# Self-Stabilizing Leader Election for Single-Hop Wireless Networks despite Jamming

Andrea Richa[1], Christian Scheideler[2], Stefan Schmid[3], Jin Zhang[1]

[1] Computer Science and Engineering, SCIDSE, Arizona State University, Tempe, AZ 85287, USA; {aricha,jzhang82}@asu.edu
[2] Department of Computer Science, University of Paderborn, D-33102 Paderborn, Germany; scheideler@upb.de
[3] Deutsche Telekom Laboratories & TU Berlin, D-10587 Berlin, Germany; stefan@net.t-labs.tu-berlin.de

## ABSTRACT

Electing a leader is a fundamental task in distributed computations. Many coordination problems, such as the access to a shared resource, and the resulting inefficiencies, can be avoided by relying on a leader. This paper presents SELECT, a leader election protocol for wireless networks where nodes communicate over a shared medium. SELECT is very robust in two respects. First, the protocol is self-stabilizing in the sense that it converges to a correct solution from any possible initial network state (e.g., where no or multiple nodes consider themselves a leader). This is an appealing property, especially for dynamic networks. Second, the described protocol is resilient against a powerful reactive jammer that blocks a significant fraction of all communication rounds. The reactive model is general and of interest beyond jamming (e.g., in the context of co-existing networks). The paper also reports on experimental results obtained from our simulation framework which allows us to study convergence behavior under different types of adversarial jammers.

## Categories and Subject Descriptors

C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Access schemes*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

## General Terms

Algorithms, Reliability, Theory

## Keywords

Wireless Ad-hoc Networks, MAC Protocols, Jamming

## 1. INTRODUCTION

Leader election is a classical theme in the field of distributed algorithms. Once a leader is determined, many coordination tasks are simplified. In this paper, we consider the problem of electing a leader in a wireless network in order to coordinate access to a shared communication medium. This can greatly improve the overall throughput of the network and reduce the energy consumption at the nodes.

We focus on a harsh environment where the wireless nodes contend for a single wireless channel that is jammed by a powerful reactive adversary blocking an arbitrary constant fraction of all time slots. In addition, we require that the election protocol works correctly if started from *any* initial network state—i.e., it is *self-stabilizing*. This implies that arbitrary join and leave behavior can be tolerated. For example, when a leader node leaves the network, a substitute leader is elected.

Disruptions of the shared communication medium—either due to interference of concurrent transmissions or adversarial jamming—is one of the foremost challenges in wireless computing. *Jamming attacks* are a very cumbersome problem as they are typically easy to implement and the attacker does not need any special hardware. For example, it has been pointed out that the widely used IEEE 802.11 medium access control (MAC) protocol already fails to handle simple, oblivious jammers [6].

**Model.** We consider the problem of designing a self-stabilizing distributed protocol to elect a leader among a set $V$ of $n$ simple wireless nodes (e.g., nodes of a sensor network) that are within each other's transmission range and communicate over a single channel. For our formal analysis, we assume that the time proceeds in synchronous *rounds* (or *steps*).[1] In each round, a node may either transmit a message or sense the channel, but it cannot do both, and there is no immediate feedback mechanism telling a node whether its transmission was successful.[2] A node which is sensing the channel may either $(i)$ sense an *idle* channel (in case no transmission takes place at that round), $(ii)$ sense a *busy* channel (in case two or more nodes transmit at the current round), or $(iii)$ *receive* a packet (in case exactly one node transmits at that round). Henceforth, we will sometimes say that a message is *successfully sent* if there is exactly one transmission at this round. Thus, if a message is successfully sent, all nodes will successfully receive it in this round, except for the sender itself (the sender does not know whether the transmission was successful).

In addition to these nodes there is an adversary. We allow the adversary to know the protocol and its entire history and to use this knowledge in order to jam the wireless channel at will at any round. Such an adversary is called *adaptive*. If in addition to that the adversary also knows (through physical carrier sensing) the *cur-*

*Mobihoc'11,* May 16-19, 2011 Paris, France

[1] A round may represent the time needed to send a message, e.g., a multiple of the $50\mu s$ unit in 802.11, depending on the message size.

[2] We believe that such a feedback mechanism is problematic in the broadcast setting as it increases the communication load, and its benefits are not clear either. Moreover, this assumption strengthens our result.

*rent* channel state, we call it *reactive*. That is, a reactive adversary can distinguish between the channel being currently idle (no node transmits) or busy (either because of a successful transmission, a collision of transmissions, or too much background noise) and can instantly make a jamming decision based on that information. Whenever the adversary jams the channel, all nodes will notice a busy channel. The nodes cannot distinguish between the adversarial jamming and a collision of two or more messages that are sent at the same time.

In order to study the degree of jamming activity needed by the adversary to prevent successful message transmissions, we use the notion of a $(T, 1-\epsilon)$-bounded adversary. An adversary is called $(T, 1-\epsilon)$-*bounded* for some $T \in \mathbb{N}$ and $0 < \epsilon < 1$ if for any time window of size $w \geq T$ the adversary can jam at most $(1-\epsilon)w$ of the time steps in that window. Moreover we assume that the $n$ nodes use an encryption mechanism that prevents the adversary from inspecting their messages.

As mentioned earlier, our goal is to design a leader election protocol that is self-stabilizing despite adversarial jamming. Following the usual notation in the self-stabilization literature, the *system state* is determined by the state of all *variables* in the system. That is, the protocol and any constants used by the protocol are assumed to be immutable and not part of the system state. A system is called *self-stabilizing* if and only if (1) when starting from any state, it is guaranteed to eventually reach a legal state (convergence) and (2) given that the system is in a legal state, it is guaranteed to stay in a legal state (closure), provided that there are no faults or membership changes in the system. In our case, roughly speaking, the legal state is the state in which we have exactly one leader. We will define the set of legal states more formally when we introduce our protocol. While our protocol is randomized and the leader election has to be performed under adversarial jamming, our protocol is still guaranteed to eventually elect exactly one leader from any initial state.

**Related Work.** Leader election is an evergreen in distributed algorithms research due to its numerous applications, and there exist many theoretical and practical results [4, 12, 18, 20, 22, 24, 28, 29]. Please refer to the following two books for a good introduction: Chapter 3 in [3] and Chapter 8 in [15]. A leader election algorithm should be as flexible as possible in the sense that a correct solution is computed *independently* of the initial network state. For instance, the algorithm should be able to react to a leader departure, or be able to cope with situations where for some reasons, multiple nodes consider themselves leaders. So-called *self-stabilizing algorithms* [10] with good convergence times are an active research topic (see e.g., the works on time-adaptive self-stabilization such as [19]), and several self-stabilizing leader election protocols are known already, e.g., [2, 8, 16] (see also the fault-contained solutions such as [13]). None of these approaches allows us to elect a leader in a wireless network that is exposed to harsh interference or even adaptive jamming. However, interruptions of communication is often unavoidable in wireless systems, and we believe that electing a leader can be particularly useful in such harsh environments.

Interference (either due to collisions or jamming) on the shared medium renders the task of electing a leader challenging; on the other hand, once a leader is determined, throughput may be improved significantly due to the coordinated medium access. It is well-known that jamming attacks are often simple and cheap to implement, and there exists a large body of literature on the subject [11, 14, 17, 25]. Only the closest results to ours can be discussed here, and for a broader overview on the field, we refer the reader to the literature reviews provided in the corresponding papers.

Classic defense mechanisms operate on the physical layer [21, 23] and there exist approaches both to *avoid* as well as to *detect* jamming. Spread spectrum and frequency hopping technologies have been shown to be very effective to avoid jamming with widely spread signals. These physical layer solutions are orthogonal to our work, and can improve the robustness of the protocol presented here further. However, the ISM frequency band used by IEEE 802.11 variants is too narrow to effectively apply spread spectrum techniques [7].

Recent work has also studied *MAC layer strategies* against jamming, including coding strategies (e.g., [9]), channel surfing and spatial retreat (e.g., [1, 31]), or mechanisms to hide messages from a jammer, evade its search, and reduce the impact of corrupted messages (e.g., [30]). Unfortunately, these methods do not help against an adaptive jammer with *full* information about the history of the protocol, like the one considered in our work.

The works closest to ours are the resilient MAC protocols studied in [5, 26, 27]. In particular, our adversarial model was introduced by Awerbuch et al. [5] who present a jamming-resistant MAC protocol that guarantees a constant throughput against an adaptive adversary in a single-hop environment. The MAC protocol has been adapted for multi-hop wireless networks [26] as well as for reactive jammers [27]; interestingly, a competitive throughput can be obtained even in these more general scenarios, although the nodes' cumulative sending probability may vary within a larger range compared to the single-hop, non-reactive scenario. Our work complements this line of research on MAC protocols [5, 26, 27] and focuses on a fundamental application in wireless networks: the problem of electing a leader. To achieve this, our leader election protocol builds upon the techniques described in [5] to adjust medium access probabilities in a multiplicative manner to resolve contention efficiently and recover from jammed time periods, and introduces a scheme on top of the MAC protocol that allows the nodes to obtain a consistent view on the application's state and the presence of potential leaders. (Note that we slightly adapted the medium access protocol itself and tailored it for the leader election problem: for instance, leaders increase their medium access probabilities faster than followers in order to increase the likelihood of a successful leader message transmission.)

A leader election protocol has already been sketched briefly in [5]. However, this algorithm is rather preliminary and not robust. For instance, if a leader leaves the network, the other nodes cannot realize its absence in order to, e.g., elect a substitute. In contrast, in this paper, we propose a self-stabilizing leader election protocol that converges to the desired state from *any* initial configuration. We believe that this is a vital property in real networks where membership is dynamic. Moreover, in contrast to [5], we focus on a *reactive jammer model*, because (1) many nodes support carrier sensing today; (2) the reactive model is more general and hence more difficult than the adaptive model; and finally, note that (3) a reactive model can also make sense in scenarios without jammers, e.g., in *co-existing networks*: many MAC protocols based on carrier sensing activate nodes during idle time periods. It turns out that protocols that perform well under adaptive jammers may have a high convergence time under reactive jammers, and hence additional techniques are required. For example, by selective jamming, a reactive adversary can find out certain (e.g., pseudo-random and secret) communication patterns and become even more powerful in the future. Moreover, in contrast to the protocol in [5], we synchronize of the nodes' sending probabilities (which also improves fairness).

**Our Contribution.** This paper presents SELECT ("SElf-stabilizing Leader EleCTion"), a protocol that solves the leader election problem in harsh environments—namely in wireless networks under adversarial *reactive* jamming—and in a self-stabilizing manner, independently of the initial network state. We believe that self-stabilization is a crucial feature in real networks where membership is often dynamic. Although our algorithm is randomized, we will present a formal proof that its correctness holds deterministically. Moreover, while our analysis is rather involved, the SELECT protocol itself is simple and hence easy to implement.

Concretely, in this paper we will derive the following theorem.

THEOREM 1.1. *Given an arbitrary initial configuration and in the absence of state faults, our leader election protocol reaches a state where there is exactly one leader and $n-1$ followers, despite a reactive $(T, 1-\epsilon)$-bounded jammer, for any $T$ and any constant $\epsilon > 0$.*

In SELECT, the nodes do not have to know anything about the system for the protocol to work. The only assumption that we need is that some fixed common parameter $\gamma$ used by the nodes satisfies $\gamma = O(1/(\log T + \log\log n))$. As $\log T$ and $\log\log n$ are small for all reasonable values of $T$ and $n$, this is scalable and not a critical constraint, as it leaves room for a super-polynomial change in $n$ and a polynomial change in $T$ over time.[3] Thus, in practice we expect that choosing $\gamma$ to be a sufficiently small constant yields a good performance for any practical network, which is confirmed by our simulations.

## 2. THE SELECT PROTOCOL

SELECT is based on the following idea. Each node $v$ maintains a parameter $p_v$ which describes $v$'s probability of accessing the medium at a given moment of time. That is, in each round, each node $v$ decides to transmit a message with probability $p_v$ (e.g., in an attempt to become a leader). (This is similar to classic random backoff mechanisms where the next transmission time $t$ is chosen uniformly at random from an interval of size $1/p_v$.) The nodes adapt and synchronize their $p_v$ values over time in a multiplicative increase multiplicative decrease manner, i.e., the value is lowered in times of high interference or increased during times where the channel is idling. However, $p_v$ will never exceed $\hat{p}$, for some constant $0 < \hat{p} < 1$.

In addition, each node maintains two variables, a threshold variable $T_v$ and a counter variable $c_v$. $T_v$ is used to estimate the adversary's time window $T$: a good estimation of $T$ can help the nodes recover from a situation where they experience high interference in the network. In times of high interference, $T_v$ will be increased and the sending probability $p_v$ will be decreased.

Initially, every node $v$ sets $c_v := 1$ and $p_v := \hat{p}$. Note however that while we provide some initial values for the variables in our description, our protocol is self-stabilizing and works for *any* initial variable values, as we will show in our proofs.

SELECT distinguishes between two node roles: *follower* and *leader*. We use $s_v$ to indicate the role of the node: $s_v = 1$ means that node $v$ is a leader, whereas $s_v = 0$ means $v$ is a follower. The basic idea of our protocol is to divide time into intervals of a small number of rounds specified by the constant parameter $b > 5$ (we use the variable $mc$ as a modulo counter); in the following, we will refer to a sequence of rounds between two consecutive $mc = 0$ events as a *b-interval*. (Of course, it can happen that all $b$ slots of an interval are jammed.)

Our protocol is based on the concept of so-called *leader slots*, special rounds—in each $b$-interval through which SELECT cycles—in which leaders are obliged to send an alive message (a so-called *leader message*) and in which followers keep silent. The idea is that the followers learn that the leader has left in case of an idling medium during a leader slot (of course, the leader slots may be jammed!) and a new election is triggered automatically.

SELECT uses four *leader slots*:[4] $ls_1, ls_2, ls_3$ and $ls_4$. Of course, in the beginning, all nodes may have different $ls$ values and may disagree on which slots during the $b$-interval are leader slots. However, over time, the nodes synchronize their states and a consistent view emerges. For the synchronization, five temporary variables $ls'_0, ls'_1, ls'_2, ls'_3$, and $ls'_4$ are used, which store future $ls$ values.

Depending on whether the node is of type follower or leader, the leader slots are updated differently: At the beginning of a new $b$-interval, a leader copies its $ls'_i$ values to the $ls_i$ values. A follower on the other hand copies the $ls'$ values "diagonally" in the sense that $ls'_i$ is copied to $ls'_{i+1}$ for $i \in \{0, 1, 2, 3\}$. As we will see, this mechanism ensures that an elected leader covers the leader slot $ls_3$ *of each follower*. (SELECT guarantees that the reactive adversary has no knowledge about the $ls_3$ slots at all until it is already too late to prevent a successful election.) Another special slot besides $ls_3$ is $ls'_0$ which is a random seed to mix the execution for increased robustness.

In Figure 1 we give the detailed formal description of the follower and the leader protocol, respectively. Recall that our algorithms can tolerate any initial values of $mc$, $p_v$, $T_v$, $c_v$, $s_v$, $s'_v$, $ls_1$, $ls_2$, $ls_3$, $ls_4$, $ls'_0$, $ls'_1$, $ls'_2$, $ls'_3$, $ls'_4$. For instance, in the beginning, all nodes $v$ may be leaders and for all $v$, $s_v = 1$. However, the fixed parameters used by the algorithms, namely $\hat{p}, \gamma$, or $b$, are assumed to be immutable.

Both the follower and the leader algorithm consist of three main parts. The $b$-interval wise update (Lines $2-4$) makes sure that $ls$ values are refreshed frequently. Lines $6-33$ (in case of a follower) and Lines $5-24$ (in case of a leader) are used for medium access in order to synchronize the nodes' states (by a message that includes $c_v$, $T_v$, and $p_v$ values) and give nodes the chance to become or remain leader (by a 'LEADER' message). The last sections of the algorithms are used to react to high interference (by reducing $p_v$) and to reset leader slots. The reason for checking whether $ls_3$ is undefined in Line 6 of the follower protocol is to keep the leader slots hidden from the reactive adversary until it is already too late to prevent a successful leader election.[5]

Both the follower and the leader protocol depend on the following crucial CONDITION.

DEFINITION 2.1 (CONDITION). *We define* CONDITION *(Line 37 for followers, and Line 28 for leaders) as the event that at least one 'LEADER' message was received during the past $b \cdot T_v$ steps.*

The idea is that if CONDITION is fulfilled, we know that the protocol is already in a good state. Moreover, we will see that the adversary cannot prevent CONDITION to become true for a long time as the $T_v$ values would continue to increase.

Finally, also note that leaders increase $p_v$ faster (i.e., by larger multiplicative factors) during idle rounds than followers. With this mechanism, SELECT improves the likelihood that a 'LEADER' message gets through and hence that a unique leader is elected.

[3] On the other hand, note that the assumption that the nodes know constant factor approximations of $n$ or $T$ directly would render the problem trivial. Moreover, such an assumption is unrealistic and non-scalable.

[4] It is an open question whether a protocol with less leader slots can be devised.

[5] This check would not be necessary against a non-reactive adversary.

```
Algorithm 1 Leader Election: Follower
$mc := c_v \mod b$
if $mc = 0$ then
    $ls_1 := ls'_0, ls_2 := ls'_1, ls_3 := ls'_2, ls_4 := ls'_3$
    $s_v := s'_v$
end if
if ($ls_3 =$ undefined) or ($mc \neq ls_1$ and $mc \neq ls_2$ and $mc \neq ls_3$ and $mc \neq ls_4$) then
    $v$ decides with $p_v$ to send a follower message
    if $v$ sends a follower message then
        the message contains:
        $cc_1 := ls'_0, cc_2 := ls'_1, cc_3 := ls'_2, cc_4 := ls'_3, c_{new} := c_v, T_{new} := T_v, p_{new} := p_v$
    end if
end if
if $v$ does not send a follower message then
    $v$ senses the channel
    if channel is idle then
        if $mc = ls_3$ then
            $s'_v := 1$
            $p_v := \hat{p}$
        else
            $p_v := \min\{(1+\gamma)p_v, \hat{p}\}$
        end if
    else if $v$ receives ‘LEADER’ then
        $s'_v := 0$
        $ls_3 := undefined$
        $ls'_2 := undefined$
    else if $v$ receives a tuple of $\{cc_1, cc_2, cc_3, cc_4, c_{new}, T_{new}, p_{new}\}$ then
        $T_v := T_{new}$
        $p_v := (1+\gamma)^{-1}p_{new}$
        $c_v := c_{new}$
        $ls'_0 := random(0, b-1)$
        $ls'_1 := cc_1, ls'_2 := cc_2, ls'_3 := cc_3, ls'_4 := cc_4$
    end if
end if
$c_v := c_v + 1$
if $c_v \geq b \cdot T_v$ then
    $c_v := 0$
    if (not CONDITION) then
        $p_v := (1+\gamma)^{-1}p_v, T_v := T_v + 1$
        $ls'_0 := undefined$, $ls'_1 := undefined$, $ls'_2 := undefined$, $ls'_3 := undefined$, $ls'_4 := undefined$
    else
        $T_v := \max\{T_v - 1, 4\}$
    end if
end if
```

```
Algorithm 2 Leader Election: Leader
$mc := c_v \mod b$
if $mc = 0$ then
    $ls_1 := ls'_1, ls_2 := ls'_2, ls_3 := ls'_3, ls_4 := ls'_4$
end if
if $mc = ls_1$ or $mc = ls_2$ or $mc = ls_3$ or $mc = ls_4$ then
    $v$ sends the leader message ‘LEADER’
else
    $v$ decides with $p_v$ to send ‘LEADER’
    if $v$ does not send ‘LEADER’ then
        $v$ senses the channel
        if channel is idle then
            $p_v := \min\{(1+\gamma)^2 p_v, \hat{p}\}$
        else if $v$ receives a message then
            $p_v := (1+\gamma)^{-1}p_v$
            if message is ‘LEADER’ then
                $s_v := 0$, $s'_v := 0$
                $ls_3 := undefined$, $ls'_2 := undefined$
            else if message is a follower message, i.e., a tuple of $\{cc_1, cc_2, cc_3, cc_4, c_{new}, T_{new}, p_{new}\}$ then
                $c_v := c_{new}, T_v := T_{new}$
                $ls'_1 := cc_1, ls'_2 := cc_2, ls'_3 := cc_3, ls'_4 := cc_4$
            end if
        end if
    end if
end if
$c_v := c_v + 1$
if $c_v \geq b \cdot T_v$ then
    $c_v := 0$
    if (not CONDITION) then
        $p_v := (1+\gamma)^{-1}p_v, T_v := T_v + 1$
        $ls'_0 := undefined$, $ls'_1 := undefined$, $ls'_2 := undefined$, $ls'_3 := undefined$, $ls'_4 := undefined$
    else
        $T_v := \max\{T_v - 1, 4\}$
    end if
end if
```

**Figure 1: Algorithm for followers (*left*) and leaders (*right*).**

## 3. ANALYSIS

This section shows that the randomized SELECT protocol is guaranteed to eventually reach a situation where there is exactly one leader and $n - 1$ followers. We make use of the following definitions. First, we define the system state.

DEFINITION 3.1 (STATE AND SYSTEM STATE). *The* state of node *$v$ is determined by the state of the variables $p_v$, $T_v$, $c_v$, $s_v$, $s'_v$, $mc$, $ls'_0$, $ls_1$, $ls'_1$, $ls_2$, $ls'_2$, $ls_3$, $ls'_3$, $ls_4$ and $ls'_4$. The* state of the system *is the set of the states of all nodes.*

We use the following $LS_L$ set to describe the union of all possible leader slot values present in the system.

DEFINITION 3.2 (THE $LS_L$ STATE SET). *For any given system state, let $LS_L = \{ls_1(v), ls_2(v), ls_3(v), ls_4(v) \mid v \text{ is leader}\} \setminus \{undefined\}$.*

The system can be in several special states which are formalized next: follower states, pre-leader states, and leader states. Let $[b] = \{0, \ldots, b-1\}$.

DEFINITION 3.3 (FOLLOWER STATE). *A state $S$ is called a* follower state, *denoted by $S \in$ FOLLOWER, if all the following conditions hold. **(i)** All nodes are followers ($\forall v \in V : s_v = 0$); **(ii)** for every node $v$: $ls_1(v), ls_2(v), ls_3(v), ls_4(v) \in [b] \cup \{undefined\}$, $ls'_1(v)$, $ls'_2(v)$, $ls'_3(v)$, $ls'_4(v) \in [b] \cup \{undefined\}$, $ls'_0(v) \in [b]$; **(iii)** the follower nodes can be partitioned into two sets $\{v\}$ and $V \setminus \{v\}$, according to their $ls'$ values ($v$ is the node that successfully sent the last follower message); for each $w \in V \setminus \{v\}$: $ls'_1(w) = ls'_0(v)$, $ls'_2(w) = ls'_1(v)$, $ls'_3(w) = ls'_2(v)$, $ls'_4(w) = ls'_3(v)$, and $ls_2(w) = ls_1(v)$, $ls_3(w) = ls_2(v)$, and $ls_4(w) = ls_3(v)$; **(iv)** for any pair of follower nodes $v, w \in V$ with $ls'_2(v) \in [b]$ and $ls_3(v) \in [b]$, $c_v = c_w$ and $T_v = T_w$.*

We use the concept of so-called *pre-leader states*, i.e., states that result from follower states before some nodes become leaders.

DEFINITION 3.4 (PRE-LEADER STATE). *A state $S$ is called a* pre- leader state, *denoted by $S \in \mathrm{PRE-LEADER}$, if it is a follower state, and at least one follower node $v$ has $s'_v = 1$.*

While in the beginning, the leader sets may be large as each node regards different slots during the $b$-interval as the "leader slots", over time the values synchronize and the $LS$ sets become smaller. This facilitates a fast leader (re-) election.

DEFINITION 3.5 (LEADER STATE). *A state $S$ is called a* leader state*, denoted by $S \in$ LEADER, if all the following conditions are satisfied:*

***(i)** There is at least one leader, i.e., $|\{v|v \in V : s_v = 1\}| \geq 1$; **(ii)** for every node $v$, $ls_1(v), ls_2(v), ls_3(v), ls_4(v) \in [b] \cup \{undefined\}$, $ls'_1(v), ls'_2(v), ls'_3(v), ls'_4(v) \in [b] \cup \{undefined\}$, $ls'_0(v) \in [b]$; **(iii)** let $v$ be any follower and let $w$ be any follower or leader, then $ls_3(v) \in \{ls_1(w), ls_2(w), ls_3(w), ls_4(w)\} \cup \{undefined\}$, $ls'_2(v) \in \{ls'_0(w), ls'_1(w), ls'_2(w), ls'_3(w)\} \cup \{undefined\}$; **(iv)** $|LS_L| \leq 5$; **(v)** for every follower $w$ with $ls_3(w) \in [b]$ or $ls'_2(w) \in [b]$, $c_w = c_v$ and $T_w = T_v$ for any leader $v$.*

So in a leader state, it holds that any follower's $ls_3$ and $ls'_2$ slots are covered by either another follower's $ls$ and $ls'$ slots, or a leader's $ls$ and $ls$ slots (cf Condition (iii)).

Finally, it is useful to define safe and legal states.

DEFINITION 3.6 (SAFE AND LEGAL STATE). *A system state $S$ is called* safe *(denoted by $S \in$ SAFE) if $S \in$ FOLLOWER or $S \in$ LEADER, and* legal *(denoted by $S \in$ LEGAL) if $S$ is safe and there is exactly one node $v$ with $s_v = 1$.*

Thus, according to our definitions, any legal state is also a safe state. In the following, let $\mathcal{S}$ be the set of all possible system states, SAFE $\subset \mathcal{S}$ be the set of all *safe* system states and LEGAL $\subset$ SAFE be the set of all *legal* system states.

The proof of Theorem 1.1 unfolds in a number of lemmas. An interesting property of our randomized algorithm is that it is *guaranteed* to be correct, in the sense that deterministically exactly one leader is elected; only the runtime is probabilistic (i.e., depends on the random choices made by SELECT).

First, we study leader messages.

LEMMA 3.7. *For any network state it holds that if a leader successfully transmits a 'LEADER' message, the system will immediately enter a legal state.*

PROOF. When a node (either follower or leader) receives a 'LEADER' message, it sets $ls_3$ and $ls'_2$ to $undefined$ (Lines $22-25$ in Figure 1 *left*; after Lines $15-17$ of Figure 1 *right*), and considers itself a follower. Thus, in the new state, there is exactly one leader (the sender of the 'LEADER' message) and $n-1$ followers. The state is also a safe state, namely a leader state: Conditions $(i)$ and $(ii)$ are fulfilled trivially. Condition $(iv)$ also holds as there is only one leader that has four slots. Condition $(iii)$ is fulfilled because nodes receiving a 'LEADER' message reset their slots $ls_3$ and $ls'_2$; since $ls_3$ and $ls'_2$ are undefined for a follower, also Condition $(v)$ holds. □

We next consider what happens if nodes hear a message sent by a follower.

LEMMA 3.8. *For any network state it holds that when a follower successfully transmits a message, the system is guaranteed to enter a safe state at the beginning of the next $b$-interval.*

PROOF. First note that if a leader message gets through before the next $b$-interval, the claim holds trivially due to Lemma 3.7.

Otherwise we distinguish two cases: (A) For every node $v$, $s'_v = 0$ (not pre-leader) and $s_v = 0$ (not leader) by the end of current $b$-interval. (B) There is at least one node $v$ with either $s'_v = 1$ (pre-leader) or $s_v = 1$ (leader) by the end of current $b$-interval.

In Case (A), after the follower message has been successfully sent, there are still $n$ followers and no leaders or pre-leaders. We will show that the system enters the follower state at the beginning of the next $b$-interval. Let us refer to the follower node that sent the message by $v$ and to any remaining node by $w$. When $w$ receives the message from $v$ (Lines $26-32$ in Figure 1 *left*), it sets $ls'_1(w) := ls'_0(v)$, $ls'_2(w) := ls'_1(v)$, $ls'_3(w) := ls'_2(v)$, and $ls'_4(w) := ls'_3(v)$. The $c$ values become the same ($c_w = c_v$), and $T_w := T_v$. The new state therefore fulfills the follower state conditions: Clearly, Conditions $(i)$, $(ii)$, and $(iv)$ are fulfilled immediately, and Condition $(iii)$ holds as well, as for all followers $w$ that did not send a message and follower $v$ which sent a message, at the beginning of the next $b$-interval: $ls_3(w) = ls'_2(w) = ls'_1(v) = ls_2(v)$, $ls_3(v) = ls'_2(v) = ls'_3(w) = ls_4(w)$, and $ls_1(v) = ls_2(w) = ls'_0(v) = ls'_1(w)$.

For Case (B), observe that during the remainder of the $b$-interval the number of pre-leader nodes with $s'_v = 1$ cannot decrease, and hence there will be at least one leader at the beginning of the next $b$-interval. We now show that the new state will indeed be a leader state as nodes "synchronize" with the follower node that sent the message. Without loss of generality, assume that node $u$ is the last follower that successfully sent a follower message in the current $b$-interval. Let us refer to the other follower nodes by $v_1$ and to the leader nodes or the pre-leader nodes (i.e., the followers $v$ with $s'_v = 1$) by $v_2$. Again, Conditions $(i)$ and $(ii)$ are fulfilled trivially. As for Condition $(iii)$, we need to consider two sub-cases:

(Case 1) No node experienced an idle channel in its $ls_3$ slot after the message has been successfully sent. If this is the case and follower $u$ is not a pre-leader, it holds that for follower $v_1$: $ls'_2(v_1) = ls'_2(v_2) = ls'_1(u)$ in the current $b$-interval, and $ls_3(v_1) = ls_2(v_2) = ls_2(u)$ at the beginning of the next $b$-interval; on the other hand, if follower $u$ is a pre-leader, then in the current $b$-interval it holds that for follower $v_1$: $ls'_2(v_1) = ls'_2(v_2) = ls'_1(u)$, and $ls_3(v_1) = ls_2(v_2) = ls_1(u)$ at the beginning of the next $b$-interval. Hence, Condition $(iii)$ holds. Regarding the cardinality of the leader set $LS_L$, observe that at the beginning of the next $b$-interval, if $u$ is not a pre-leader, all leaders will have $ls_1 = ls'_0(u), ls_2 = ls'_1(u), ls_3 = ls'_2(u), ls_4 = ls'_3(u)$, and hence $LS_L = \{ls'_0(u), ls'_1(u), ls'_2(u), ls'_3(u)\}$, therefore $|LS_L| \leq 5$; otherwise, if $u$ is a pre-leader, then $LS_L = \{ls'_0(u), ls'_1(u), ls'_2(u), ls'_3(u), ls'_4(u)\}$, therefore $|LS_L| \leq 5$.

(Case 2) One or more nodes experienced an idle channel in their $ls_3$ slots after the message has been successfully sent. In the following, we prove this case correct assuming that $u$ is a follower and not a pre-leader. If $u$ is a pre-leader, the proof is analogous.

1. If $v_1$ experienced the idle channel at its $ls_3$ time slot, and became a pre-leader:
   Note that a node $v_1$ may experience an idle channel after receiving the message from $u$ and hence become a pre-leader, however Condition $(iii)$ is still satisfied, as it holds that for follower $u$: $ls'_2(u) = ls'_3(v_2) = ls'_3(v_1)$ in the current $b$-interval and $ls_3(u) = ls_3(v_2) = ls_3(v_1)$ at the beginning of the next $b$-interval. As for the cardinality of the leader set $LS_L$, observe that at the beginning of the next $b$-interval, all leaders will have $ls_1 = ls'_0(u), ls_2 = ls'_1(u), ls_3 = ls'_2(u), ls_4 = ls'_3(u)$, and hence $LS_L = \{ls'_0(u), ls'_1(u), ls'_2(u), ls'_3(u)\}$, therefore $|LS_L| \leq 5$.

2. If $u$ experienced the idle channel at its $ls_3$ time slot, and became a pre-leader:
   If node $u$ experienced an idle channel after successfully

sending the message, $u$ became a pre-leader, and we have for a follower $v_1$, $ls'_2(v_1) = ls'_2(v_2) = ls'_1(u)$ in the current $b$-interval and $ls_3(v_1) = ls_2(v_2) = ls_1(u)$ at the beginning of the next $b$-interval. Hence, Condition $(iii)$ is satisfied. As for $|LS_L|$, observe that at the beginning of the next $b$-interval, for a leader $v_2$, $ls_1 = ls'_0(u), ls_2 = ls'_1(u), ls_3 = ls'_2(u), ls_4 = ls'_3(u)$, while for the remaining leader $u$, it holds that $ls_1 = ls'_1(u), ls_2 = ls'_2(u), ls_3 = ls'_3(u), ls_4 = ls'_4(u)$. Hence, also in this case, we have that $|LS_L| \leq 5$.

Finally, Condition $(v)$ is true for both of the sub-cases, because the $c_v$ and $T_v$ values are "synchronized" when the follower message is received (Lines 27 and 29 in Figure 1 *left*; Line 19 in Figure 1 *right*). □

An important property of SELECT is that once it is in a safe state, it will remain so in future (given that there are no external changes). Similar properties can be derived for other states, as we will see.

LEMMA 3.9. *Once the system is in a safe state, it will remain in a safe state in the future.*

PROOF. We study what can happen in one round, and show that in each case, the safety properties are maintained. In a round, (A) either a 'LEADER' message is successfully sent, (B) a follower message is successfully sent, (C) there are collisions or the channel is jammed, or (D) there is an idle channel.

In Case (A), the claim directly follows from Lemma 3.7 and from the fact that safe states are a super set of the legal states (SAFE ⊃ LEGAL). In Case (B), the claim follows from Lemma 3.8 and by the fact that the system is in the safe state already.

In Case (C), if the channel is blocked, follower nodes (even those which sent a message in this round) do not change their state except for the synchronized rounds in Lines $35 - 43$, and similarly for the leaders in Lines $26 - 34$. Our protocols guarantee that the leaders have the same $c_v$ and $T_v$ values as the followers when $ls_3$ and $ls'_2$ are valid, and since the leaders experience the same number of successful transmissions and idle time steps as the followers do (single-hop network), the claim follows.

If there is an idle channel (Case (D)), all nodes $v$ for which $ls_3(v) = mc$ will set $s'_v = 1$ in the current $b$-interval, while other values remain the same. It is clear that from this point on until the end of the current $b$-interval, the claim holds. Moreover, as we show next, the claim is still true at the beginning of next $b$-interval. If $ls_3(v)$ is undefined, then the claim holds trivially, as no states will change in this case. If $ls_3(v) = mc$ for any node $v$ and the nodes experience an idle channel, there is no leader since, if there was a leader, according to Condition $(iii)$ of the leader state definition (Definition 3.5), a follower's $ls_3$ slot would always be covered by a leader slot of a leader, which yields the contradiction. Hence, the current safe state must be a pre-leader state. Let $v$ denote the followers that have $s'_v = 0$ (i.e., they are not pre-leaders); let $u$ denote the followers with $s'_u = 1$ (pre-leaders). In the current $b$-interval, we have $ls'_2(v) \in \{ls'_0(u), ls'_1(u), ls'_2(u), ls'_3(u)\} \cup \{undefined\}$, which is true according to Condition $(iii)$ of the follower state definition (Definition 3.3). Then, at the beginning of next $b$-interval, $u$ will become a leader, and hence we have $ls_3(v) = ls'_2(v)$, $ls_1(u) = ls'_1(u)$, $ls_2(u) = ls'_2(u)$, and $ls_3(u) = ls'_3(u)$. This implies that $ls_3(v) \in \{ls'_0(u), ls_1(u), ls_2(u), ls_3(u)\}$, which satisfies Condition $(iii)$ of the leader state Definition 3.5. Conditions $(i)$ and $(ii)$ are clearly satisfied. Condition $(iv)$ holds simply because we have shown (in Lemma 3.8, Case $(B)$), when there is an idle time step, $|LS_L| \leq 5$. Condition $(v)$ is true because we always synchronize the $c_v$ and $T_v$ values. □

LEMMA 3.10. *Once a system is in a leader state, it will remain in a leader state in the future.*

PROOF. Lemma 3.9 tells us that the system will never leave a safe state. Therefore, it remains to prove that there will always be at least one node $v$ with $s_v = 1$. This clearly holds as the only way a leader can become a follower again is by receiving a 'LEADER' message (see Lines $15 - 17$), which of course implies that another leader is still active and remains to be a leader. Also, since we are in a leader state, Condition $(v)$ holds and it further implies that leaders will never invalidate their $ls$ slots before the followers. This guarantees that the protocol will never get out of a leader state. □

LEMMA 3.11. *Once a system is in a legal state, it will remain in a legal state in the future.*

PROOF. By Lemma 3.9, we know that our system will never leave a safe state again, and hence, we only need to prove that there will always be exactly one node $v$ with $s_v = 1$. This is true because in the safe state, a follower node $w$ can never become a leader, as its $ls_3(w)$ slot is covered by the leader $v$: $ls_3(w) \in \{ls_1(v), ls_2(v), ls_3(v), ls_4(v)\}$ and $ls'_2(w) \in \{ls'_0(v), ls'_1(v), ls'_2(v), ls'_3(v)\}$ (Condition $(iii)$ of leader state). Since a follower will never send a 'LEADER' message, $v$ will remain a leader forever, which proves the claim. □

Regarding convergence, note that the system quickly enters a safe state, deterministically.

LEMMA 3.12. *For any initial system state with $\hat{T} = \max_v T_v$, it takes at most $b \cdot \hat{T}$ rounds until the system is in a safe state.*

PROOF. We distinguish three cases: if a leader message gets through sometimes in these rounds, then the claim holds by Lemma 3.7; if a follower message gets through, then the claim holds by Lemma 3.8. If within $\max_v T_v b$ rounds neither a follower message nor a leader message gets through, all nodes will have to reset their $ls$ slots (since CONDITION in Line 37 (Figure 1 *left*) resp. Line 28 (Figure 1 *right*) is not met). This however constitutes the safe state (all conditions fulfilled trivially), which is maintained according to Lemma 3.9. □

Armed with these results, we can prove convergence.

LEMMA 3.13. *For any safe state,* SELECT *will eventually reach a legal state.*

PROOF. We divide the proof in two phases: the phase where the protocol transitions to the leader state from the follower state, and the phase where it transitions to the legal state from the leader state.

1. Follower state to leader state

   If CONDITION is fulfilled, we know that a 'LEADER' message got through and the system is in a legal state (and hence also in a leader state). As long as CONDITION is not fulfilled, $T_v$ is increasing for each node $v$. So eventually, $\hat{T} = \max_v T_v \geq 2T/b$. We can also provide a lower bound on the cumulative probability $p$. W.l.o.g. suppose that $T \geq (3/\epsilon)\log_{1+\gamma} n$ (a smaller $T$ will only make the jammer less flexible and weaker). Suppose that $p$ is at most $\epsilon/4$ throughout some $T$-interval $I$. Then it follows from the standard Chernoff bounds that there are at most $\epsilon T/3$ busy steps in $I$ with high probability.[6] If this is true, then no matter how the adversary jams during $I$, at least

[6] "With high probability", or short "w.h.p.", means a probability of at least $1 - 1/n^c$ for any constant $c > 0$.

$(1-\epsilon/3)T - (1-\epsilon)T = 2\epsilon T/3$ non-jammed steps will be idle, which implies that the cumulative probability at the end of $I$ will be by a factor of at least $(1+\gamma)^{\epsilon T/3} \geq n^3$ higher than at the beginning of $I$. Using this insight, it follows that eventually a $T$-interval is reached with $p > \epsilon/4$. Once such a $T$-interval has been reached, it is easy to show that $p$ will not get below $1/n^2$ any more w.h.p. so that for every $T$-interval afterwards there is a time point $t$ with $p > \epsilon/4$ w.h.p. So infinitely often the following event can take place with some lower-bounded, positive probability:

Consider two consecutive $T$-intervals $I_1$ and $I_2$ starting at a time when $c_v = 0$ for every node $v$. Suppose that $I_1$ just consists of busy steps and $I_2$ just consists of idle time steps. Then the adversary has to leave $\epsilon T$ busy time steps in $I_1$ non-jammed and $\epsilon T$ idle time steps in $I_2$ non-jammed. For $I_1$, there is a positive probability in this case that exactly 3 messages from different nodes are successfully sent in 3 different $b$-intervals. In this case, all but one follower respect the leader slots (as their $ls_3$-value is defined) while the follower that sent the last successful message may still send out messages at *all* time steps (as its $ls_3$-value is still undefined, see Line 6 of the follower protocol). Thus, it is indeed possible that all time steps in $I_1$ are busy. Up to that point, the adversary has not learned anything about the leader slots. In $I_2$, there is also a positive probability that none of the followers transmits a message throughout $I_2$ so that all time steps are idle. As the adversary does not know which of them is a leader slot and has to leave $\epsilon T$ non-jammed, there is a positive probability that $ls_3$ is non-jammed, and some of the followers become pre-leaders and then leaders.

Thus, the expected time to get from a follower to a leader state is finite.

2. Leader state to legal state

If there is only one leader in the leader state, the system is already in a legal state by definition. If there is more than one leader, then we distinguish between the following cases. If CONDITION is fulfilled, we know that a 'LEADER' message got through and the system is in a legal state. Otherwise, the leaders will invalidate all of their $ls$ slots once their $c_v$ values are reset to 0. At this point there is a positive probability that for the next $T$ steps a 'LEADER' message is successfully sent. As the adversary has to leave $\epsilon T$ time steps non-jammed, at least one 'LEADER' message will be successfully transmitted within these $T$ steps so that the system reaches a legal state.

Analogous to the followers in the previous case, one can lower bound the cumulative probability of the leaders (in fact, the leaders will eventually reach a time point with a cumulative probability of $\Omega(\epsilon)$ as they increase their probabilities in case of an idle channel more aggressively than the followers) so that the chance above of successfully transmitting a 'LEADER' message repeats itself infinitely often with a lower-bounded positive probability. Thus, the expected time to get from a leader to a legal state is finite as well.

From these cases, the lemma follows. □

# 4. EXPERIMENTS

We conducted several simulations to study the behavior of SELECT under different types of jammers and interference.

## 4.1 Performance under Jamming

For our formal analysis, we introduced the notion of a $(T, 1-\epsilon)$-bounded adversary for some $T \in \mathbb{N}$ and $0 < \epsilon < 1$ which denotes that for any time window of size $w \geq T$ the adversary can jam at most $(1-\epsilon)w$ of the time steps in that window. While our protocol is provably robust to *any* adversary meeting these constraints, for our simulations, we will need to focus on specific instantiations. For example, we will consider an adversary that reactively jams all non-idle time periods only (as long as the budget is not used up), in order not to waste energy jamming idling periods.

We consider jammers of different powers, one that can block the channel 90% of the entire time, one that blocks 70% of the time, and a "weak" one that blocks 50% of the time (i.e., $\epsilon \in \{0.1, 0.3, 0.5\}$, resp.). We set $T = 100$ and consider a $b$-interval (see Figure 1) with parameter $b = 15$ (smaller $b$ values are possible as well). Experiments are repeated 50 times for each individual setting, and average values are recorded correspondingly. We run each experiment until one and only one leader is elected.

We conducted experiments with different types of reactive jammers: jammers $ADV_{\text{rand}}$ which interrupt transmissions *at random*, jammers $ADV_{\text{busy}}$ which only jam busy periods where one or more nodes transmit, and jammers $ADV_{\text{idle}}$ which jam the channel whenever it is idle. Concretely, for $ADV_{\text{busy}}$ and $ADV_{\text{idle}}$ we assume that the adversary will jam each busy resp. idle time period until the "jamming budget" is used up for this $T$-period. For $ADV_{\text{rand}}$ we set the jamming probability per round equal to $(1-\epsilon)$. $ADV_{\text{idle}}$ may appear less challenging to the deal with. However, note that an adversary may be able to lead a protocol to suboptimal states by jamming idle time periods. Moreover, this scenario also describes interference from co-existing networks where nodes are activated in quiet times. Hence, this adversary constitutes an interesting case that should not be neglected in the analysis.

Recall from Lemma 3.12 that from any initial state, the safe state is reached quickly, and hence, we are mainly interested in the convergence time from the safe state to the legal state. Figure 2 (*left*) plots the corresponding convergence times. At first sight the runtime may appear to be rather high. For example, under an adversary $ADV_{\text{rand}}$ that jams 90% of the entire time, it takes a few thousand time steps. However, note that this result implies that during the merely a few hundred non-jammed time steps, the five hundred nodes are able to successfully coordinate the medium access among themselves—without being able to distinguish between time periods with collisions and time periods that are jammed!— and use the computed access probabilities to elect a leader. We believe that when taking this into account, and although we do not have any lower bounds, the convergence time is very good and probably cannot be improved much with alternative schemes.

Figure 2 (*middle* and right) presents the corresponding convergence times for the reactive jammers $ADV_{\text{busy}}$ and $ADV_{\text{idle}}$. As expected, jamming the busy channel yields higher convergence times, also when comparing these results to our experiments with $ADV_{\text{rand}}$. In contrast, interestingly, for $ADV_{\text{idle}}$, the runtime is fairly independent of the adversarial power: a reactive jammer blocking idle channels gives similar results as $ADV_{\text{rand}}$. Clearly, among the scenarios we investigated, the most effective strategy for the adversary is to reactively jam the busy time periods as long as the total number of jammed time steps does not exceed $(1-\epsilon)\cdot T$.

Figure 3 complements Figure 2 by studying the execution times in smaller networks.

Our protocol aims to quickly reach a cumulative sending probability around a small constant, such that on expectation, roughly one node will try to transmit a message in a non-jammed step. Thus, given the constant probability of having a successful trans-

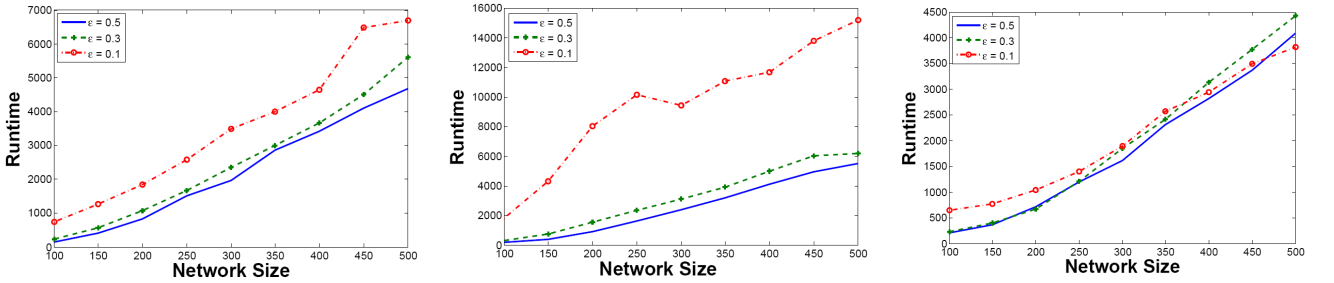

**Figure 2: *Left:* Convergence time from safe state to legal state, where the adversary $ADV_{\text{rand}}$ jams the channel. We ran our protocol until exactly one leader is elected. *Middle:* Convergence time from safe state to legal state, where a reactive adversary $ADV_{\text{busy}}$ jams the channel when one or more nodes are transmitting. We ran SELECT until only one leader is elected. *Right:* Convergence time from safe state to legal state under the reactive $ADV_{\text{idle}}$ adversary.**

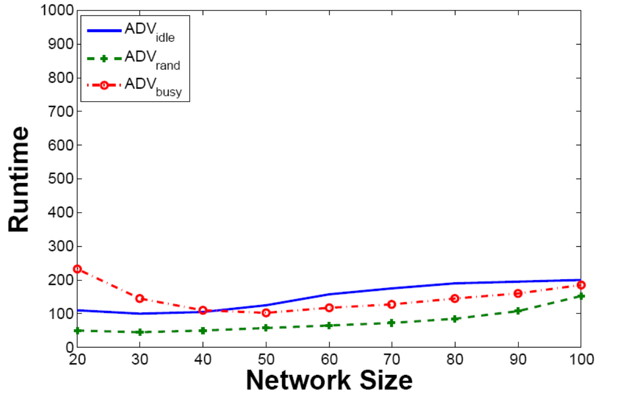

**Figure 3: Like Figure 2, but for smaller networks and using $\epsilon = 0.5$ and $\hat{p} = 1/24$.**

mission, a follower messages will get through soon, the nodes synchronize, and the $ls$ slots are defined as well. Since the leaders' sending probabilities reach higher values more quickly than the sending probabilities of the followers (according to Line 12 of Figure 1 *right*), a leader message gets through soon, yielding a legal state. We consider two initial states, a "well-initialized one" where all nodes have the maximum access probability $\hat{p}$ (in simulation we set $\hat{p} = 1/24$), where there is no leader in the network, and where the $ls$ and $ls'$ slots are all invalidated (according to Definition 3.3, this implies that we are in the follower state); and one with "arbitrary initialization" where the roles and variables are chosen at random (each node is either follower or leader, $p_v$ is chosen uniformly at random between 0 and 1, and the $ls$ values uniformly at random between 0 and $b-1$). Our experiments show that both scenarios yield similar results, which indicates that the convergence time of self-stabilization if fairly independent of the initial state.

Figure 4 (*left*) shows a typical trace of the cumulative probabilities over time when the protocol is well initialized, i.e., the protocol starts from the follower state. Initially, all nodes are followers, and we will denote the cumulative sending probability of the followers by $p_F$, and the cumulative sending probability of the leaders by $p_L$. At beginning, $p_F > 500 \cdot \hat{p} > 10$ while $p_L = 0$. As time goes on, $p_F$ decreases quickly until it falls in an interval of small constant range (i.e., $p_F < 10$), and multiple successful transmissions happen which synchronize the nodes' $ls$ and $ls'$ values. Next, multiple leaders are elected because many followers sense an idle time step in their $ls_3$ slot. That is why $p_L$ emerges at the same point in time as $p_F$ decreases dramatically to a value between 0 and 1. Then, the nodes continue to adjust their transmission probabilities depending on the channel state, until the first leader message gets through and all the other leaders become followers; this yields the quick decrease of $p_L$ and increase of $p_F$ accordingly. One and only one leader is elected after this point. Subsequently, both $p_F$ and $p_L$ remain within a small constant range. Figure 4 (*right*) shows the cumulative probabilities when the protocol starts from an "arbitrary" state ($p_v$, $T_v$, leaders and follower roles, etc. chosen at random). In the beginning, there are both followers and leaders in the network. It can be seen that SELECT converges fast, similarly to the well-initialized case. After the legal state is reached, both $p_F$ and $p_L$ also remain in a small constant range.

## 4.2 Co-existing Networks

Our leader election protocol is robust to arbitrary (but bounded with respect to time) interruptions of the availability of the medium, and it is convenient to regard these interruptions as caused by a malicious adversary. However, there are many other forms of interference to which our protocol is resilient and under which the few available time slots can be exploited effectively. In the following, we briefly report on one more source of interference, namely co-existing protocol instances. Concretely, we remove the jammer from the network and we compare the performance of our leader election protocol when run alone to situations where additional networks (of the same size) are concurrently trying to elect a leader and interfere with the other protocol instances accordingly.

Figure 5 (*left*) plots the averaged runtime until successful leader election for one, two, three and four co-existing networks, as a function of the corresponding sub-network sizes (i.e., four co-existing networks imply a four times larger total number of nodes). Our results indicate that each additional interfering network increases the runtime by a factor corresponding to the additional nodes. The convergence time among the co-existing networks exhibits a high fairness, as can be seen in Figure 5 (*right*): in all networks, a leader is elected almost at the same time.

# 5. CONCLUSION

This paper has introduced the first self-stabilizing leader election protocol, SELECT, for wireless networks operating in harsh

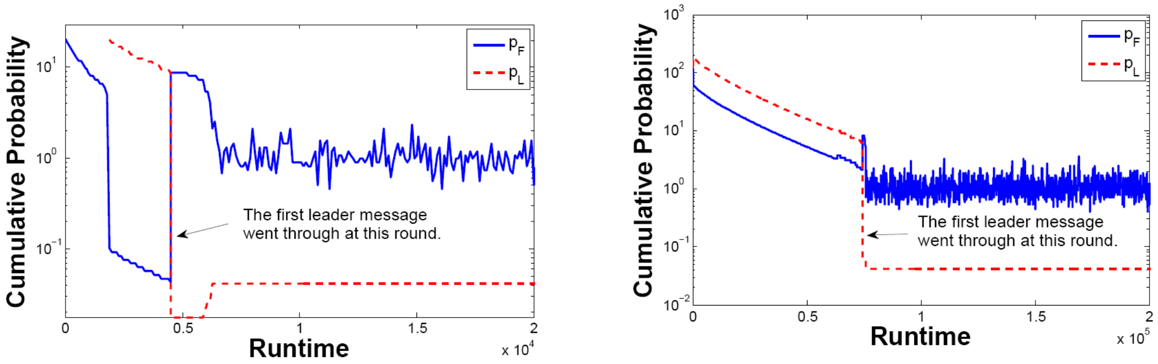

**Figure 4: *Left:* Fast convergence of $p_F$ and $p_L$ ($\epsilon = 0.5$, $T = 100$, network with 500 nodes) under $ADV_{\text{busy}}$ when the protocol starts from a safe state. *Right:* Corresponding convergence of $p_F$ and $p_L$ when the protocol starts from an arbitrary state.**

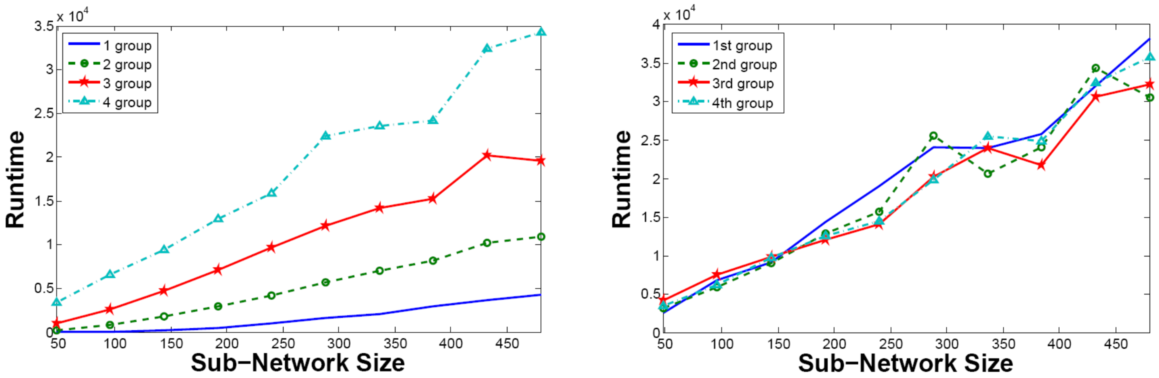

**Figure 5: *Left:* Convergence time of co-existing networks (as a function of their individual sizes) performing the leader election algorithm. *Right:* Fair convergence time among co-existing networks.**

environments, e.g., environments with hard-to-predict interference from co-existing networks or environments subject to (both adaptive and reactive) adversarial jamming. Although the nodes are not able to distinguish between collisions due to external interference or jamming and concurrent transmissions of other nodes in the network, they are able to coordinate access to the medium in the few and arbitrary time periods without external interference, and subsequently elect a leader in a robust manner. Although our protocol is randomized, it yields deterministic guarantees. There are several important open directions for future research. For example, the formal study of convergence times under different adversaries is an open problem. Another open problem is the generalization of our algorithms to multi-hop networks where leaders need to be elected in different regions (e.g., in order to construct a sparse backbone).

## Acknowledgments

This work was supported in part by NSF awards CCF-0830791 and CCF-0830704, and by the DFG-Project SCHE 1592/1-1. We would like to thank our shepherd Vijay Subramanian from Northwestern University for his help during the preparation of the final manuscript.

# 6. REFERENCES

[1] G. Alnifie and R. Simon. A multi-channel defense against jamming attacks in wireless sensor networks. In *Proc. of Q2SWinet '07*, pages 95–104, 2007.

[2] Gheorghe Antonoiu, Gheorghe Antonoiu, Pradip K Srimani, and Pradip K Srimani. A self-stabilizing leader election algorithm for tree graphs. *Journal of Parallel and Distributed Computing*, 34:227–232, 1996.

[3] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (Chapter 3)*. John Wiley & Sons, 2004.

[4] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proc. STOC*, 1987.

[5] Baruch Awerbuch, Andrea Richa, and Christian Scheideler. A jamming-resistant mac protocol for single-hop wireless networks. In *Proc. of PODC '08*, 2008.

[6] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa. On the performance of IEEE 802.11 under jamming. In *Proc. of IEEE Infocom '08*, 2008.

[7] T. Brown, J. James, and A. Sethi. Jamming and sensing of encrypted wireless ad hoc networks. In *Proc. of MobiHoc '06*, pages 120–130, 2006.

[8] Shukai Cai, Taisuke Izumi, and Koichi Wada. Space complexity of self-stabilizing leader election in passively-mobile anonymous agents. In *Proc. 16th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2009.

[9] J.T. Chiang and Y.-C. Hu. Cross-layer jamming detection and mitigation in wireless broadcast networks. In *Proc. of MobiCom '07*, pages 346–349, 2007.

[10] Edsger W. Dijkstra. Self-stabilization in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[11] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, Dariusz R. Kowalski, Calvin Newport, Fabian Kuhn, and Nancy Lynch. Reliable distributed computing on unreliable radio channels. In *Proc. 2009 MobiHoc S3 Workshop*, 2009.

[12] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[13] Sukumar Ghosh and Arobinda Gupta. An exercise in fault-containment: self-stabilizing leader election. *Inf. Process. Lett.*, 59(5):281–288, 1996.

[14] S. Gilbert, R. Guerraoui, and C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. In *Proc. of OPODIS '06*, 2006.

[15] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzicka, and W. Unger. *Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance (Chapter 8)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[16] Gene Itkis, Chengdian Lin, and Janos Simon. Deterministic, constant space, self-stabilizing leader election on uniform rings. In *Proc. 9th International Workshop on Distributed Algorithms (WDAG)*, pages 288–302, 1995.

[17] C.Y. Koo, V. Bhandari, J. Katz, and N.H. Vaidya. Reliable broadcast in radio networks: The bounded collision case. In *Proc. of PODC '06*, 2006.

[18] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Trans. Program. Lang. Syst.*, 12(1):84–101, 1990.

[19] Shay Kutten and Boaz Patt-Shamir. Time-adaptive self stabilization. In *Proc. PODC*, 1997.

[20] Seungjoon Lee, Dave Levin, Vijay Gopalakrishnan, and Bobby Bhattacharjee. Backbone construction in selfish wireless networks. In *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[21] Xin Liu, Guevara Noubir, Ravi Sundaram, and San Tan. Spread: Foiling smart jammers using multi-layer agility. In *Proc. of Infocom '07*, pages 2536–2540, 2007.

[22] Koji Nakano and Stephan Olariu. Randomized leader election protocols in radio networks with no collision detection. In *ISAAC '00: Proceedings of the 11th International Conference on Algorithms and Computation*, pages 362–373, London, UK, 2000. Springer-Verlag.

[23] Vishnu Navda, Aniruddha Bohra, Samrat Ganguly, and Dan Rubenstein. Using channel hopping to increase 802.11 resilience to jamming attacks. In *Proc. of Infocom '07*, pages 2526–2530, 2007.

[24] Koji Nikano and Stephan Olariu. Uniform leader election protocols for radio networks. *IEEE Trans. Parallel Distrib. Syst.*, 13(5):516–526, 2002.

[25] A. Pelc and D. Peleg. Feasibility and complexity of broadcasting with random transmission failures. In *Proc. of PODC '05*, 2005.

[26] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. A jamming-resistant mac protocol for multi-hop wireless networks. In *Proc. DISC*, 2010.

[27] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive and fair medium access despite reactive jamming. In *Proc. 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[28] Georgios Smaragdakis, Ibrahim Matta, and Azer Bestavros. Sep: A stable election protocol for clustered heterogeneous wireless sensor networks. In *Proc. 2nd International Workshop on Sensor and Actor Network Protocols and Applications (SANPA)*, 2004.

[29] Dan E Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM J. Comput.*, 15(2):468–477, 1986.

[30] A.D. Wood, J.A. Stankovic, and G. Zhou. DEEJAM: Defeating energy-efficient jamming in IEEE 802.15.4-based wireless networks. In *Proc. of SECON '07*, 2007.

[31] W. Xu, T. Wood, and Y. Zhang. Channel surfing and spatial retreats: defenses against wireless denial of service. In *Proc. of Workshop on Wireless Security*, 2004.