Internet Mathematics, 12:402–418, 2016
Copyright © Taylor & Francis Group, LLC
ISSN: 1542-7951 print/1944-9488 online
DOI: 10.1080/15427951.2016.1199391

LEADER ELECTION IN SPARSE DYNAMIC NETWORKS WITH CHURN

John Augustine,[1] Tejas Kulkarni,[2] and Sumathi Sivasubramaniam[1]
[1]*Algorithms and Complexity Theory Lab, Computer Science Department, Indian Institute of Technology Madras, Chennai, India*
[2]*Department of Computer Science, University of Warwick, Coventry, UK*

Abstract *We investigate the problem of electing a leader in a sparse but well-connected synchronous dynamic network in which up to a fraction of the nodes chosen adversarially can leave/join the network per time step. At this churn rate, all nodes in the network can be replaced by new nodes in a constant number of rounds. Moreover, the adversary can shield a fraction of the nodes (which may include the leader) by repeatedly churning their neighborhood and, thus, hindering, their communication with the rest of the network. However, empirical studies in peer-to-peer networks have shown that a significant fraction of the nodes are usually stable and well connected. It is, therefore, natural to take advantage of such stability to establish a leader that can maintain good communication with the rest of the nodes. Because the dynamics could change eventually, it is also essential to reelect a new leader whenever the current leader either has left the network or is not well-connected with rest of the nodes. In such reelections, care must be taken to avoid premature and spurious leader election resulting in more than one leader being present in the network at the same time.*
We assume a broadcast-based communication model in which each node can send up to $O(\log^3 n)$ bits per round and is unaware of its receivers a priori. We present a randomized algorithm that can, in $O(\log n)$ rounds, detect and reach consensus about the health of the leader (i.e., whether it is able to maintain good communication with rest of the network). In the event that the network decides that the leader's ability to communicate is unhealthy, a new leader is reelected in a further $O(\log^2 n)$ rounds. Our running times hold with high probability provided a sufficiently large fraction of the nodes remain stable during the reelection process. Furthermore, we are guaranteed with high probability that there is at most one leader at any time.

1. INTRODUCTION

Building distributed networks that can behave in a united manner despite the presence of a large number of nodes, many of which could be faulty, is a fundamental goal in distributed computing research. At the heart of this research agenda are two important symmetry-breaking problems: (i) agreement [27, 23] and (ii) leader election [14, 1, 28]. In the (binary) agreement problem, each node in the network proposes a bit that is either 0 or 1, and they must all agree on either 0 or 1 under the condition that the agreed bit value must have been proposed by at least one node. This problem captures the essence of ensuring a

Address correspondence to Sumathi Sivasubramaniam, Algorithms and Complexity Theory Lab, Computer Science Department, Indian Institute of Technology Madras, Chennai, India. Email: sumathivel89@gmail.com

unified behavior despite multiple proposed courses of action. In many contexts, however, we would like a sustained unified behavior for which our best option is to "elect" a single leader node and authorize that leader to make the required choices. Needless to say, both leader election and agreement have a history spanning several decades and have played a crucial role in the development of distributed computing.

Although there have been several works on leader election in faulty settings [1, 17], there are very few investigations [8, 9, 3] of the leader election problem in settings where the network is dynamically evolving. Traditional leader election algorithms and their techniques don't seem to work in dynamic networks. But with dynamic networks (see [21] and references therein) gaining a great deal of importance in today's context, especially with the advent of mobile sensor networks, peer-to-peer networks, etc., it is important for us to design algorithms for fundamental problems such as agreement and leader election that can work well in dynamic networks. In an attempt to fill this gap, we study the problem of electing, maintaining,[1] and, when needed, reelecting a leader in highly dynamic networks.

Our focus, motivated by applications in peer-to-peer networks, is on sparse dynamic networks that can experience very high churn. In particular, we adopt the model first presented in [6] in which, at every time step, up to a linear fraction of the nodes in the network can be churned out and a fresh linear fraction can enter the network. Under such high churn, the entire set of nodes in the network can be replaced by fresh nodes within a constant number of rounds. Clearly, we cannot hope to elect and maintain a leader under such heavy churn. However, a detailed empirical study of real-world networks [29] indicates that although peer-to-peer networks experience heavy churn, that churn is restricted to a subset of the nodes, and a significant fraction of the nodes are, in fact, stable for reasonably long periods of time. Therefore, the focus of our work is to understand how we can elect and maintain a leader whenever the network exhibits a minimal amount of stability. In particular, if there is a small fraction of nodes that are present in the network for a sustained period of time, we show that we can, in fact, elect and maintain a leader.

1.1. Our Contribution

We model our dynamic network as a sparse evolving sequence of graphs that experience heavy churn. (We formally define our model in Section 2.1.) The motivation for our model comes from unstructured peer-to-peer (P2P) networks that nodes can join and leave at will. Typically, each peer in a P2P network connects to a bounded number of other nodes. Even though the communication graphs are sparse, we assume that they are well connected in the sense that they are expanders. Such expander graph topologies are used widely in modeling static networks [13, 18, 19, 20, 30], our work, along with other recent works [6, 5, 4], showcases the benefit of assuming good expansion in dynamic settings.

In order to ensure that our algorithm is robust, we allow an oblivious adversary to design our dynamic network. The oblivious adversary is aware of the leader election protocol, but it is unaware of the randomness that the protocol may use. In sparse networks, the adversary can easily isolate some nodes by repeatedly churning their neighborhood; the isolated nodes may not be able to communicate with other nodes. Later, at some opportune

[1]We crucially include the notion of maintenance along with leader election because in highly dynamic networks, in addition to the traditional notion of leader election, there is also a need to maintain the leader. In particular, we might need to check the availability of the leader and, if needed, initiate a new leader election.

time, it can choose to allow the nodes to freely communicate. Such adversarial strategies can potentially cause confusion in the network. It is important, therefore, to design the leader election protocol in such a way that it is robust against any strategy the adversary might employ.

Our contribution is a leader election protocol along with a framework to maintain the leader for such adversarially controlled sparse dynamic networks. Our algorithm guarantees the following properties with high probability.[2]

1. There is at most one leader in the network at any time.
2. When the leader is present and is not isolated, most nodes will be able to hear messages sent by the leader.
3. If the leader is either isolated for $\Omega(\log n)$ rounds or churned out, the network will be able to detect this anomaly and initiate reelection within $O(\log n)$ rounds.
4. Once reelection starts, it is guaranteed to terminate with the election of a new leader within $O(\log^2 n)$ rounds, provided a reasonable fraction of the nodes in the network are stable.

We note here that deterministic binary agreement is impossible even when the churn is restricted to a small constant number per round [6]. Because leader election would imply binary agreement, this impossibility extends to leader election as well. Thus, we must employ randomization to study problems under such heavy churn.

1.2. Technical Contributions

The main challenge in designing leader election algorithms for adversarially designed sparse dynamic networks with churn is the ability of the adversary to shield and release nodes at will. Consider the following illustrative scenario. Typically, when the leader is well connected with rest of the network, most nodes will hear regular beep messages sent from the leader. Suppose the leader is shielded and is unable to communicate with most nodes in the network. The beep messages will dry out, and, at some point, the other nodes will want to reelect a new leader. Crucially, we must note that the adversary has precise understanding of the point in time when the nodes will choose to reelect a new leader. So the adversary can wait untill a few rounds before that point in time and then allow the leader to communicate beep messages freely. Because the number of rounds for such free communication is limited (and adversarially timed), we will have to grapple with the worst case scenario in which about half the nodes have heard beeps from the leader (at the point in time when they have to decide whether to reelect), and the other half has not heard beeps from the leader. This could lead to a situation wherein half the nodes elect a new leader, while the other half retain the old leader. The adversarial strategy outlined might seem limited to adaptive adversaries, however, it is in fact amenable even to oblivious adversaries because it can isolate the current leader with constant probability. The rest of the strategy can be implemented once the leader is isolated.

To overcome this adversarial strategy, we introduce a novel algorithmic technique in which, instead of relying directly on beeps from the leader, we count the number of nodes

[2]Throughout this article "with high probability" (abbreviated to w.h.p) refers to probability $\geq 1 - \frac{1}{n^c}$ for any fixed $c \geq 1$, and n is the number of nodes in the network at any time, which we assume to be stable.

that heard beeps from the leader. If this counting is accurate, then each node can make a robust decision about the health of the leader based on a collective perception rather than its own perception. Thankfully, there are established robust techniques [6] to count a select subset of nodes despite heavy churn (cf. Algorithm 1).

The rest of the technical challenge is to ensure that the required properties (that our algorithm must guarantee w.h.p) are maintained despite all possible adversarial strategies. In order to maintain clarity, we have not focused on optimizing the constants in our asymptotic notations.

1.3. Related Works

There are relatively few works on leader election in dynamic networks. The works most related to our work are [8, 9, 3], in which the authors study leader election in networks that can experience heavy churn. Their model differs from ours in the sense that their model guarantees communication between every pair of nodes that remains in the network for a sustained period of time. Our notion of the stability of nodes in the dynamic network comes from the studies in [29], which show that a fraction of nodes (peers) in P2P networks remain highly stable, while the others are under constant churn.

In [8, 9] the authors study networks with arbitrary amounts of churn and provide an algorithm that guarantees deterministic reelection in $O(nD)$ rounds where D is the diameter of the network. They further show that any deterministic algorithm will require $\Omega(nD)$ rounds, thus establishing tightness. Consequently, in [3], the authors focused on randomized algorithms. Their algorithm can reelect a leader in $O(D \log n)$ rounds w.h.p. Furthermore, they also show that this running time cannot be improved.

To the best of our knowledge, we are unaware of other works that study leader election in sparse networks. However, there have been some recent efforts to study the agreement problem. In [6], as mentioned earlier, the authors show that an "almost-everywhere" agreement is impossible to achieve with deterministic algorithms in networks that exhibit even a constant churn rate.

In the context of mobile ad hoc networks, [24] presents an algorithm for electing a leader in each connected component of a network with a changing topology and proves its correctness for the case when there is a single topology change. In [16], a leader election protocol that elects a leader as soon as the topology becomes stable is described for asynchronous networks with dynamically changing topologies. Several other leader election algorithms for mobile environments are considered in [7, 24, 16, 25, 26, 31, 10].

The authors of [15] present a leader election algorithm in a model where the entire space accessible by mobile nodes is divided into nonintersecting subspaces. When nodes meet in a common subspace, they decide which node continues to participate in the leader election protocol. Moreover, [15] presents a probabilistic analysis for the case in which the movement of nodes are modeled as random walks.

Finally, we note that the notion of self-stabilization initiated by [11] is quite relevant in our context. The goal in self-stabilization is to ensure that the state of the network eventually becomes correct (under some specified notion of correctness) regardless of its starting state. In fact, one can view the work by Malpani et al. [24] as a self-stabilizing leader election protocol. To the best of our knowledge, the goal in self-stabilization is to bring the state of the network to a correct state. Our work differs in the sense that the underlying network is constantly changing, and our goal is to ensure that a leader is elected and maintained ad infinitum. For more details on self-stabilization, we refer the reader to an excellent book [12].

2. PRELIMINARIES AND PROBLEM STATEMENT

2.1. Modeling Dynamic Networks

Given a synchronous dynamic network $\mathcal{G}$ (as introduced in [6]), we want to elect a leader in spite of churn. Time is divided into discrete intervals of equal length called rounds, which are numbered integrally in sequence starting from 1. We model $\mathcal{G}$ as a sequence of connected undirected graphs $(G^1, G^2, G^3 \dots)$, where each $G^r = (V^r, E^r)$ is the communication graph in round r. The set V^r is the set of all nodes present in round r, and an edge $e = (u, v) \in E^r$ represents the ability of nodes u and v to communicate with each other. Each node has an ID. We assume that no two nodes present in the same round have the same ID. This sequence of graphs is generated by an oblivious adversary that is aware of the details of the protocol, but is unaware of the actual values of the random bits. The rules that the adversary must observe will be spelled out shortly. In a round, a node communicates with each neighbor by broadcasting at most one message of length $O(\log^3 n)$ bits.

Throughout this article, we assume that the nodes have a common clock and are therefore aware of the current round number. To keep the algorithm and analysis simple, we assume that all nodes are aware of the size of the network n.

To facilitate the leader election context, each node u is equipped with a special variable called LEADER_u, which points to $\perp$ if u doesn't have a leader or stores the ID of u's current leader. A typical round r consists of the following steps.

1. The oblivious adversary updates the network communication graph to G^r.
2. Nodes perform local computation.
3. Nodes broadcast messages to their neighbors in G^r.
4. Finally, nodes receive messages sent by their neighbors.

We make the following assumptions about the kind of changes that our dynamic network can undergo. Note in particular that the oblivious adversary is required to ensure that these assumptions are followed.

1. **Stable network size**: $\forall r$, $|V^r| = n$. At every round r, the adversary must ensure that the number of nodes in the network is maintained at n. Furthermore, we assume that the value of n is common knowledge among the nodes in the network. (Although this assumption makes our exposition cleaner, this is not a rigid requirement. Our algorithm can be adapted to work as long as the number of nodes is within a bounded range $[(1-\delta)n, (1+\delta)n]$, for a suitably small $\delta > 0$.)
2. **Churn**: We say that a node u is churned out in round r if, $u \in V^{r-1}$ but $u \notin V^r$. For every $r \geq 1$, $|V^r \setminus V^{r+1}| = |V^{r+1} \setminus V^r| \leq \mathcal{L} = \epsilon n$, where $\mathcal{L}$ is the churn limit and $\epsilon > 0$ is a fixed constant.
3. **Bounded degree expanders**: Each instantaneous graph G^r at round r in a sequence of graphs is a bounded degree expander of vertex expansion α, i.e., $\forall S \subset V^r$ s.t. $|S| \leq \frac{n}{2}$, $\frac{|\Gamma_r(S)|}{|S|} \geq 1 + \alpha$, where $\Gamma_r(S)$ is the set of closed neighbors of S in V^r (i.e.,

$\Gamma_r(S) = \{u | u \in S\} \cup \{u \in V^r | \exists v \in S$ such that $(u, v) \in E^r\})$ and $\alpha \in (0, 1]$ is a fixed constant.

Our goal is to formulate a leader election problem such that whenever the network experiences some stability, the network is able to elect a leader. In order to formalize the notion of stability we consider, we begin with some basic characteristics of the type of dynamic networks we consider.

2.2. Characteristics of Dynamic Networks and Known Algorithms

When a node u wants to flood a message m, it creates a pair $(m, \langle\text{CONDITION}\rangle)$ and repeatedly broadcasts the pair to its neighbors while the $\langle\text{CONDITION}\rangle$ is satisfied. Each recipient in turn repeatedly broadcasts the pair again as long as the $\langle\text{CONDITION}\rangle$ is satisfied.[3] For example, if a node wants to send a beep message starting from round r for a period of $O(\log n)$ rounds to all its other nodes, it can flood $(\mathsf{beep}, \langle\text{WHILE ROUND NUMBER} \leq r + O(\log n)\rangle)$. We now need to understand the extent to which the message is flooded, for which we need the notion of the influence set of u, which is closely related to Lamport's causality [22]. Formally, the *influence set of a node u starting from round r to $r' \geq r$* (denoted $\mathsf{Influence}_r(u, r')$) is defined recursively as follows:

$$\mathsf{Influence}_r(u, r) = \Gamma_r(\{u\})$$
$$\mathsf{Influence}_r(u, r') = \Gamma_{r'}(\mathsf{Influence}_r(u, r' - 1) \cap V^{r'}), \qquad \text{when } r' > r. \tag{1}$$

Informally, the influence set of u starting from round r to r' is the set of vertices in $V^{r'}$ that received (at some round in $[r, r']$) a message flooded from u starting at round r. Note that u is not required to be in $\mathsf{Influence}_r(u, r')$ for $r' > r$ because the $\mathsf{Influence}_r(u, r')$ may be nonempty even after u is churned out. The influence set can also be defined for a set U of nodes as follows:

$$\mathsf{Influence}_r(U, r') = \bigcup_{u \in U} \mathsf{Influence}_r(u, r'). \tag{2}$$

The *dynamic distance between u and v starting at round r* (denoted $DD_r(u \to v)$) is the smallest $\delta \geq 0$, such that $v \in \mathsf{Influence}_r(u, r + \delta)$. We define the *churn expansion ratio* as $\beta = \epsilon(1 + \alpha)/\alpha$. As in [6], we assume that β is sufficiently small; in our case an upper bound of $1/20$ is sufficient. The following lemmas guarantee that the influence set of a sufficiently large set U grows very quickly, and furthermore, such large sets cannot be contained.

Lemma 2.1. (cf. [6]). *Consider any set $U \subseteq V^{r-1}$ such that $|U| \geq \beta n$. There is a constant δ independent of n, but dependent on ϵ and α such that* $|\mathsf{Influence}_r(U, r + \delta)| \geq n - \beta n$.

[3]In explaining algorithms via pseudocode in this article, we will only state the initial flooding with the appropriate conditioning that controls the extent to which the message is flooded. The repeated flooding (based on the condition) will not be explicitly stated, but rather implicitly assumed.

Lemma 2.2. (cf. [6]) *Let U be any subset of V^r, $r \geq 1$, such that $|U| \geq \beta n$. There is at least one $u^* \in U$ such that for some $\Delta' \in O(\log n)$, $|\mathsf{Influence}_r(u^*, r + \Delta')| > n - \beta n$.*

Corollary 2.3. *Let S be the set of nodes in V^r such that for any $s \in S$, $|\mathsf{Influence}_r(s, r + \Delta')| < \beta n$. The cardinality of S is less than βn.*

In this article, we design algorithms that run in cycles of $\Delta \in O(\Delta') = O(\log n)$ rounds. This cycle period Δ is chosen to be sufficiently large. For concreteness, we set $\Delta = 2\Delta'$.

2.3. Support Estimation

Using the notions described so far, we can describe a distributed randomized algorithm to estimate the size of a set of nodes that satisfy some condition [6] (see Algorithm 1 for the formal pseudocode). We will repeatedly use this algorithm in our work. For simplicity in presenting the support estimation algorithm, we assume that the algorithm is initiated in round 1 on $G^1 = (V^1, E^1)$. Suppose $\mathcal{R}$ nodes in V^1 are colored red. We call $\mathcal{R}$ the support of the red nodes. We need most nodes in the network to be able to estimate $\mathcal{R}$ to within a small error.

Algorithm 1 Algorithm to estimate the support $\mathcal{R}$ of red nodes when $\mathcal{R} \geq n/2$

```
The following pseudocode is executed at every node $u$.
$P \in \Theta(\log n)$ controls the precision of our estimate. Its exact value is worked out in the proof of Theorem 2.4.

At round 1:
1: Draw $P$ random numbers $s_1, s_2, \ldots, s_i, \ldots, s_P$, each from the exponential random distribution with rate 1.
   {Each $s_i$ is chosen with a precision that ensures that the smallest possible positive value is at most $\frac{1}{n^{\Theta(1)}}$; Note that $\Theta(\log n)$ bits suffice.}
2: For each $s_i$, create a message $m_u(i)$ containing $s_i$ and a terminating condition: HAS ENCOUNTERED A MESSAGE $m_v(i)$ WITH A SMALLER RANDOM NUMBER.
   {Notice that a node $u$ will flood exactly one message at each index $i$—in particular the smallest random number encountered by node $u$ with message index $i$}
3: For each $i$, initiate flooding of message $m_u(i)$.

For the next $t = \Theta(\log n)$ rounds:
4: Continue flooding messages, respecting their termination conditions.
   {It is easy to see that the number of bits transmitted per round through a link is at most $O(\log^2 n)$.}

At the end of the $\Theta(\log n)$ rounds:
5: For each $i$, the node $u$ holds a message $m_v(i)$. Let $\bar{s}_u(i)$ be the random number contained in $m_v(i)$.
6: $\bar{s}_u \leftarrow \frac{\sum_i \bar{s}_u(i)}{P}$.
7: Node $u$ outputs $1/\bar{s}_u$ as its estimate of $\mathcal{R}$. {Now that the estimation is completed, all messages can be terminated.}
```

Theorem 2.4. (from [6]). *Consider an oblivious adversary and let γ be a an arbitrary fixed constant ≥ 1. Let $\bar{\mathcal{R}} = \max(\mathcal{R}, n - \mathcal{R})$. By executing Algorithm 1 to estimate both $\mathcal{R}$ and $n - \mathcal{R}$, we can estimate $\bar{\mathcal{R}}$ to within $[(1-\delta)\bar{\mathcal{R}}, (1+\delta)\bar{\mathcal{R}}]$ for any $\delta > 2\beta$ with probability at least $1 - n^{-\gamma}$.*

2.4. Shielding in Dynamic Networks

We are now ready to introduce the notion of *shielding*. Informally, the term shielding captures an adversary's ability to hinder communication between a node u and most other nodes in the network. There are two ways that communication can be hindered. On the one hand, the adversary can "mute" u by limiting its influence set to be very small for a sufficiently long period of time. On the other hand, the adversary can hinder communication by disallowing most nodes to send messages to u (thus isolating them), again, for a sufficiently long period of time.

We formally define the notion of shielding using the notion of influence sets. We say that *a node u is shielded from round r to $r + P$*, for some $P \geq \Delta$ (cf. Lemma 2.2) if at least one of the following conditions holds.

Muted. $\mathsf{Influence}_t(u, t+\Delta) < \beta n$ for every $t \in [r, r+P-\Delta]$. (Note that Lemma 2.1 ensures that once the influence set exceeds βn, it will soon reach $n - \beta n$. Furthermore, it is not difficult to see that it will never decrease below $n - \beta n$.)

Isolated. For every $t \in [r, r+P-\Delta]$, there is a set $X \subseteq V^t$ of size at least $n - \beta n$ nodes such that $u \notin \mathsf{Influence}_t(X, t+\Delta)$. From Lemma 2.2, $|\mathsf{Influence}_t(X, t+\Delta)| \geq n - \beta n$, and therefore, u is in a small (less than βn sized) set of nodes that are not in $\mathsf{Influence}_t(X, t+\Delta)$.

A node that is shielded for a sufficiently long $P \in \Omega(\log n)$ rounds is simply said to be *shielded*. A node that is not shielded is called an unshielded node. A leader that is present in the network and unshielded is said to be a healthy leader. Otherwise, it is said to be unhealthy. (We sometimes use the term *health of the leader* to refer to its status of health.)

2.5. Problem Definition

Our goal is to design a leader election protocol that works despite network dynamism. Ideally, we would like to ensure that there is a unique leader in the network at all times. Unfortunately, the adversary, though oblivious, can ensure that the leader is churned out in $1/\epsilon$ rounds. Alternatively, with constant probability the adversary can shield the leader for arbitrary lengths of time. Intuitively, this shielding can happen due to the repeated churning of a cut set. Thus, we require a leader election protocol that is able to self-diagnose the health of the leader and, if necessary, reelect a new leader. In this sense, we require a self-healing leader election protocol that runs ad infinitum.

Formally, any leader election algorithm developed should satisfy the following conditions w.h.p. for some suitable constants c_1 and $c_2 > c_1$.

Agreement. At any round r, $\textsc{leader}_u \neq \perp \wedge \textsc{leader}_v \neq \perp \rightarrow \textsc{leader}_u = \textsc{leader}_v$ $\forall u, v \in V^r$. Although this condition might sound trivial, one must note that a careless leader election protocol might, for example, end up with two leaders: (i) a shielded old leader and (ii) a newly elected leader. The agreement condition forbids this situation.

Stability. This condition is designed to prevent frivolous leader reelections. Intuitively, if a node u changes its leader variable, then either u itself is shielded (and therefore does not have reliable information about the health of the leader) or the leader has become unhealthy. More formally, if at any round r, $\textsc{leader}_u = v$ and at round $r+1$, $\textsc{leader}_u \neq v$ then either

- node u or node v was shielded for a period of at least $c_1 \log n$ rounds contained within rounds $r - O(\log n)$ and r, or
- node v was churned out from the network between rounds $r - O(\log n)$ and r.

Timely reelection. The intention behind this condition is to ensure that an unhealthy leader does not remain a leader for too long. More precisely, if a leader is shielded for a period of $c_2 \log n$ rounds, then, (i) the current leader must reset its $\textsc{leader}$ variable to $\perp$ and (ii) a new leader must be elected. (We allow the possibility that the old leader is again reelected as the new leader, provided it has reached a healthy state again.)

Bounded reelection time T. Let r be a time such that either (i) the leader is churned out or (ii) the leader is shielded from round r for at least $c_2 \log n$ rounds. We say that our algorithm has bounded reelection time T if, under the condition that a reasonable number (say, $n/2$ for convenience) of nodes in the network are stable for a period of time T, i.e., $|\bigcap_{r' \in [r, r+T]} V^{r'}| \geq n/2$, a new leader must have been elected sometime between $[r+1, r+T]$.

Leader Awareness. Suppose a node v sets $\textsc{leader}_v = v$ in round r and remains the leader until round r'. Then, for every $r'' \in [r, r']$, nodes in $\mathsf{Influence}_r(v, r'')$ must have set their leader variable to v by the end of round r''. This condition can be easily maintained (as we do) via beep messages sent from the leader—a notion closely related to the notion of "power supply" introduced in [2].

Validity. If some node u elects another node v as its leader in round r, then v must have been the leader in some round in $[r - O(\log n), r]$.

3. RANDOMIZED ALGORITHM FOR DYNAMIC LEADER ELECTION

Now, we present a randomized algorithm for solving dynamic leader election on the model described in the previous section. We also prove that our algorithm is correct and terminates w.h.p. in $O(\log^2 n)$ rounds. The goal of any algorithm solving this problem is to elect and sustain the leader as long as it can and have most nodes in the network be aware of the leader's presence. Although this might be quite straightforward in many settings, our adversarially designed sparse network model with high levels of churn poses some unique challenges. The main difficulty stems from the adversary's ability to shield a set of up to βn nodes. Furthermore, the adversary can choose to release (i.e., unshield) the shielded nodes at any point in time that suits its purpose. For example, the adversary may lay down the network topology in such a way that the leader might be physically present in the network but might get muted in such a way that most nodes in the network (at least $n - \beta n$ nodes to be precise) are unable to hear any messages/beeps from the leader, making it ineffective or partially effective. One way an adversary can realize such topology is by churning out leader's current neighbors in every round. Suppose the nodes try to detect such muting of the leader and attempt to reelect a new leader. Just when the nodes are about to start the reelection process, the adversary can release the leader and, due to the well-connected nature of the network, about half the nodes might have heard the beep messages. Should those nodes that heard the beep message proceed with the reelection? Answering this question in a unified manner is crucial to ensure that the network has a leader that is elected in agreement with most nodes in the network.

A crucial centerpiece of our algorithm is a regular beep message sent out by the leader. We assume throughout this study that the following code is executed at every time step. The regular beeps sent out at every time step ensures that most nodes in the network

Algorithm 2 Beeps from the leader

```
1: The leader (if present) initiates the flooding of a beep message with the current time stamp.
2: Every other node floods the most recently initiated beep message provided it was initiated in the
   last $\Delta$ rounds.
3: Furthermore, whenever a node $u$ hears a beep from a node $v$ and $\text{LEADER}_u \neq v$, it sets $\text{LEADER}_u \leftarrow$
   $v$.
```

(i.e., at least $n - \beta n$ nodes) are aware of the leader's presence when the leader is unshielded. The following lemma follows quite easily.

Lemma 3.1. *The ad infinitum execution of Algorithm 2 in every round (deterministically) ensures (i) leader awareness and (ii) validity as required in the problem definition.*

When a node u does not hear beep messages for a period of Δ rounds, there are two (not necessarily exclusive) possibilities. Either, (i) the leader may have been muted (or churned out, which has the same effect as being muted), or, (ii) the node u might have been isolated. The node u must then, in concert with rest of the nodes, decide if a reelection is required; this is addressed in the next section.

3.1. The Framework

We now present the leader election framework (Cf. Algorithm 3) that is expected to run in the background and keep a check on the health of the leader. The framework is responsible for invoking the leader election protocol when the current leader is either churned out or shielded. Note that in this distributed context, it is important for the entire network to reach an agreement on reelecting a new leader. A naïve use of the existing agreement algorithm [6] will require $\Theta(\log^2 n)$ rounds. The important downside to using the existing agreement algorithm is that the leader must be shielded for $\Omega(\log^2 n)$ rounds before the network can start taking countermeasures. To circumvent this downside, we present a protocol that checks the health of the leader and, when the leader is shielded (or churned out) for a period of $\Theta(\log n)$ rounds, ensures that most nodes in the network agree to perform a reelection.

The leader election framework runs in cycles of length Δ rounds. Each cycle starts at a round number that is a multiple of Δ. Thus, the start of each cycle is common knowledge among all nodes in the network. At the start of each cycle, the nodes currently in the network initiate the support estimation (discussed in Section 2.2) of (i) the number of nodes (in the current round) that received a beep message in the previous cycle, (ii) the number of nodes (in the current round) that have set their `elect` bit to `true`, and (iii) the total number of nodes in the network in the current round. These support estimations will take $\Theta(\log n)$

rounds to complete and, in particular, will be completed by the end of the current cycle. At that point, the leader (if present) will check for the following conditions.

1. The number of nodes that received beep messages must not be too low. If it is too low, say, less than $3\beta n$, then the leader resets its own LEADER variable because the leader has been muted for a period of at least $\Delta' = \Theta(\log n)$ rounds in the previous cycle. Suppose the leader is not muted; then, the leader's beep messages should have been received by at least βn nodes in Δ' rounds, and then, by Lemma 2.1, the beep messages are guaranteed to reach all but at most βn nodes in a further $O(1)$ rounds.
2. Furthermore, because we are guaranteed that the total number of nodes in the network is n, any accurate estimate of n must be close to n. Thus, if the leader's estimate is below some threshold, say, below $n - 3\beta n$, then the leader, w.h.p., must have been isolated. Therefore, it again resets its own leader variable.

The rest of the nodes must follow suit and conclude that the leader is, in fact, either shielded or churned out and then agree on a reelection. A straightforward and seemingly correct approach would be for each node u to check if it hears recent beep messages (initiated within the last Δ rounds) and decide to start the leader election protocol if no beep messages were heard. Unfortunately, this will not work; u might not have heard the beep messages because the adversary might have isolated u. Counting the number of nodes that received beep messages will provide the node with some extra information. If the number of nodes that received beep messages (denoted by BEEP$_u$ in Algorithm 3) is low, then, clearly, the leader is muted. However, a single threshold will not work because each node could estimate a different number and the adversary can ensure that some half of the nodes will decide to reelect while the other half concludes otherwise. We, therefore, need a mechanism to reinforce the decision to reelect. Each node u has an elect bit that indicates whether u believes that a reelection is imminent. If the support estimation for the number of nodes that have set their elect bit to true exceeds a large number (say $n - 3\beta n$), then we can conclude w.h.p. that the leader has reset and the nodes actually proceed with reelection.

Lemma 3.2. *Algorithm 3 (w.h.p.) guarantees the timely reelection condition required in the leader election problem definition.*

Proof. Suppose the current leader is muted for an entire cycle. Then, at the end of the cycle, w.h.p., at least $n - \beta n$ nodes would not have heard any beep in that cycle. Therefore, in the next cycle, at least $n - \beta n$ nodes will estimate BEEP$_u$ to fewer than $3\beta n$ w.h.p. (cf. line number 10 in Algorithm 3) and, therefore, will set their ELECT bits to true. In the next cycle, a support estimation of the number of nodes that set their ELECT bits to true (cf. line number 12 in Algorithm 3) will w.h.p. lead to (i) the leader resetting its LEADER variable and (ii) most ($\geq n - \beta n$) nodes entering the reelection protocol. □

Lemma 3.3. *Algorithm 3 guarantees the stability condition required in the leader election problem definition w.h.p.*

The proof of Lemma 3.3. is deferred to the full version of the article due to space limitations.

Algorithm 3 Framework for the detection of Leader's health

```
1: {We describe the steps executed by the nodes in each cycle (of duration $\Delta$ rounds). We assume
   that a new cycle starts at every round with a round number that is a multiple of $\Delta$.}
2: {The steps are described from the perspective of a single arbitrary node $u$ in the network.}
3: {These steps are assumed to be executed throughout the life of the network, even when the leader
   election protocol (in Algorithm 4) is executed.}
4: {Each node has a Boolean variable elect normally set to false. Intuitively speaking, a node
   sets its elect bit to true when it gathers evidence that a reelection is necessary. When the
   evidence becomes overwhelming, it sets it to true and initiates the leader election protocol (in
   Algorithm 4).}
At the start of the cycle:
5: Each node $u$ initiates support estimation for the following quantities:
   1. the number of nodes (in the current round) that received a beep message in the previous cycle,
   2. the number of nodes (in the current round) that have set their elect bit to true.
   3. the total number of nodes in the network in the current round.
At the end of the cycle:
6: {The three support estimations should be complete. Let $\textsc{beep}_u$ be the number of nodes that
   received beeps in the previous cycle (as estimated by $u$). Let elects be the number of nodes that
   had set their elect bit to true in the previous cycle (again, as estimated by $u$). Finally, let $\mathsf{size}_u$
   be the estimate of the number of nodes in the network according to $u$.}
7: if $\textsc{leader}_u = u \wedge (\textsc{beep}_u \leq 7\beta n \vee \mathsf{size}_u \leq n - 3\beta n)$ then
8:     {$u$ is the leader and it is shielded.}
9:     $\textsc{leader}_u \leftarrow \perp$
10: else if $(\textsc{beep}_u \leq 3\beta n)$ then
11:     The node $u$ sets its own elect bit to true.
12: else if elects $\geq n - 3\beta n$ then
13:     The node $u$ sets its own elect bit to true.
14:     $\textsc{leader}_u \leftarrow \perp$
15:     Start the reelection algorithm in the next round with a round number that is a multiple of $4\Delta$.
16: else
17:     The node $u$ sets its own elect bit to false.
```

3.2. The Leader Election Algorithm

We are now ready to describe the leader election protocol that is invoked whenever the framework described in Algorithm 3 reaches an agreement to reelect a new leader. The protocol is formally described in pseudocode format in Algorithm 4. This protocol works in phases of four cycles (each requiring Δ rounds) called quarters, thus, each phase takes a total of 4Δ rounds. Since the round numbers are assumed to be common knowledge, we start a phase in a round number that is a multiple of 4Δ (cf. line number 15 of Algorithm 3). The activities in each quarter (at a high level first) are presented along with an ideal flow of execution. Along the way, we describe adversarial strategies designed to derail the execution and the manner in which the algorithm ensures robustness against such adversarial strategies.

Quarter 1 In this quarter, the nodes generate random numbers uniformly and independently from (0, 1). The random numbers (along with ID of nodes that generated them) are flooded with priority to smaller random numbers. The intention is for the node that generated the smallest random number to become the leader. Note however, that the adversary can mute close to βn nodes and release them at will. This could potentially violate the agreement

Algorithm 4 Leader election algorithm

```
1: {The Algorithm operates in phases. Each phase consists of four cycles (of $\Delta$ rounds each) called
   quarters.}
Quarter 1. (Generating and flooding random numbers)
2: At the start of the quarter, each node $u$ generates a random number $r_u \in (0, 1)$ (with
   sufficient number of bits to ensure uniqueness w.h.p.) and initiates a flooded message
   $M(u) = ((u, r_u), \langle r_u$ IS THE SMALLEST RANDOM NUMBER SEEN SO FAR IN AN $M(\cdot)$ MESSAGE$\rangle \vee$
   $\langle$WE HAVE REACHED THE END OF THE SECOND QUARTER.$\rangle)$.
3: {The intention is to let most nodes know the ID of the node that generated the smallest $r_u$.}
4: At the end of the quarter, each node $u$ sets $\mathsf{Winner}_u$ variable to the message with the smallest
   random number that entered $u$. {Note that the flooding continues in the second quarter.}
Quarter 2. (Picking $O(\log n)$ random numbers for support estimation.)
5: {All steps in this quarter are initiated at the start of the quarter.}
6: At the start of the quarter, each node $u$ chooses itself with probability $\frac{\log n}{n}$. {Note that by a simple
   application of Chernoff's bound, we can ensure that at most $O(\log n)$ nodes will be selected
   w.h.p.}
7: if $u$ chose itself then
8:     Node $u$ initiates a flooded message $C(u) = (\mathsf{Winner}_u$,
       $\langle$WHILE IT IS STILL THE SECOND QUARTER$\rangle)$.
Quarter 3. (Estimating support of various parameters.)
9: {All steps in this quarter are initiated at the start of the quarter.}
10: Initiate the support of the number of nodes that received the LONG_BEEP message. {Cf. line
    number 26 for source of these long beeps from the previous phase.}
11: Initiate estimation of the number of nodes in the network.
12: Let $M^*$ be the $M(\cdot)$ message with the smallest random number that flooded over $u$ until the end
    of the second quarter.
13: Let $\mathsf{min}$ be the ID in $M^*$.
14: if One of the $C(\cdot)$ messages received by $u$ contains a $\mathsf{Winner}$ whose ID equals $\mathsf{min}$ then
15:     Initiate support estimation procedure for the number of nodes that received $M^*$ as their $M(\cdot)$
        message with the smallest random number.
Quarter 4. (Leader Election)
16: {The following steps are executed at the start of the quarter.}
17: {w.h.p., only up to $O(\log n)$ different supports for various $M(\cdot)$ messages will be estimated.
    Among them, let $\mathsf{max}$ be the ID of the $M(\cdot)$ message with the highest support (as estimated by
    $u$). }
18: if the estimated number of nodes in the network is fewer than $n - 3\beta n$, then
19:     LEADER$_u \leftarrow \perp$. {Node $u$ is isolated. Cf. stability condition in the problem definition.}
20: else if the support of the LONG_BEEP message is at least $n - 3\beta n$, then
21:     Do nothing. {There seems to be an unmuted leader.}
22:     {At most one node $u$ generated the random number with sufficient support and that node alone
        may enter the following "else if" conditional block.}
23: else if the support of $\mathsf{max}$ is at least $n - 3\beta n$ and $\mathsf{max} = u$, then
24:     LEADER$_u \leftarrow u$.
25:     Initiate regular flooded BEEP messages in every round (as required in Algorithm 2).
26:     In addition, $u$ initiates the flooding of a special LONG_BEEP message with the condition
        $\langle$FLOOD FOR NEXT $3\Delta$ ROUNDS$\rangle$.
```

condition in our problem statement — if close to βn nodes are released close to the end of the first quarter, then with probability close to β, the smallest random number would have been heard by only about half the nodes. To counter this, we will employ support estimation (in a subsequent quarter) to count, for various random numbers that were generated, the number of nodes that think each random number is the smallest. For this purpose, each node u has a variable Winner_u, which is the smallest random number (according to u at the end of the first quarter) along with the node that generated it.

Naïvely implemented, we will require $O(n)$ difference support estimation algorithms to run simultaneously because we will execute one support estimation per random number. This is unacceptable because each support estimation execution will require a message complexity of $O(\log^2 n)$ and the overall message complexity will reach an unscalable $O(n \log^2 n)$. Therefore, we need a mechanism (implemented in the second quarter) to select a few random numbers on which we perform support estimation, while ensuring that one of the selected random numbers will be the smallest (assuming the smallest actually flooded to most of the network).

Quarter 2 To select a few random numbers on which we perform support estimation, we subsample $O(\log n)$ nodes in the start of the second quarter and these subsampled nodes flood their Winner variables. The intention is that support estimation will be performed only on those $O(\log n)$ (w.h.p) random numbers that are contained in the flooded Winner_u messages. Notice that if the smallest random number has high support (say, at least $n-3\beta n$), then, w.h.p., the smallest random number will be included among the $O(\log n)$ support estimations.

Quarter 3 Herein, we perform the following estimations.

1. Firstly, we initiate estimation of the support of (up to $O(\log n)$ w.h.p.) chosen random numbers.
2. We also estimate the total number of nodes in the network so that the leader can test whether the estimation is reasonably close to n, which is the true number of nodes in the network. If not, then the leader can, w.h.p., conclude that it is isolated and therefore reset itself; this check is done in the fourth quarter.
3. In addition, we estimate the support of the number of nodes that have heard a special form of beep message called LONG_BEEP initiated each time a new leader is elected (in the fourth quarter). This is to ensure that if a leader had been elected in the previous phase and his LONG_BEEP is heard in the current phase by many nodes, then a new leader should not be elected in the current phase.

Quarter 4 The decisions made in the quarter are based on the support estimations that were performed in the third quarter. First, if a node's estimate of the total number of nodes in the network is sufficiently less than n, then, it concludes that it is isolated. Therefore, to avoid violating the agreement condition (i.e., in case a new leader is elected and the node is unaware of it because of its isolation), it resets its leader variable. The rest of the fourth quarter is executed by at most one node u that generated the random number whose support (as estimated by u itself) is at least $n - 3\beta n$. (Clearly, w.h.p. there can only be one such node because two or more random numbers cannot be estimated (albeit by different nodes) to each have $n - 3\beta n$ support.) The node u first ensures that any leader from the previous phase (if there had been one) has been muted and would have reset itself — this check is performed by estimating the support of the LONG_BEEP messages from the previous phase. Thus, when the node u sets its LEADER$_u$ to itself, it has confirmed (w.h.p.) that there is no other leader in the network and its own random number has had the highest support, it sets its LEADER$_u$ to itself and starts beeping. Finally, node u initiates the flooding of a one-time LONG_BEEP message for a period of 3Δ rounds. To see the need for the LONG_BEEP notice that there may be a lag of some $O(\log n)$ rounds from the time a leader is elected to the time the other nodes in the network set their leader variables; sometimes this lag might be very short and other times a bit longer. Therefore, it is possible that some nodes might stop the leader election protocol while others are still trying to elect a new leader. Therefore, a newly elected leader sends out a LONG_BEEP message to avoid a reelection in the very next phase. Even if some nodes did start the reelection process, if the support of the LONG_BEEP messages is sufficiently large, no new leader will be elected.

The following two lemmas will lead to the final result stated in Theorem 3.6.

Lemma 3.4. *The framework in Algorithm 3 and the leader election protocol in Algorithm 4 together guarantee the agreement condition in the problem definition w.h.p.*

Proof. Notice that when at least one node decides to start the reelection process in Algorithm 3, w.h.p., the leader has reset its LEADER variable (cf. proof of Lemma 3.3.) Therefore, most nodes (i.e., at least $n - \beta n$ nodes) will decide to start reelection protocol in the next cycle. Because this is the only way that the network will enter the leader election protocol, we can assume that there will be no leader in the network when the leader election protocol starts. Subsequently, when a leader is elected, care is taken to ensure that the random number generated by the candidate leader has an estimated support of $n - 3\beta n$ nodes, implying that, w.h.p., no other random number can have a support sufficient for some other node to elect itself leader. Once the leader is elected, it issues a LONG_BEEP message to ensure that the next leader election phase does not lead to a new leader. Unless the leader is shielded, its beeps will be heard and the framework will take over. However, if the leader is shielded, then it will reset its leader variable (cf. line number 9 in Algorithm 3) and a new execution of the leader election protocol will be initiated. □

Lemma 3.5. *The reelection time T (defined in the bounded reelection time requirement stated in the problem definition) is upper bounded by $O(\log^2 n)$ rounds w.h.p.*

Proof. In each phase of the reelection protocol, the nodes generate random numbers and flood them with the intention of electing the node that generated the smallest random number u^* as the leader. Clearly u^* is equally likely to be any node in the first round of the phase and the oblivious adversary is unaware of its identity. Therefore, with constant probability, u^* is unshielded for a period of 4Δ rounds. It follows, therefore, that in $O(\log n)$ phases, a leader is elected w.h.p. □

Based on the list of lemmas proved, the following theorem follows.

Theorem 3.6. *Algorithm 2, Algorithm 3, and Algorithm 4 together solve the leader election problem as we have formulated and ensure that a shielded or churned out leader can be detected in $O(\log n)$ rounds and a new leader can be elected within $O(\log^2 n)$ rounds, provided that a reasonable fraction (say $n/2$) nodes are stable in the network.*

4. CONCLUSION AND FUTURE WORK

We have provided a leader election framework and protocol that elects and maintains a leader in a highly dynamic network, provided a fraction of the nodes are stable. For simplicity, we have thus far assumed that half the nodes in the network must be stable for our result to hold. One can easily see that our results will extend without any asymptotic penalties even if the size of the stable set of nodes is much smaller, say, at least $4\beta n$. Furthermore, for simplicity in exposition, we have assumed that the network has a stable size of n nodes in any round. We note that this is not a rigid requirement. Our algorithm can be easily modified to work under the milder requirement that the number of nodes in the network lie in $[(1-\delta)n, (1+\delta)n]$ for some constant $\delta \geq 0$.

We leave a number of questions open. In the future, we hope to consider models in which there is far less stability. In particular, the question we leave open is whether a leader election protocol can be designed when only one node is stable and unshielded. Finally, it will be interesting to understand the feasibility of leader election in such highly dynamic networks when Byzantine nodes are present.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of a previous version of the article for their useful suggestions and for pointing us to relevant references.

REFERENCES

[1] H. H. Abu-Amara. “Fault-Tolerant Distributed Algorithm for Election in Complete Networks.” *IEEE Trans. Comput.* 37:4 (1988), 449–453.

[2] Y. Afek and A. Bremler. “Self-Stabilizing Unidirectional Network Algorithms by Power-Supply.” In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, (SODA ’97)*, pp. 111–120. New York, NY: ACM, 1997.

[3] J. Augustine, T. Kulkarni, P. Nakhe, and P. Robinson. “Robust Leader Election in a Fast-Changing World.” In *Proceedings Ninth International Workshop on Foundations of Mobile Computing, (FOMC’13)*, Israel, 2013.

[4] J. Augustine, A. R. Molla, E. Morsy, G. Pandurangan, P. Robinson, and E. Upfal. “Storage and Search in Dynamic Peer-to-Peer Networks.” In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA’13)*, pp. 53–62. New York, NY: ACM, 2013.

[5] J. Augustine, G. Pandurangan, and P. Robinson. “Fast Byzantine Agreement in Dynamic Networks.” In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing (PODC’13)*, pp. 74–83. New York, NY: ACM, 2013.

[6] J. Augustine, G. Pandurangan, P. Robinson, and E. Upfal. “Towards Robust and Efficient Distributed Computation in Dynamic Peer-to-Peer Networks.” In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA’12)*. New York, NY: ACM, 2012.

[7] A. Boukerche and K. Abrougui. “An Efficient Leader Election Protocol for Mobile Networks.” In *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing (IWCMC ’06)*, pp. 1129–1134. Vancouver, British Columbia, Canada: ACM, 2006.

[8] H. C. Chung, P. Robinson, and J. L. Welch. “Regional Consecutive Leader Election in Mobile Ad-hoc Networks.” In *Proceedings of the 6th International Workshop on Foundations of Mobile Computing*, (DIALM-POMC ’10), pp. 81–90. New York, NY, USA: ACM, 2010.

[9] H. C. Chung, P. Robinson, and J. L. Welch. “Optimal Regional Consecutive Leader Election in Mobile Ad-hoc Networks.” In *Proceedings of the 7th ACM ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing* (FOMC ’11), pp. 52–61. New York, NY: ACM, 2011.

[10] A. K. Datta, L. L. Larmore, and H. Piniganti. “Self-Stabilizing Leader Election in Dynamic Networks.” In *Proceedings of the 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems of (SSS’10)*, pp. 35–49. New York, NY: Springer, 2010.

[11] E. W. Dijkstra. “Self-Stabilizing Systems in Spite of Distributed Control.” *Commun. ACM* 17:11 (1974), 643–644.

[12] S. Dolev. *Self-Stabilization*. Cambridge, MA: MIT Press, 2000.

[13] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. "Fault Tolerance in Networks of Bounded Degree." *SIAM J. Comput.* 17:5 (1988), 975–988.

[14] H. Garcia-Molina. "Elections in a Distributed Computing System." *IEEE Trans. Comput.* 31:1 (1982), 48–59.

[15] K. Hatzis, G. Pentaris, P. Spirakis, V. Tampakis, and R. Tan. "Fundamental control algorithms in mobile networks." In *Proceedings of the 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'09)*, pp. 251–260. New York, NY: ACM, 1999.

[16] R. Ingram, P. Shields, J. Walter, and J. L. Welch. "An Asynchronous Leader Election Algorithm for Dynamic Networks." In *Proceedings of 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, pp. 1–12. IEEE, 2009.

[17] A. Itai, S. Kutten, Y. Wolfstahl, and S. Zaks. "Optimal Distributed t-Resilient Election in Complete Networks." *Software Engineering, IEEE Transactions on* 16:4 (1990), 415–420.

[18] B. M. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. "Fast Asynchronous Byzantine Agreement and Leader Election with Full Information." *ACM Transactions on Algorithms* 6:4 (2010).

[19] V. King and J. Saia. "Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary." *J. ACM* 58:18 (2011), 1–24.

[20] V. King, J. Saia, V. Sanwalani, and E. Vee. "Towards Secure and Scalable Computation in Peer-to-Peer Networks." In *Procceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 87–98. IEEE. 2006.

[21] F. Kuhn and R. Oshman. "Dynamic Networks: Models and Algorithms." *SIGACT News* 42:1 (2011), 82–96.

[22] F. Kuhn, N. Lynch, and R. Oshman. "Distributed Computation in Dynamic Networks." In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing (STOC'10)*, pp. 513–522. New York, NY: ACM, 2010.

[23] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem." *ACM Trans. Program. Lang. Syst.* 4:3 (1982), 382–401.

[24] N. Malpani, J. L. Welch, and N. Vaidya. "Leader Election Algorithms for Mobile Ad Hoc Networks." In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'00)*, pp. 96–104. Boston, MA: ACM, 2000.

[25] S. Masum, A. Ali, and M. Bhuiyan. "Fundamental Control Algorithms in Mobile Networks." In *Procceedings of the 20th International Conference on Advanced Information Networking and Applications AINA'06*, pp. 29–34. Vienna, Austria: IEEE Computer Society, 2006.

[26] P. Parvathipuram, V. Kumar, and G.-C. Yang. "An Efficient Leader Election Algorithm for Mobile Ad Hoc Networks. In *Proceedings of the 1st International Conference on Distributed Computing and Internet Technology, (ICDCIT'04)*, pp. 32–41. Bhubaneswar, India: Springer, 2004.

[27] M. Pease, R. Shostak, and L. Lamport. "Reaching Agreement in the Presence of Faults." *J. ACM* 27:2 (1980), 228–234.

[28] D. Peleg. "Time-Optimal Leader Election in General Networks." *J. Parallel Distrib. Comput.* 8:1 (1990), 96–99.

[29] D. Stutzbach and R. Rejaie. "Understanding Churn in Peer-to-Peer networks." In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pp. 189–202. New York, NY: ACM, 2006.

[30] E. Upfal. "Tolerating a Linear Number of Faults in Networks of Bounded Degree. *Inf. Comput.* 115:2 (1994), 312–320.

[31] S. Vasudevan, J. Kurose, and D. Towsley. "Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks." In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)*, pp. 350–360. IEEE, 2004.