect information from their children and send messages to all their neighbors, too.

The proof of correctness for this phase relies on three preliminary lemmas.
Lemma V.12 (Cluster height). At the end of Phase F, trees all have height lower than \( \lfloor n/m \rfloor + H + 2 = \Theta(n/m) \).

Lemma V.13 (Number of clusters). At the end of Phase F, there are at most \( n/H = O(m) \) clusters.

Lemma V.14. All nodes in the tree affected by the cut eventually learn about the cut.

We can then state the correctness claim for Phase F.

Lemma V.15 (Inter-cluster connections). Each node is eventually informed of the correct number of neighbor clusters connected to it either directly or through its offspring.

6) Phase OF (recovery from out-of-tree failure): In Phase OF, nodes update their routing tables and recursively notify their fathers following the failure of an inter-cluster link. The proof of correctness relies on a preliminary lemma.

Lemma V.16 (Termination of Phase OF). At the end of Phase OF, all nodes’ routing tables are correct.

We then state the correctness claim for Phase OF.

Lemma V.17 (Resilience to inter-cluster failures). Phase 4 of the algorithm correctly terminates in presence of \( c - 1 \) simultaneous adversarial failures.

VI. COMPLEXITY ANALYSIS

The overall time and byte complexity of the proposed algorithm are reported in Table I. The complexity of byte-optimal GHS and time-optimal flooding (as discussed in Section IV) are reported in Table II. We first give proofs of these results, and then we discuss their implications.

Table I: Time and byte complexity of the proposed tunable algorithm, for a given tuning parameter \( m \in (1, n) \).

<table>
<thead>
<tr>
<th>Phase</th>
<th>Time</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1</td>
<td>( O(n \log(n/m)) )</td>
<td>( O(</td>
</tr>
<tr>
<td>Phase 2</td>
<td>( O(n) )</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Phase 3</td>
<td>( O(n/m) )</td>
<td>( O(</td>
</tr>
<tr>
<td>Phase 4</td>
<td>( O(Diam(G)/n/m) )</td>
<td>( O(</td>
</tr>
<tr>
<td>Phase F</td>
<td>( O(n/m) )</td>
<td>( O(</td>
</tr>
<tr>
<td>Phase OF</td>
<td>( O(n/m) )</td>
<td>( O(n \log n) )</td>
</tr>
</tbody>
</table>

Table II: Time and byte complexity of flooding and GHS.

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flooding</td>
<td>( O(Diam(G)) )</td>
<td>( O(</td>
</tr>
<tr>
<td>GHS</td>
<td>( O(n \log n) )</td>
<td>( O(n \log n +</td>
</tr>
</tbody>
</table>

A. Proofs for Complexity Results

Complexity of Phase 1:

Time complexity: The GHS algorithm proceeds in stages, each requiring at most \( O(n) \) time units. At each phase, the size of the smaller cluster at least doubles. Phase 1 terminates when every cluster is larger than \( n/m \); the time complexity is therefore upper bounded by \( O(n \log n/m) \).

Byte complexity: At each stage, communication within the clusters requires \( O(n) \) messages. Furthermore, \( 2n \) test-accept messages are sent at each stage; each node accepts exactly one connection. Each edge is also rejected at most once during the algorithm. The overall number of messages exchanged is therefore \( O(n \log n/m + |E|) \). Messages carry one Cluster ID (which can be represented with \( \log n \) bytes) at most; hence, their size is upper bounded by \( \log n \).

Complexity of Phase 2:

Time complexity: The algorithm proceeds from the leaves of the trees formed in Phase 1 to their roots and vice versa, akin to a convergecast followed by a broadcast. The time complexity is therefore upper bounded by twice the height of the trees formed in Phase 1, which is itself upper bounded by \( 2n \).

Byte complexity: Each non-root node sends exactly one message to its father, either to notify it of the number of its children or to sever the connection. It receives exactly one message to notify that (i) Phase 2 is over, (ii) authorize a cut, or (iii) revert a cut. The number of messages exchanged is therefore upper bounded by \( 2n \). Messages contain the numbers of offspring and/or Cluster IDs. The message size in Phase 2 is therefore upper bounded by \( O(\log n) \). The resulting byte complexity is \( O(n \log n) \).

Complexity of Phase 3:

Time complexity: Once all nodes are in Phase 3, nodes discover their neighbors’ clusters in two time units at most: one to send inquiries to all non-root-channels, one to collect replies. The subsequent convergecast requires as many time units as the height of the tree, which is upper bounded by \( \lfloor n/m \rfloor + H \). The overall time complexity is therefore \( O(\lfloor n/m \rfloor + H + 2) = O(n/m) \).

Byte complexity: Each non-tree edge is crossed by two messages: an inquiry about the Cluster ID and a reply. Each edge belonging to a tree is charged with one convergecast message. The overall number of messages exchanged is therefore upper bounded by \( 2(|E| - n) \) (inter-cluster) + \( n \) (intra-cluster).

Inquiries on non-tree edges have constant size. Replies contain a Cluster ID: their size is \( \log n \). Messages relayed over the tree store the number of connections with each neighbor cluster: their size is therefore upper bounded by \( m \log n \), since they contain up to \( m \) Cluster IDs. The byte complexity of Phase 3 is \( O(2(|E| - n) \log n + nm \log n) \).

Complexity of Phase 4:

Time complexity: Let us abstract Phase 4 by building an artificial network \( G_c(V_c, E_c) \) composed of \( O(m) \) nodes, each corresponding to one of the existing clusters in \( G \) and labeled accordingly. Nodes in \( G_c \) are connected if at least one edge exists among the corresponding clusters in \( G \). Let us also define a stage time complexity as the time required for information to travel from the root of one cluster to the root of its neighbor. Phase 4 is simply flooding on \( G_c \); the algorithm is therefore guaranteed to terminate in \( Diam(G_c) \) stages. Note that \( Diam(G_c) = O(Diam(G)) \) and \( Diam(G_c) = O(m) \). The time complexity of one stage is upper bounded by \( 2(|n/m| + H) + 1 \): in each stage, information travels away from the root across the cluster in \( O(|n/m| + H) \) time units, then hops from a cluster to the next one and is finally convergecast to the root. The overall time complexity of Phase 4 is therefore \( Diam(G_c)(2(|n/m| + H) + 1) = O(Diam(G)(n/m)) = O(n) \).

Byte complexity: In absence of failures, each of the \( O(n) \) edges belonging to a tree is crossed by information about one
cluster at most twice: once when the cluster learns about the information, once when information is relayed to neighbors. Duplicate information is discarded. The overall byte complexity of intra-cluster messages in Phase 4 is therefore upper bounded by \(2m n \log n\). Each of the \(O(c|E_c|)\) inter-cluster connections is also crossed by information about each cluster once: clusters send new information once after they receive it. The associated byte complexity is \(O(c|E_c| |E_c| \log n)\). The overall byte complexity is therefore \(O(m(n + c|E_c|) \log n)\).

Complexity of Phase F:

Time complexity: All nodes within a tree are informed of a link failure within two cluster heights, i.e., within \(2(\lfloor n/m \rfloor + H)\) time units of the failure. The node downstream of the failure broadcasts the information to its offspring directly, whereas the upstream node informs the root which, in turn, broadcasts information to other nodes. If a tree is found to be too small, a search for the minimum weight outgoing edge is initiated. Any node can be rejected by \(O(H)\) other nodes in the same group at most; furthermore, the first reply may be delayed by as much as \((\lfloor n/m \rfloor + H - 1)\) time units as nodes are informed of the cut. The time complexity of splitting is upper bounded by twice the height of the tree, as in Phase 2. In Phase F, the maximum height of a tree is \((\lfloor n/m \rfloor + 2H)\): before the failure, no tree can be taller than \((\lfloor n/m \rfloor + H)\) and only trees smaller than \(H\) rejoin an existing cluster. Once the tree has been reformed, it updates its neighbors about its Cluster ID and rebuilds the internal routing tables. Neighbors update their own routing tables, too. As in Phase 3, the time complexity is upper bounded by \(O(n/m)\). The overall time complexity of Phase F is therefore \(O(n/m)\).

Byte complexity: The number of messages required to inform all nodes in a broken cluster of a failure is \(O(n)\): one message is charged to each node in the cluster, and cluster size (as opposed to cluster height) has a trivial upper bound equal to \(n\). The corresponding byte complexity is \(O(n \log n)\): messages carry a unique Cut ID containing two node UIDs and one Cluster ID. If a MWOE search is initiated, each of the \(O(H)\) nodes is rejected by at most \(H - 2\) siblings and accepted by one neighbor: \(O(H^2)\) messages are exchanged. The subsequent convergecast requires \(O(H)\) messages. The splitting procedure requires up to \(2m n\) messages, i.e., twice the size of a cluster. Finally, exploring connections with neighbor clusters can require up to \(|2|E|\) messages (which dominates the message complexity of Phase F) and updating routing tables requires up to \(n\) messages with a convergecast.

Messages informing nodes of a failure and exploring neighbor clusters carry a Cluster ID and a unique cut identifier. Nodes unaffected by the failure must update their routing tables by adding or removing information about three clusters at most: the original cluster may disappear and its two halves may have two existing trees. The size of all these messages is therefore \(O(\log n)\).

On the other hand, clusters containing nodes affected by the failure must update their routing tables thoroughly: connections to many clusters may have been lost in the cut and, if nodes join an existing tree, their ancestors must be notified of newly available connections. Up to \(n\) messages of size \(m \log n\) may therefore be sent.

If nodes are performing multiple consensus rounds (e.g., to track a time-varying quantity), no further messages are required: once the routing tables have been restored, the newly formed clusters just wait until the next round of consensus. If, on the other hand, consensus on a single, static value is to be performed, neighbor clusters have to update new or mutilated clusters, who may have lost messages because of the failure: the corresponding byte complexity is the same as Phase 4 of the algorithm.

Complexity of Phase OF:

Time complexity: When an inter-cluster link failure occurs, nodes on both sides of the failure update their routing table and inform their fathers, which do the same until the information reaches the root. The associated time complexity is upper bounded by the height of a tree, i.e., \(|n/m| + H = O(n/m)|\). Note that cluster flooding (Phase 4) does not stop while Phase OF is executed unless more than \(c - 1\) failures occur while routing tables are being updated.

Byte complexity: Each node along the path between the nodes next to the failure and their roots send exactly one message to its father. The overall message complexity is therefore upper bounded by \(2(\lfloor n/m \rfloor + H)\) messages. Each message carries updated information about one cluster: message size is therefore \(O(\log n)\) and the associated byte complexity is \(O(n/m \log n)\).

B. Discussion

The theoretical analysis shows that (i) the worst-case time and byte performance of our algorithm is intermediate with respect to GHS and flooding and (ii) the algorithm has the same byte complexity as GHS for \(m = 1\) and the same time complexity as flooding for \(m = n\). The algorithm therefore solves the parametrized convex consensus problem.

Time complexity: The time complexity of our algorithm is dominated by Phase 1 and Phase 4. In particular:

- Phase 1, which only needs to be executed once, sports time complexity lower than GHS by \(O(n \log m)\).
- The time complexity of Phase 4 is worse than flooding’s by a factor of \(n/m\). It is also upper bounded by \(O(n)\), since \(\text{Diam}(G_c) = O(m)\).

Byte complexity: The byte complexity of our algorithm is dominated by the cost of Phase 4, with \(O(m(n + c|E_c|) \log n)\) bytes exchanged. In particular:

- Flooding can require as many as \(O(n|E| \log n)\) bytes: our algorithm’s byte complexity is lower than flooding’s by a factor of \(n/(mc)\) at least.
- Our algorithm’s worst-case byte complexity is at least \(m/\log n\) times higher than GHS, which requires \(O((n \log n + |E|) \log n)\) bytes.
- The recurring byte cost of consensus on GHS, once a tree has been established, is \(O(n \log n)\): our algorithm, on the other hand, requires \(O(m(n + c|E_c|) \log n)\) bytes for each agreement, over a factor of \(m\) higher than GHS, even after a structure has been established.

Robustness: Our algorithm exhibits robustness properties intermediate between GHS and flooding. In particular:

- Recovery from an intra-tree failure can be achieved in \(O(n/m)\) time steps. The same failure recovery protocol requires \(\Omega(h)\) time units on the spanning tree that GHS builds, where \(h = O(n)\) is the tree height: all nodes must be informed before new edges are added to ensure that no cycles are created. Flooding does not require any reconfiguration after edge failures.
• Each of the $n-1$ edges belonging to the tree built by GHS is a single point of failure (SPF); in our hybrid algorithm, edge trees (and therefore SPFs) are $n - m$.

VII. NUMERICAL EXPERIMENTS

We tested our algorithm on I/O automata [7], spatially localized according to a random geometric graph model. For simplicity of implementation, we considered a synchronous setting (hence, the time unit is in this case represented by a “computation round”). Our tunable algorithm, GHS and flooding were executed on random geometric graphs counting 10 to 750 nodes, in increments of 10. For each number of nodes, 100 executions on randomly generated networks were considered. Our algorithm, GHS and flooding were executed on the same networks to ensure consistency of results. The hybrid algorithm was executed with four different values for $m$, decreasing from $m = n/10$ to $m = 3$ through $m = n / \log n$ and $m = \log n$.

Numerical results are shown in Figure 2 and confirm our theoretical results: our algorithm achieves time and byte complexity intermediate between that of a time-optimal algorithm (flooding) and that of a byte-optimal algorithm (GHS). Performance varies smoothly as the tuning parameter is modified. Figure 3 shows the Pareto front (time versus bytes) for different values of the tuning parameters.

![Figure 2: Time and byte complexity of our algorithm, GHS and flooding.](image)

(a) Rounds to completion. (b) Bytes exchanged.

![Figure 3: Pareto front (time versus bytes) for our algorithm (with different values for $m$), GHS and flooding.](image)

VIII. CONCLUSION

In this paper we studied the problem of designing a decentralized algorithm for the convex consensus problem that allows trade-offs among execution time, communication complexity, and robustness to link failures. Essentially, our algorithm leads the nodes in a stationary robotic network to self-organize into a semi-hierarchical structure with variable-size clusters, within which information is flooded. The size of the clusters is such that a desired balance between convergence rate and communication complexity can be achieved.

This paper leaves numerous important extensions open for further research. First, it is of interest to extend this algorithm to the setting of fast evolving networks (i.e., where edge insertions and deletions are frequent) and moving nodes. Second, our algorithm’s message complexity can indeed be higher than that of flooding. Our algorithm is therefore unsuitable whenever message complexity, as opposed to byte complexity, is a good proxy for energy cost. Accordingly, we plan to design a tunable algorithm achieving time-optimal and message-optimal behavior. Third, we plan to study the problem of time/communication trade-offs on broadcast communication network models. Fourth, it is of interest to extend our algorithm to more general collective decision-making problems. Finally, we plan to apply our algorithm to “classic” decentralized control problems (e.g., deployment of sensor nodes) to explore the potential benefits.

REFERENCES