AN OPTIMAL SELF-STABILIZING FIRING SQUAD*

DANNY DOLEV[†§], EZRA N. HOCH[†], AND YORAM MOSES[‡]

Abstract. Consider a fully connected network where up to t processes may crash, and all processes start in an arbitrary memory state. The self-stabilizing firing squad problem consists of eventually guaranteeing simultaneous response to an external input. This is modeled by requiring that the non-crashed processes "fire" simultaneously if some correct process received an external "GO" input, and that they only fire as a response to some process receiving such an input. This paper presents FIRE-SQUAD, the first self-stabilizing firing squad algorithm.

The FIRE-SQUAD algorithm is optimal in two respects: (a) Once the algorithm is in a safe state, it fires in response to a GO input as fast as any other algorithm does, and (b) Starting from an arbitrary state, it converges to a safe state as fast as any other algorithm does.

Key words. Distributed algorithms, Firing squad, Self-stabilization, Synchronous system, Common knowledge, Simultaneity

AMS subject classifications. 68M14

1. Introduction. The firing squad problem was first introduced in [2, 3]. Informally, in the context of distributed computing, it is assumed that at any given round a process may receive an external "GO" input, which is considered a request for the correct processes to simultaneously "fire." Roughly, a good solution is a protocol satisfying three properties: (a) if some process fires in round r then all the non-crashed processes fire simultaneously in round r; (b) if a correct process receives a GO input in round r' then it will fire at some later round $r > r'$; and (c) a process fires in round r only if some process received a GO input in some round $r' < r$. (The formal definition, presented later, disallows a solution in which a single GO input induces multiple firing events.)

Requiring the processes to fire simultaneously captures an important aspect of distributed systems: There are cases in which it is important that activities begin in the same round, *e.g.,* when one distributed algorithm ends and another one begins, and the two may interfere with each other if their executions overlap. Similarly, many synchronous algorithms are designed assuming that all sites start participating in the same round of communication. Finally, simultaneity may be motivated by the fact that a distributed system interacts with the outside world, and these interactions should often be simultaneously consistent. A non-simultaneous announcement to financial (stock) markets may enable unfair arbitrage trading, for example.

This paper focuses on solving the firing squad problem in a self-stabilizing manner. Conceptually, the model is one in which the system may be unstable and undergo fairly arbitrary changes for an unknown (arbitrarily long) finite period. These changes are thought of as *transient* errors. From some point, however, transient errors end and all processes start behaving accroding to their protocols. A self stabilizing algorithm

*A shorter version of this work appeared in "The 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems" (SSS 2009). This is the author's copy of the paper - the paper will be published in SIAM J. on Computing.

†The Hebrew University of Jerusalem, Jerusalem, Israel

‡Technion—Israel Institute of Technology, Haifa, Israel

§This research was supported in part by the Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), by the ISG (Israeli Smart Grid) Consortium, administered by the office of the Chief Scientist of the Israeli ministry of Industry and Trade and Labor, and by the Google Inter-university center for Electronic Markets and Auctions".

is one in which the system's behavior is guaranteed to conform to the intended specification, within finite time from when the transient errors end. Formally, self stabilizing algorithms are modelled as starting out at an arbitrary initial state (corresponding to the result of arbitrary transient failures), and following the protocol with the goal of converging into correct behavior in finite time. Self stabilization is a broad subfield of fault-tolerant distributed systems. See [6] for the standard reference to the field.

Coordinating simultaneous actions is not subsumed by the consensus task. Indeed, even when no transient failures are considered possible (so there is a global clock and no self-stabilization is required), solving the firing squad problem or simultaneously deciding in a consensus task can be considerably harder than plain consensus [4, 8]. This implies, in particular, that clock synchronization [5, 11, 7, 12, 18] does not suffice for solving the firing squad problem in a self-stabilizing manner; as it can be seen as providing round-numbers to a self-stabilizing environment, which still leaves the firing squad problem as a non-trivial problem.

The firing squad problem is a primary example of a problem requiring simultaneously coordinated actions by the non-faulty processes. Simultaneous coordination has been shown to be closely related to the notion of common knowledge [10, 9], and this connection has been used to characterize the earliest time required to reach simultaneous consensus, firing squad, and related problems in a variety of failure models [8, 16, 1, 17, 13, 15]. One of the insights obtained from this literature is the fact that the time at which a simultaneous action that is based on initial values or external inputs can be performed depends in a crucial way on the pattern in which failures occur.

A general fault-tolerant service supporting simultaneous agreement, called *continuous consensus*, was defined in [13]. In this setting, each of the processes maintains a list of events of interest that have taken place in the run, and the lists at all non-faulty processes are guaranteed to be identical at all times. The authors present an optimal (non-stabilizing) implementation of such a service, which is a protocol called ConCon. If we define the events monitored by ConCon to be of the form (GO, p, k), corresponding to a GO message arriving at process p at the end of round k, then a firing squad protocol can be obtained from ConCon simply by having the non-faulty processes fire exactly when a (GO, p, k) event first appears in their identical copies of the "common" list. We shall refer by CCfs to the solution of the firing squad problem that is based on ConCon.

Traditionally, the firing squad problem assumes that processes do not recover, *i.e.,* failed processes stay failed forever. Moreover, even though it is easy to extend the firing squad problem so that it can be repeatedly executed (*i.e.,* allow for multiple firings over time, given that multiple GO inputs are received), it assumes that nothing in the system goes amiss—except possibly for the crash failures being accounted for. Adding support for handling transient faults increases the robustness of a firing squad algorithm in this aspect. Indeed, a self stabilizing solution will, in particular, be able to cope with process recovery: Following process recoveries, the system will eventually converge to a valid state and continue operating correctly.

Transient faults alter a process's memory state in an arbitrary way. A self-stabilizing algorithm [6] is assumed to start in an arbitrary state and be guaranteed to eventually reach a state from which it operates according to its intended specification. Starting the operation at an arbitrary state enables the adversary to "plant" false information, such as the receipt of GO messages in the past, which can cause the algorithm to unjustifiably fire, either immediately, or within a few rounds. One of the challenges in designing an efficient self-stabilizing firing squad algorithm is in bounding the damage that can be caused by such false information in the initial state.

Perhaps the first candidate solution would be to initiate an instance of CCFS in every round, with $t+1$ instances executing concurrently at any given time, where t is an upper bound on the number of possible crashed processes. Firing would then take place if it is dictated by any of the instances. Since the component instances of such a solution are not themselves stabilizing, all we can show is that such a solution is guaranteed to stabilize after $t+1$ rounds, regardless of the failure pattern. We shall present a solution that does not consist of such a concurrent composition. Moreover, it performs subtle consistency checks to restrict the impact of false information that appears in the initial state. As a result, in some cases we obtain stabilization in as little as two rounds.

The above discussion points out the stabilization time as an important aspect of a self-stabilizing firing squad algorithm. Another central performance parameter is its swiftness: Once the algorithm has stabilized, how fast does it fire given that some process receives a GO input? In addition to solving the self-stabilizing firing squad problem, the algorithm presented in this paper is also optimal in terms of both its stabilization time, and its swiftness.

The main contributions of this paper are:

- A self-stabilizing variant of the firing squad problem is defined, and an algorithm solving it in the case of crash failures is presented.
- The proposed algorithm, called FIRE-SQUAD, is shown to be optimal both in terms of the time it requires to stabilize and in terms of the time it takes, after stabilization, to fire in response to a GO input.
- Finally, the optimality is demonstrated in a fairly strong sense: For every possible failure pattern, both stabilization time and swiftness are the fastest possible, in any correct algorithm. In extreme cases this enables stabilization in two rounds and firing in one round.

The rest of the paper is organized as follows. Section 2 describes the model and defines the problem at hand. Section 3 provides lower bounds for the optimality properties. Section 4 describes the proposed solution, FIRE-SQUAD, and proves its correctness and optimality. Finally, Section 6 concludes with a discussion.

2. Model and Problem Definition. The system consists of a set $\mathcal{P} = \{1, \ldots, n\}$ of processes. Communication is done via message passing, and the network is synchronous and fully connected. The system starts out at time[1] $k = 0$, and a communication round r starts at time $k = r - 1$ and ends at time $k = r$. At time k each process computes its state according to its state at time $k - 1$, the internal messages it received by time k (sent by other processes at time $k - 1$) and external inputs (if any) that it received at time k. In addition, at any time $k \geq 0$ a process can produce an external output (such as "firing").

We focus on external inputs in the form of GO messages, and external outputs consisting of firing actions. Let $\mathcal{I}_p^k \in \{0, 1\}$ represent the external input of process p at time k. We say that p received an external GO input at time k if $\mathcal{I}_p^k = 1$; Otherwise, (if $\mathcal{I}_p^k = 0$), we say that p *did not* receive a GO input. Let $\mathcal{I}_p = \{\mathcal{I}_p^k\}_{k=0}^{\infty}$, let $\mathcal{I}^k = \{\mathcal{I}_p^k\}_{p \in \mathcal{P}}$ and let $\mathcal{I} = \{\mathcal{I}_p\}_{p \in \mathcal{P}}$. Thus, $\mathcal{I}$ is "the input pattern", and $\mathcal{I}^k$ is the (joint) input at time k. In a similar manner define $\mathcal{O}_p^k \in \{0, 1\}, \mathcal{O}_p, \mathcal{O}^k$ and $\mathcal{O}$ as

[1] All references to "time" in this paper refer to non-negative integer times.

the output pattern. If $\mathcal{O}_p^k = 1$ we say that p fires at time k, and if $\mathcal{O}_p^k = 0$ we say p does not fire at time k. It will be convenient to say that a fire action occurs at time k if $\mathcal{O}_p^k = 1$ for some process p, and similarly that a GO input is received at time k if $\mathcal{I}_p^k = 1$ for some p.

Denote by t an *a priori* bound on the number of faulty processes in the system. For ease of exposition, we assume that $t < n-1$, so that there are at least two processes that need to coordinate their actions. We assume the *crash* failure model, in which a faulty process p does not send any messages after its failing round; it behaves correctly before its failing round, and sends an arbitrary subset of its intended messages during its failing round.

A failure pattern describes for each time k which processes have failed by time k, and for each of these processes that fails in round k (*i.e.,* did not fail by time $k-1$), the pattern determines which of its outgoing communication channels are blocked (and hence do not deliver its messages) in round k. Notice that a process may fail in round k even if all of its messages are delivered. We denote a failure pattern by $\mathcal{F}$, and by $\mathcal{F}^k$ the set of processes that fail in $\mathcal{F}$ by time k. Observe that $\mathcal{F}^k \subseteq \mathcal{F}^{k+1}$; in the crash failure model failed processes do not recover. Similarly, we use $\mathsf{G}^k = \mathcal{P} \setminus \mathcal{F}^k$ to denote the set of processes that are non-faulty at time k. Finally, G will denote the set of processes that remain non-faulty throughout $\mathcal{F}$, *i.e.,* $\mathsf{G} = \bigcap_{k=0}^{\infty} \mathsf{G}^k$. Notice that the set G is always defined in terms of a failure pattern $\mathcal{F}$, which is typically clear from the context.

In addition to crashes, there are also transient faults. Formally, we denote by $\mathcal{S}_p^k$ the state of a process p at time k. We denote by $\mathcal{S}^k = (\mathcal{S}_1^k, \ldots, \mathcal{S}_p^k, \ldots, \mathcal{S}_n^k)$ the *global* state of the entire system at time k. Transient faults are captured by the assumption that the system may start from any (arbitrary) state, and there is some round r such that for all rounds $r' \geq r$ the intended algorithm operates as written. In other words, for any possible state S, if $\mathcal{S}^0 = S$ then eventually (starting from some round r) the algorithm operates correctly, as defined below.

For the following analysis, each algorithm $\mathcal{A}$ is assumed to have an initial state $\mathcal{S}_{init}^{\mathcal{A}}$. For self-stabilizing algorithms, we fix an arbitrary state as $\mathcal{S}_{init}^{\mathcal{A}}$ (as the algorithm should converge starting from any initial state). The *a priori* bound of t on the number of failures is assumed to be hard-wired into the algorithm, and is not affected by transient faults. Such an algorithm is assumed to be executed only in the context of failure patterns in which at most t processes crash. For such failure patterns $\mathcal{F}$, the algorithm $\mathcal{A}$ produces an output pattern $\mathcal{O}$ starting from state $\mathcal{S}$ given an input $\mathcal{I}$; we denote this output pattern by $\mathcal{O} = \mathcal{A}(\mathcal{S}, \mathcal{I}, \mathcal{F})$.

Informally, the Firing Squad problem requires that: (1) all processes fire together (*"simultaneity"*); (2) if a GO input is received then a fire action occurs (*"liveness"*); and (3) the number of fire actions is not larger than the number of received GO inputs (*"safety"*). Formally,

DEFINITION 2.1. *Let $\mathcal{O} = \mathcal{A}(\mathcal{S}, \mathcal{I}, \mathcal{F})$ and let G denote the set of processes that remain non-faulty throughout $\mathcal{F}$. We say that $\mathcal{O}$ satisfies the $\mathsf{FS}(k)$ properties (capturing correct firing-squad behavior from time k on) w.r.t. $\mathcal{I}$, $\mathcal{F}$, and $\mathcal{O}$, if the following conditions hold for all $k' \geq k$:*

1. *(simultaneity) If $\mathcal{O}_p^{k'} = 1$ for some $p \in \mathcal{P}$ then $\mathcal{O}_q^{k'} = 1$ for all $q \in \mathsf{G}$;*
2. *(liveness) If $\mathcal{I}_p^{k'} = 1$ for some $p \in \mathsf{G}$, then there is $k'' > k'$ s.t. $\mathcal{O}_p^{k''} = 1$;*
3. *(safety) The number of times k'' satisfying $k \leq k'' \leq k'$ at which a fire action occurs at k'' is not larger than the number of times h in the range $0 \leq h < k'$ at which GO inputs are received.*

We can use the $\texttt{FS}(k)$ properties to define when an algorithm solves the firing squad problem in a self stabilizing manner. We first use it to define the stabilization time of an algorithm as follows:

DEFINITION 2.2 (Stabilization time). *The* stabilization time *of $\mathcal{A}$ on $\mathcal{S}$, $\mathcal{I}$ and $\mathcal{F}$, denoted by $\texttt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})$, is the minimal $k \geq 0$ such that $\texttt{FS}(k)$ holds with respect to $\mathcal{I}$, $\mathcal{F}$, and $\mathcal{O} = \mathcal{A}(\mathcal{S}, \mathcal{I}, \mathcal{F})$. (If $\texttt{FS}(k)$ holds for no finite k, then $\texttt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F}) = \infty$.)*

Notice that the *"safety"* property in $\texttt{FS}(k)$ relates outputs starting from time k to inputs starting from time 0. Here's why: Since we consider time 0 to be the point at which transient errors end, if the system starts in a state in which "it appears as if" GO inputs were received before time 0, the good processes may fire after time 0 without a GO message actually having been received. Once all firings induced by such "phantom" GO inputs have occurred, we can legitimately require firing events to happen only in response to genuine GO message receipts. We thus think of the stabilization time, at which in particular the safety property of $\texttt{FS}(k)$ holds, as one after which no firing will occur in response to phantom GO messages. Rather, every firing will be justifiable as a response to some GO message received at or after time 0.

DEFINITION 2.3 (SSFS Algorithm). *An algorithm $\mathcal{A}$ solves the Self stabilizing Firing Squad problem ($\mathcal{A}$ is an SSFS algorithm, for short) if there exists a $k < \infty$ such that $\texttt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F}) \leq k$ for every system state $\mathcal{S}$, input pattern $\mathcal{I}$ and failure pattern $\mathcal{F}$.*

Observe that in a setting with no transient faults, an algorithm $\mathcal{A}$ solves the (non-self-stabilizing) Firing Squad problem if it satisfies $\texttt{FS}(0)$ with respect to $\mathcal{I}$, $\mathcal{F}$, and $\mathcal{O}$, for every $\mathcal{I}$, $\mathcal{F}$ and $\mathcal{O} = \mathcal{A}(\mathcal{S}^{\mathcal{A}}_{init}, \mathcal{I}, \mathcal{F})$.

Notice that Definition 2.3 implies that any SSFS algorithm $\mathcal{A}$ has at least one global state from which the firing squad properties are guaranteed to hold. Denote one of these global states by $\mathcal{S}^{\mathcal{A}}_{stab}$, or simply $\mathcal{S}_{stab}$ when $\mathcal{A}$ is clear from the context.

2.1. Optimality Measures. In this work we are interested in finding an optimal SSFS algorithm. We start by defining stabilization time optimality, which measures how quickly algorithm $\mathcal{A}$ stabilizes.

DEFINITION 2.4. *An SSFS algorithm $\mathcal{A}$ is said to* optimally stabilize *if the following holds for every SSFS algorithm $\mathcal{B}$ and every failure pattern $\mathcal{F}$:*

$$\max_{\mathcal{S},\mathcal{I}}\{\texttt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} \quad \leq \quad \max_{\mathcal{S},\mathcal{I}}\{\texttt{stab}(\mathcal{B}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} .$$

Definition 2.4 defines optimality of an algorithm $\mathcal{A}$ with respect to its stabilization time, *i.e.,* how quickly $\mathcal{A}$ stabilizes according to all of the $\texttt{FS}$ requirements. The intuition behind defining optimality in terms of worst-case $\mathcal{S}$ and $\mathcal{I}$ is to avoid algorithms that are "specific" to an initial global state or input pattern. Thus, by requiring optimality in the worst-case we ensure that the algorithm cannot be hand-tailored to a specific setting, but rather needs to solve the SSFS problem in a "generic" manner.

We now turn to the issue of comparing the responsiveness of distinct firing squad algorithms. Specifically, we are concerned with how quickly an algorithm fires after a GO message is received (once the algorithm has stabilized). For simplicity, we consider receipts of GO by non-faulty processes, since the problem specification forces a firing following such a receipt. Another subtle issue is that if GO messages are received in different rounds between which there is no firing, then it may be difficult to figure out which GO message the next firing is responding to. Again for simplicity, we will be interested in what will be called *sequential* input patterns, in which a GO is not received before all previous GO's have been followed by firings. More formally, we define:

DEFINITION 2.5 (Sequential inputs). *Let $\mathcal{A}$ be an SSFS algorithm. We say that the input $\mathcal{I}$ is sequential with respect to ($\mathcal{A}$, $\mathcal{S}$, $\mathcal{F}$) if (i) no* GO *inputs are received according to $\mathcal{I}$ at times $k < \mathtt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})$, (ii)* GO *inputs are received in $\mathcal{I}$ only by processes from G, and (iii) if $k_1 < k_2$ and* GO *inputs are received at both k_1 and k_2, then there is an intermediate time $k_1 < k' \leq k_2$ at which a fire action occurs.*

The following definition formally captures the number of firing events that occur between the stabilization time and a given time k.

DEFINITION 2.6. *Let $\mathcal{A}$ be an SSFS algorithm and let $\mathcal{O} = \mathcal{A}(\mathcal{S}, \mathcal{I}, \mathcal{F})$. We define $\#[(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F}), k]$ to be the number of rounds k' in the range $\mathtt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F}) \leq k' \leq k$ such that $\mathcal{O}_p^{k'} = 1$ holds for some process p (*i.e., *a firing occurs at time k').*

By definition, if $k < \mathtt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})$ then $\#[(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F}), k] = 0$. With the last two definitions, we are now able to formally compare the responsiveness of different SSFS algorithms:

DEFINITION 2.7 (Swiftness). *Let $\mathcal{A}$ and $\mathcal{B}$ be SSFS algorithms. We say that $\mathcal{A}$ is* at least as swift as *$\mathcal{B}$ if $\mathcal{A}$ fires at least as quickly as $\mathcal{B}$ on all sequential inputs. Formally, we require that for every failure pattern $\mathcal{F}$, input $\mathcal{I}$, and states $\mathcal{S}_{\mathcal{A}}$ of $\mathcal{A}$ and $\mathcal{S}_{\mathcal{B}}$ of $\mathcal{B}$, the following holds. If $\mathcal{I}$ is sequential both with respect to ($\mathcal{A}$, $\mathcal{S}_{\mathcal{A}}$, $\mathcal{F}$) and with respect to ($\mathcal{B}$, $\mathcal{S}_{\mathcal{B}}$, $\mathcal{F}$), then $\#[(\mathcal{A}, \mathcal{S}_{\mathcal{A}}, \mathcal{I}, \mathcal{F}), k] \geq \#[(\mathcal{B}, \mathcal{S}_{\mathcal{B}}, \mathcal{I}, \mathcal{F}), k]$ holds for every time k. An SSFS algorithm $\mathcal{A}$ is optimally swift if it is at least as swift as $\mathcal{B}$ for every SSFS algorithm $\mathcal{B}$.*

We are now in a position to state the main result of the paper: The FIRE-SQUAD algorithm of Figure 4.1 is an SSFS algorithm (Theorem 5.16), it optimally stabilizes (Theorem 5.17) and is optimally swift (Theorem 5.18).

3. Lower Bounds. In this section we provide lower bounds for the stabilization time and for the swiftness of any SSFS algorithm $\mathcal{A}$. The lower bounds build upon previous results in the field of simultaneous agreement.

Recall that if $\mathcal{A}$ is a non-self-stabilizing Firing Squad algorithm, then $\mathtt{stab}(\mathcal{A}, \mathcal{S}_{init}^{\mathcal{A}}, \mathcal{I}, \mathcal{F}) = 0$ for all $\mathcal{I}$ and $\mathcal{F}$. Therefore, in the non-self-stabilizing case, it only makes sense to compare algorithms in terms of their swiftness. In a non-self-stabilizing setting, the firing squad protocol CCFS (based on CONCON [13]) is optimally swift. We will use it as a benchmark and yardstick for expressing and analyzing the performance of self-stabilizing firing squad protocols. To compare the performance of different algorithms, we make use of the following definitions.

DEFINITION 3.1. *We denote by $\delta(\mathcal{F}, k)$ the number of processes known at time k to be faulty by the processes in G^k in a run of* CCFS *with failure pattern $\mathcal{F}$.* Intuitively, $\delta(F, k)$ stands for the number of failures that are *discovered* by time k in a run with pattern $\mathcal{F}$. We remark that $\delta(\mathcal{F}, k)$ is well-defined, because the same number of faulty processes are discovered (at the same times) in all runs of CCFS that have failure pattern $\mathcal{F}$. Moreover, since CCFS detects failures as a full-information protocol does, no algorithm $\mathcal{A}$ can discover more failed processes than CCFS does (see [8]). Thus, $\delta(\mathcal{F}, k)$ is an upper bound on the number of failed process discovered by time k by any algorithm $\mathcal{A}$ in an execution in which the failure pattern is $\mathcal{F}$.

CCFS makes essential use of a notion of *horizon*, which is roughly the time by which past events are guaranteed to become common knowledge. This motivates the following definitions.

DEFINITION 3.2 (Horizons). *Given a failure pattern $\mathcal{F}$, the* horizon distance *at time k, denoted by $\mathtt{H\_dis}(\mathcal{F}, k)$, is $t + 1 - \delta(\mathcal{F}, k)$. The* absolute horizon *at time k,* *denoted* $\mathtt{absH}(\mathcal{F}, k)$*, is* $k + \mathtt{H\_dis}(\mathcal{F}, k)$.

While the absolute horizon is an upper bound on when events become common knowledge, the publication time is a lower bound on this time. It is defined as follows:

Definition 3.3 (Publication Time). *Given a failure pattern* $\mathcal{F}$*, the* publication time *for (time)* k*, denoted by* $\pi(\mathcal{F}, k)$*, is* $\min_{k' \geq k}\{\mathtt{absH}(\mathcal{F}, k')\}$.

When $\mathcal{F}$ is clear from the context, it will be omitted, and we write simply $\delta(k)$, $\mathtt{H\_dis}(k)$, $\mathtt{absH}(k)$ and $\pi(h)$.

As shown in [13], for a given failure pattern $\mathcal{F}$, a GO input received at time k is "common knowledge" not before time $\pi(\mathcal{F}, k)$. Thus, for a specific algorithm $\mathcal{A}$, the publication time for 0 bounds (from below) the time k at which the first firing action can occur in $\mathcal{O} = \mathcal{A}(\mathcal{S}_{stab}, \mathcal{I}, \mathcal{F})$.

The publication time $\pi(\mathcal{F}, k)$ is a generalization of notions developed in [8] for Simultaneous (single-shot, non-stabilizing) Consensus. In that paper, a notion of the *waste* of $\mathcal{F}$ is defined, and information about initial values—which can be viewed in our setting as being about external inputs at time 0—becomes common knowledge at time $t + 1 - \mathit{waste}$. In our terminology, this occurs precisely at the publication time $\pi(\mathcal{F}, 0)$ for events of time 0.

The intuition behind the first lower bound is that if CCFS receives a GO input at time 0, then it fires at time $\pi(0)$ (Lemma 3.4). Since CCFS is optimal, an SSFS algorithm $\mathcal{A}$ cannot fire faster. Therefore, if we consider $\mathcal{A}$ starting in a global state where $\mathcal{A}$ "thinks" it received a GO input 1 round ago, then $\mathcal{A}$ will not fire before time $\pi(0) - 1$. The formal proof appears in the proof of Theorem 3.6.

Lemma 3.4. *Let* $\mathcal{F}$ *be any failure pattern and let* $\mathcal{I}$ *be an input pattern for which* $\mathcal{I}_q^k = 0$ *for every process* q *and time* $k \geq 0$*, except for one process* $p \in \mathsf{G}$ *for which* $\mathcal{I}_p^0 = 1$*. The first fire action of* $\mathcal{O} = \mathrm{CCFS}(\mathcal{S}_{init}^{\mathrm{CCFS}}, \mathcal{I}, \mathcal{F})$ *occurs at time* $\pi(\mathcal{F}, 0)$.

Proof. This follows directly from the analysis in [13]. □

Notation 1. *For input* $\mathcal{I}$ *and an integer* $i \geq 0$*, denote by* $\mathcal{I}(i^{\rightarrow})$ *the input pattern that is obtained by excluding the first* i *rounds of* $\mathcal{I}$*. Formally,* $\mathcal{I}(i^{\rightarrow})^k = \mathcal{I}^{k+i}$ *for all* $k \geq 0$*. Similarly denote* $\mathcal{F}(i^{\rightarrow})$ *(w.r.t.* $\mathcal{F}$*).*

Lemma 3.5. *Let* $\mathcal{F}$ *be a failure pattern. Let* $\mathcal{F}'$ *be a failure pattern with no faults at time* $k = 0$ *and* $\mathcal{F}'(1^{\rightarrow}) = \mathcal{F}$*. Then* $\pi(\mathcal{F}', 0) \geq \pi(\mathcal{F}, 0)$.

Proof. For every time k we have that $\delta(\mathcal{F}', k) \leq \delta(\mathcal{F}, k)$. Therefore, $\mathtt{absH}(\mathcal{F}', k) \geq \mathtt{absH}(\mathcal{F}, k)$ holds for all $k \geq 0$. Thus, $\min_{k \geq 0}\{\mathtt{absH}(\mathcal{F}', k)\} \geq \min_{k \geq 0}\{\mathtt{absH}(\mathcal{F}, k)\}$, *i.e.,* $\pi(\mathcal{F}', 0) \geq \pi(\mathcal{F}, 0)$. □

We can now prove our first lower bound result, stating that the worst case stabilization time of every SSFS algorithm $\mathcal{A}$ is at least $\pi(0)$.

Theorem 3.6. $\max_{\mathcal{S},\mathcal{I}}\{\mathtt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} \geq \pi(\mathcal{F}, 0)$ *holds for every SSFS algorithm* $\mathcal{A}$ *and every failure pattern* $\mathcal{F}$.

Proof. To prove this theorem, we find a state $\mathcal{S}$ and input $\mathcal{I}$ such that $\mathtt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F}) \geq \pi(\mathcal{F}, 0)$. Since $\mathcal{A}$ solves the SSFS problem there is a global state $\mathcal{S}_{stab}$ from which all of the $\mathtt{FS}$ properties hold.

Let $p \in \mathsf{G}$ be a process that is non-faulty throughout $\mathcal{F}$, and consider the following input pattern $\hat{\mathcal{I}}$, where $\hat{\mathcal{I}}_q^k = 0$ for all q, k, except that $\hat{\mathcal{I}}_p^0 = 1$. Consider $\hat{\mathcal{F}}$ to be a failure pattern with no failures at time $k = 0$ (*i.e.,* $\hat{\mathcal{F}}^0 = \emptyset$) and $\hat{\mathcal{F}}(1^{\rightarrow}) = \mathcal{F}$ for the rest. Due to *"liveness"*, $\mathcal{A}$'s run from $\mathcal{S}_{stab}$ with input $\hat{\mathcal{I}}$ and failures $\hat{\mathcal{F}}$ will eventually fire; denote the firing time as k (*i.e.,* $\mathcal{O}_p^k = 1$ for some process p).

By Lemma 3.4, $\pi(\hat{\mathcal{F}}, 0)$ is the optimal time for simultaneous firing, and since starting from $\mathcal{S}_{stab}$ all properties hold, including *"simultaneity"*, it holds that $k \geq \pi(\hat{\mathcal{F}}, 0)$.

Consider global state $\mathcal{S}$ of $\mathcal{A}$ after executing a single round with $\hat{\mathcal{I}}$ as input and $\hat{\mathcal{F}}$ as failure pattern and $\mathcal{S}_{init}$ as starting global state. Consider the run of $\mathcal{A}$ from $\mathcal{S}$ with input $\mathcal{I}$ and failure pattern $\mathcal{F}$. $\mathcal{A}$ must fire at time $k-1$, as it cannot distinguish the run from $\mathcal{S}_{init}, \hat{\mathcal{I}}, \hat{\mathcal{F}}$ and from $\mathcal{S}, \mathcal{I}, \mathcal{F}$. By Lemma 3.5, $\pi(\hat{\mathcal{F}}, 0) \geq \pi(\mathcal{F}, 0)$, and therefore $\mathcal{A}$ will not fire before time $k-1 \geq \pi(\hat{\mathcal{F}}, 0) - 1 \geq \pi(\mathcal{F}, 0) - 1$. However, notice that $\mathcal{I}$ contains only "0" inputs, implying that *"safety"* does not hold for $\mathcal{A}$ when starting from $\mathcal{S}$ with input $\mathcal{I}$ and failure $\mathcal{F}$ for the first $\pi(\mathcal{F}, 0) - 1$ rounds. *I.e., "safety"* can hold starting from time $\pi(\mathcal{F}, 0)$ and on. Therefore, $\max_{\mathcal{S},\mathcal{I}}\{\mathtt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} \geq \pi(\mathcal{F}, 0)$. $\square$

Our second lower bound result, informally stating that no SSFS algorithm can fire faster than CCFS, is captured by the following theorem. (Notice that the claim is made with respect to sequential input patterns.)

THEOREM 3.7. *Let $\mathcal{A}$ be an SSFS algorithm, $\mathcal{I}$ a sequential input, $\mathcal{F}$ a failure pattern and $\mathcal{O} = \mathcal{A}(\mathcal{S}_{stab}, \mathcal{I}, \mathcal{F})$. For every $k \geq 0$ for which a* GO *input is received in $\mathcal{I}^k$ there is no fire action in $\mathcal{O}$ during times k' satisfying $k < k' < \pi(\mathcal{F}, k)$.*

Proof. Let $\mathcal{A}$, $\mathcal{I}$, $\mathcal{F}$ and $\mathcal{O}$ be as assumed in the claim. Suppose, by way of contradiction, that there are k and k' such that $k < k' < \pi(\mathcal{F}, k)$, a GO message is received in $\mathcal{I}^k$, and a fire action takes place in $\mathcal{O}^{k'}$. Assume, without loss of generality, that no fire action occurs in $\mathcal{O}(k'')$ for earlier times $k < k'' < k'$. Denote by $\mathcal{S}^k$ the global state of $\mathcal{A}$ at time k.

Since $\mathcal{A}$ started to run from $\mathcal{S}_{stab}$, $\mathtt{FS}(0)$ holds with respect to $\mathcal{I}, \mathcal{F}$ and $\mathcal{O}$. Since $\mathcal{I}$ is sequential, and k' is the minimal time for which $\mathcal{O}$ has a fire action after time k, we have that $\mathcal{I}(k^{\rightarrow})$ contains a GO input at time 0 and does not contain a GO input until time k'. Therefore, $\mathcal{O} = \mathcal{A}(\mathcal{S}^k, \mathcal{I}(k^{\rightarrow}), \mathcal{F}(k^{\rightarrow}))$ will have its first fire action at time $k' - k$.

From the optimality of CCFS and Lemma 3.4, we have that $\mathcal{A}$ cannot fire before time $\pi(\mathcal{F}(k^{\rightarrow}), 0)$. Thus $k' - k \geq \pi(\mathcal{F}(k^{\rightarrow}), 0)$, which implies that $k' \geq k + \pi(\mathcal{F}(k^{\rightarrow}), 0)$. By definition of π and $\mathcal{F}(k^{\rightarrow})$ we have that $\pi(\mathcal{F}, k) \leq k + \pi(\mathcal{F}(k^{\rightarrow}), 0)$, contradicting the assumption that $k < k' < \pi(\mathcal{F}, k)$. $\square$

4. Solving SSFS. The algorithm FIRE-SQUAD in Figure 4.1 is an SSFS algorithm that both optimally stabilizes and is optimally swift. For swiftness, the algorithm is based on the approach used in the CCFS algorithm, in which the horizon is computed by monitoring the number of failures that occur, and a firing action takes place when the receipt of a GO becomes common knowledge. The horizon computation at a process p makes use of reports that p receives from other processes regarding failures that they have observed. Following a transient fault, the state of a process may contain arbitrary (including false) information about failures. In the crash failure model, a process q will learn about (truly) crashed processes in the first round. Consequently, p will compute a correct horizon one round later, once it receives reports from all such processes q. Roughly speaking, this can be used as a basis for a (nontrivial) solution that stabilizes within two rounds of the optimal time.

In order to improve on the above and obtain an optimal algorithm, FIRE-SQUAD employs a couple of subtle consistency checks. The first one (termed "consistency check I" in Figure 4.1) involves checking the information obtained from other processes regarding failures they observed before the current round started. In the crash failure model, every failure observed by q before time $k-1$ must be directly observable by p no later than time k. So if the set of failures reported to p contains failures that p has not directly observed, then it must be time $k \leq 1$, and in computing the horizon p will use the set of failures that it has directly observed instead of the set

Algorithm FIRE-SQUAD (t)

do forever: /* *executed on process p at time k* */
/* *process p is unaware of the value of k* */

  receive all available ($Requests_q$, $Failed_q$, $Views_q$) messages from process $q \in \mathcal{P}$;

  /* *update variables according to messages of round k and external input* */
  set $Requests[0] := \mathcal{I}_p^k$;
  for $1 \leq i \leq t+1$: **set** $Requests[\mathrm{i}] := \max_q\{Requests_q[i-1]\}$;
  set $\widehat{Failed} := \bigcup_q Failed_q$;
  set $Failed := \{q :\ p$ did not hear from q in this round$\}$;
  for $1 \leq i \leq t$: **set** $Views[i-1] := \min_q\{Views_q[i]\} + 1$;

  /* *calculate horizon at time $k-1$* */
  set $Horizon := t + 1 - \min\{|\widehat{Failed}|, |Failed|\}$; /* *consistency check I* */
  set $Views[Horizon\text{-}1] := 1$;
  for $0 \leq i \leq t$: **set** $Views[i] := \max\{Views[i], Horizon - i\}$; /* *check II* */

  /* *should we fire?* */
  if for some $i' \geq Views[0]$ it holds that $Requests[i'] = 1$ then
    for $i' \leq i'' \leq t+1$: **set** $Requests[i''] := 0$;
    do "Fire";
  fi;

  /* *send round $k+1$ messages to all processes* */
  send ($Requests$, $Failed$, $Views$) to all;
od.

Clean up:
$Requests$ contains only $\{0, 1\}$ values. $Views$ contains only values $\in \{0, \ldots, t+1\}$.

FIG. 4.1. FIRE-SQUAD*: a self-stabilizing firing squad algorithm.*

of reported failures. A subtle proof shows that, in this case, the computed horizon works correctly if $k = 1$, which is crucial for the algorithm's stabilization optimality (Lemma 5.5). The second consistency check (termed "check II") is based on the fact that in normal operation the horizon distance is (weakly) monotone decreasing. The local state contains information about previous horizon computations, and our second consistency check forces it to satisfy weak monotonicity.

We now turn to describe the details of FIRE-SQUAD. The following discussion and lemmas are stated w.r.t. the algorithm and its components. For a variable *var*, we denote by var_p^k the value of *var* at process p after the computation step at time k.

Each process p has a vector $Requests_p[i]$ that represents p's information about a GO input received by some process i time units ago, which was not as yet followed by a firing action. More precisely, if $Requests_p^k[i] = 1$, then some process received a GO input at time $k - i$, and no firing action occurred between time $k - i + 1$ and time k. The vector *Requests* contains values for values of $0 \leq 0 \leq t+1$ referring to the previous $t + 1$ time units and the current time; a total of $t + 2$ entries.

In addition, each process has a set *Failed*, which consists of the processes it has seen to be failed in the current round. That is, at time k, process p's $Failed_p^k$ set contains all processes from which process p did not received messages during round k (*i.e.,* messages sent at time $k - 1$). $\widehat{Failed}$ is the union of all $Failed_q$ sets (as received from other processes) of the previous round. That is, at time k, $\widehat{Failed}_p^k$ is the union of $Failed_q^{k-1}$ as computed at time $k - 1$ by every process q from which p received messages in round k.

Finally, each process keeps track of a vector *Views*. If $Views_p^k[i] = z$ it means that at time $k+i$, data from time $k-z$ is common knowledge. The vector *Views* contains $t+1$ entries, $i = 0, \ldots, t$, for the current round and the coming t rounds.

For uniformity and ease of exposition every process p is assumed to send messages to itself in every round. Moreover, due to the nature of transient faults, a process executing the algorithm is unaware of the current round number. However, for ease of exposition in describing and analyzing the algorithm, we refer to such rounds using numbers k etc.

5. Correctness Proof. A central notion in the analysis of simultaneous actions under crash failures is that of a *clean round* [8]. We say that p fails *silently* in round k if it is not blocked according to $\mathcal{F}$ from sending messages in round k to any of the processes $q \in \mathsf{G}^k$. Thus, no process surviving round k can detect p's failure in this round.

DEFINITION 5.1 (Clean Round). *Round r in failure pattern $\mathcal{F}$ is a* clean round *if (i) no process fails silently in round $r-1$, and (ii) all processes (if any) that fail in round r fail silently.* This definition of a round r being clean in $\mathcal{F}$ coincides with the standard definition of clean rounds previously used in non-stabilizing systems [8]. In protocols such as FIRE-SQUAD, in which every process sends the same message to all other processes in every round, all (non-crashed) processes receive the same set of messages in a clean round (see Lemma 5.2).

The main result of the paper is stated in three theorems (Theorem 5.16, Theorem 5.17, Theorem 5.18) and can be summarized as follows: The FIRE-SQUAD algorithm is an SSFS algorithm that optimally stabilizes and is optimally swift.

We start with an outline of the proof, followed by a detailed proof.

5.0.1. Proof outline:.

1. Once a clean round has occurred, different processes agree on the value of *Requests* (Lemma 5.2 and Lemma 5.3);
2. Lemma 5.5 ensures monotonocity of the horizon value based on consistency check I. Lemma 5.4 shows that check I does not affect the computation after time $k = 1$. Similarly, Lemma 5.6 shows that the impact of check II is limited in time, in a manner that ensures later on to establish swiftness optimality.
3. As decision to fire depends solely on the value of *Views*[0] and the *Requests* array (Line `10`), if processes agree on the value of *Views*[0] they are guaranteed to act simultaneously, either firing together or, together, refraining from firing (Lemma 5.7);
4. Lemma 5.8 and Lemma 5.9 show that *Views*[0] is the same at all non-crashed process (once a clean round has occurred);
5. Points 1, 2 and 3 above lead to Lemma 5.10, stating that once a clean round occurs, *"simultaneity"* holds;
6. *"liveness"* holds by Lemma 5.11;
7. Lemma 5.12 and Lemma 5.13 lead to Lemma 5.14, which states that *"safety"* holds starting from round $\pi(0)$. This, according to the lower bounds, is optimal;
8. Lemma 5.15 (together with Lemma 5.13) shows that FIRE-SQUAD fires by time $\pi(k)$ given a GO input at time k. The lower bound in Theorem 3.7 implies that this is optimal;
9. Finally, Theorem 5.16, Theorem 5.17 and Theorem 5.18 show that FIRE-SQUAD is an SSFS algorithm that optimally stabilizes and is optimally swift.

The first lemma shows that different nodes agree on their variables following a clean round.

LEMMA 5.2. *If round r is clean, then the sets Failed^r, $\widehat{\mathit{Failed}}^r$, and the array Views^r are identical for all non-faulty processes.*

Proof. In the FIRE-SQUAD algorithm every process sends its *Failed* set and *Views* array to all other processes in every round. If round r is clean, then all processes receive the same information about the values of *Failed* and *Views* in the system. Thus, the value of *Views* computed on Line 6, which depends on the Views_q values received in the current round, is the same for all $p \in \mathsf{G}$. Similarly, the value of $\widehat{\mathit{Failed}}$ calculated on Line 4, which depends on the Failed_q sets received is the same at all $p \in \mathsf{G}$. Finally, in a clean round, all non-faulty processes receive messages from the same set of processes. As a result, the value of *Failed* (computed on Line 5) is the same for all $p \in \mathsf{G}$. Since changes to *Failed*, $\widehat{\mathit{Failed}}$ and *Views* performed on Line 7–13 depend only on the values of *Failed*, $\widehat{\mathit{Failed}}$ and *Views*, the same changes are performed by all non-faulty processes. The claim follows. □

The previous lemma is focused on the end of a clean round. The following lemma talks about the dth round after a clean round.

LEMMA 5.3. *Let r be a clean round, let $0 \leq d \leq t$ and let $p, p' \in \mathsf{G}^{r+d}$. Then $\mathit{Requests}_p^{r+d}[i] = \mathit{Requests}_{p'}^{r+d}[i]$ holds for all i in the range $d < i \leq t$.*

Proof. We prove the claim by induction on d. The base case is $d = 0$, in which round $r + d = r$ is a clean round, and all non-faulty processes receive the same set of messages. Thus, by Line 3, we have that $\mathit{Requests}_p^r[i] = \mathit{Requests}_{p'}^r[i]$ for all i in the range $d = 0 < i \leq t$. Let $0 < d \leq t$, and assume inductively that the claim holds for $d - 1$. The inductive assumption guarantees that when the $\mathit{Requests}_q$ arrays are sent in round $r + d$ they agree for all i satisfying $d - 1 < i \leq t$. In particular, $\max_q\{\mathit{Requests}_q[i-1]\}$ is the same for all $i > d$. Since $\mathit{Requests}_p[i]$ is set to $\max_q\{\mathit{Requests}_q[i-1]\}$ on Line 3, it follows that $\mathit{Requests}_p^{r+d}[i] = \mathit{Requests}_{p'}^{r+d}[i]$ holds for all $d < i \leq t$, as claimed. □

The purpose of Line 7 is to perform consistency check I, which compares the reported Failed_q values (from the previous round) to failures directly observed by p in the current round (stored in Failed_p). The next lemma shows that this can matter only at times $k \leq 1$. At all times $k \geq 2$, Line 7 can be viewed as having the simpler form of setting the horizon to $t + 1 - |\widehat{\mathit{Failed}}_p|$.

LEMMA 5.4. *$\mathit{Horizon}_p^k = t + 1 - |\widehat{\mathit{Failed}}_p^k|$ holds after Line 7 is executed, for all times $k \geq 2$ and $p \in \mathsf{G}^k$.*

Proof. If $k \geq 2$ then $k - 1 \geq 1$, and so the values of $\widehat{\mathit{Failed}}$ received by p at time k contain only processes that were indeed faulty by the end of round $k-1$. Since failure patterns are monotone, none of these failed processes sends p a message in round k. Hence, by Line 5 we obtain that $\widehat{\mathit{Failed}}_p^k \subseteq \mathit{Failed}_p^k$. □

We denote the first clean round in an execution of FIRE-SQUAD by r_c. By definition, $r_c \geq 1$. One of the properties underlying the use of horizons for ensuring simulatneity has to do with the (downward) monotonicity of the horizon value as time proceeds. Roughly speaking, if the horizon of a given correct process is H at time k, then the horizon of all other processes at later times $k' > k$ will not exceed H. More formally, we can show:

LEMMA 5.5. *If $k \geq 1$ then $\mathit{Horizon}_{p'}^{k+1} \leq \mathit{Horizon}_p^k$ for every $p, p' \in \mathsf{G}^{k+1}$. Moreover, for $k \geq \min\{2, r_c\}$ $\mathit{Horizon}_{p'}^{k+1} \leq \mathit{Horizon}_p^k$ for every $p \in \mathsf{G}^k$ and $p' \in$*

G^{k+1}.

Proof. We start with the second case of the lemma: Let $k \geq \min\{2, r_c\}$ and let $p \in \mathsf{G}^k$ and $p' \in \mathsf{G}^{k+1}$. In particular, either $k \geq 2$, or $k = r_c = 1$. We consider each of these cases separately. Assume that $k \geq 2$, and let $q \in \mathcal{P}$ be a process that updates Failed_q at time $k-1$. According to Line 5, Failed_q contains processes that q does not receive messages from during round $k-1$. All of these processes do in fact fail no later than round $k-1$. Thus, the set $\widehat{\mathit{Failed}}$ computed by process p at time k contains only faulty processes. The set Failed_q at time k contains all processes of Failed_q^{k-1}. Thus, the set $\widehat{\mathit{Failed}}_{p'}$ at time $k+1$ contains all processes from $\widehat{\mathit{Failed}}_p^k$. Hence, $\widehat{\mathit{Failed}}_p^k \subseteq \widehat{\mathit{Failed}}_{p'}^{k+1}$. Therefore, by Lemma 5.4, following Line 7 by p' at time $k+1$ we have that $\mathit{Horizon}_{p'}^{k+1} \leq \mathit{Horizon}_p^k$.

Now consider the case $k = r_c = 1$. Thus, p and p' receive the same set of messages during round 1, and compute Failed and $\widehat{\mathit{Failed}}$ in the same manner. Thus, $\mathit{Horizon}_p^1 = \mathit{Horizon}_{p'}^1$. Moreover, by Line 4 we have that $\widehat{\mathit{Failed}}_{p'}^2 \supseteq \mathit{Failed}_{p'}^1$. It follows that $\min\{|\mathit{Failed}_p^1|, |\widehat{\mathit{Failed}}_p^1|\} = \min\{|\mathit{Failed}_{p'}^1|, |\widehat{\mathit{Failed}}_{p'}^1|\} \leq |\widehat{\mathit{Failed}}_{p'}^2|$. By Lemma 5.4 $\mathit{Horizon}_{p'}^2 := t + 1 - |\widehat{\mathit{Failed}}_{p'}^2|$, hence $\mathit{Horizon}_{p'}^2 \leq t + 1 - \min\{|\mathit{Failed}_p^1|, |\widehat{\mathit{Failed}}_p^1|\} = \mathit{Horizon}_p^1$. That is, we obtain that $\mathit{Horizon}_{p'}^{k+1} \leq \mathit{Horizon}_p^k$.

To finish the proof, we are left to handle the case when $k = 1$ and $p, p' \in \mathsf{G}^2$. Since $p \in \mathsf{G}^2$ by time 2 we have that p' received p's round 2 messages. Implying that $\mathit{Failed}_p^1 \subseteq \widehat{\mathit{Failed}}_{p'}^2$. Moreover, due to the monotonicity of crashes, also $\mathit{Failed}_p^1 \subseteq \mathit{Failed}_{p'}^2$. Therefore, $\min\{|\widehat{\mathit{Failed}}_{p'}^2|, |\mathit{Failed}_{p'}^2|\} \geq |\mathit{Failed}_p^1| \geq \min\{|\widehat{\mathit{Failed}}_p^1|, |\mathit{Failed}_p^1|\}$. Hence, by Line 7 $\mathit{Horizon}_{p'}^2 \leq \mathit{Horizon}_p^1$. □

Denote by $\mathtt{minH}(\mathcal{F}, k)$ the lowest value of $\mathit{Horizon}_p^k$, *i.e.*, $\mathtt{minH}(\mathcal{F}, k) = \min_p\{\mathit{Horizon}_p^k\}$. When $\mathcal{F}$ is clear from the context, we write $\mathtt{minH}(k)$. Notice that $\mathtt{minH}$ is the equivalent of the horizon distance $\mathtt{H\_dis}$ with respect to Fire-Squad (recall that $\mathtt{H\_dis}$ is computed according to CCfs).

The following lemma shows that consistency check II does not affect the computation after time 2, and if the first round is clean, it is not felt after time 1. Formally, we have:

Lemma 5.6. *Let $k \geq \min\{2, r_c\} + 1$ and let $p \in \mathsf{G}^k$. Then for all $0 \leq i < t$, Line 9 does not change the value of $\mathit{Views}_p^k[i]$.*

Proof. Since $k \geq \min\{2, r_c\}+1$ we have that $k-1 \geq \min\{2, r_c\} \geq 1$. At time $k-1$, for every process q and every $0 \leq i \leq t$ it holds that $\mathit{Views}_q^{k-1}[i] \geq \mathit{Horizon}_q^{k-1} - i$, due to Line 9. At time k all processes update Views according to Line 6, thus setting every entry i (for $i \neq t$) to be $\geq \mathtt{minH}(k-1) - i$. Since $k - 1 \geq \min\{2, r_c\}$, we have by Lemma 5.5 that $\max_q\{\mathit{Horizon}_q^k\} \leq \mathtt{minH}(k-1)$. Since for every $i \neq t$, $\mathit{Views}_p[i] \geq \mathtt{minH}(k-1) - i$ it also holds that $\mathit{Views}_p[i] \geq \mathit{Horizon}_p - i$. Hence, $\max\{\mathit{Views}_p[i], \mathit{Horizon}_p - i\} = \mathit{Views}_p[i]$. Thus, Line 9 does not change $\mathit{Views}_p[i]$ for all $i \neq t$. □

Observation 1. *$\delta(k-1) \leq |\mathit{Failed}_p^k| \leq \delta(k)$ holds for every $k \geq 1$. In a similar manner, $\delta(k-1) \leq |\widehat{\mathit{Failed}}_p^{k+1}| \leq \delta(k)$.*

To establish correct behavior with respect to simultaneity, the following lemma shows that for all rounds after the first clean round, if two non-faulty nodes have the same value of $\mathit{Views}[0]$, then they either both fire or both refrain from firing.

Lemma 5.7. *Let $k \geq r_c$ and let $p, p' \in \mathsf{G}^k$. If $\mathit{Views}_p^k[0] = \mathit{Views}_{p'}^k[0]$ then p and*

p' *have the same external output at time* k *(*i.e., *they either both fire or they both do not fire at time* k*).*

Proof. Consider the value of $Views_p^k[0]$. Let $k' \leq k$ be the maximal time at which $Views_p^k[k-k']$ was updated due to Line 8. Notice that $Views_p^{k'}[k-k'] = 1$, and by the update in Line 6 it holds that $Views_p^k[0] \geq k-k'+1$. Moreover, $Horizon_p^{k'} = k-k'+1$, *i.e.,* $t+1-|\widehat{Failed}_p^{k'}| = k-k'+1$.

Between time $k'-1$ and time k there are $k-k'+1$ rounds. From the above discussion, at time $k'-1$ there were at least $t+k'-k$ failed processes. Thus, between time $k'-1$ and time k there was some clean round. Denote this clean round by r.

By Lemma 5.3, for every $i, k-r < i \leq t$, it holds that $Requests_p^k[i] = Requests_{p'}^k[i]$. Since $Views_p^k[0] \geq k-k'+1 \geq k-r+1$, we have that for every i satisfying $Views_p^k[0] \leq i \leq t$ it holds that $Requests_p^k[i] = Requests_{p'}^k[i]$. Thus, p and p' either both pass the condition of Line 10 or they both do not pass. Leading to the fact that either p, p' both fire, or they both do not fire. ◻

To use the above lemma, we need to show that $Views^k[0]$ is the same at all non-faulty nodes. This is done in two stages: First, Lemma 5.8 proves this for the case that $\texttt{minH}(k) = 1$. Second, Lemma 5.9 proves that all non-faulty nodes have the same value of $Views^k[0]$ when $\texttt{minH}(k) > 1$.

LEMMA 5.8. *For every* $k \geq \min\{2, r_c\}$ *and* $p \in \mathsf{G}^k$*, if* $\texttt{minH}(k) = 1$ *then* $Views_p^k[0] = 1$*.*

Proof. If $k = r_c$ then by Lemma 5.2 every process p has $Horizon_p^k = \texttt{minH}(k)$. Therefore, if $\texttt{minH}(k) = 1$ then by Line 8, p sets $Views_p^k[0] = 1$.

Now assume that $k \neq r_c$, and so $k \geq 2$. If $\texttt{minH}(k) = 1$, then some process q has $Horizon_q^k = 1$. Thus q has $|\widehat{Failed}_q^k| = t$. Notice that $\widehat{Failed}_q^k$ contains processes that were faulty during round $k-1$. Therefore, $Failed_q^k = \widehat{Failed}_q^k$, which leads to the conclusion that all $Failed_{q'}^{k-1}$ sets received by q and used in the construction of $\widehat{Failed}_q^k$ were received from non-faulty processes. Thus, all processes receive these sets, and process p also has $|\widehat{Failed}_p^k| = t$, leading to $Horizon_p^k = 1$. Thus, by Line 8, p has $Views_p^k[0] = 1$. ◻

LEMMA 5.9. *Let* $r \geq r_c$ *and* $p, p' \in \mathsf{G}^r$*. If* $\texttt{minH}(r) > 1$ *then* $Views_p^r[i] = Views_{p'}^r[i]$ *holds for all* $0 \leq i < \texttt{minH}(r) - 1$*.*

Proof. The proof is by induction on $r \geq r_c$. For $r = r_c$, we have by Lemma 5.2 that $Views_p = Views_{p'}$, and the claim immediately follows. For the inductive step, assume that $r > r_c$ and that the claim holds for $r-1$. We consider two cases. First assume that $\texttt{minH}(r) = \texttt{minH}(r-1)$. In this case, no process failure is discovered in round r. Thus, round r is clean, and the claim follows by Lemma 5.2 as in the base case.

Next, assume that $\texttt{minH}(r) < \texttt{minH}(r-1)$. The $Views_p[i]$ values can change only on Line 6, Line 8, and Line 9. First consider the change by Line 6. In this case, $Views[i-1]$ is set to $\min_q\{Views_q[i]\} + 1$ for $1 \leq i \leq t$. By the inductive assumption we have that $Views_p[i] = Views_{p'}[i]$ holds for all $0 \leq i < \texttt{minH}(r) - 1$ before Line 6 is applied. Since the values of $Views[j]$ before Line 6 are shifted down by one, and become the values of $Views[j-1]$ after it is applied, we obtain that $Views_p[i] = Views_{p'}[i]$ for all $0 \leq i < \texttt{minH}(r-1) - 2$ once Line 6 has completed. Since $\texttt{minH}(r) < \texttt{minH}(r-1)$, we have that $\texttt{minH}(r) - 1 \leq \texttt{minH}(r-1) - 2$. Consequently,

$Views_p[i] = Views_{p'}[i]$ for all $0 \le i < \texttt{minH}(r) - 1$ when Line 7 is reached.

On Line 8, $Views_p[Horizon^r_p - 1]$ is set to 1. By definition, $\texttt{minH}(r) \le Horizon^r_p$, so the update does not affect values $Views[i]$ for $i < \texttt{minH}(r) - 1$. Hence, the fact that $Views_p[i] = Views_{p'}[i]$ for all $0 \le i < \texttt{minH}(r) - 1$, which was shown above to hold when Line 7 is reached, also holds when Line 9 is reached.

By Lemma 5.6, since $r - 1 \ge r_c$ Line 9 does not change the value of $Views^r_p[i]$, for all $0 \le i < t$. Since $\texttt{minH}(r) - 1 \le t$ we have that after Line 9 $Views_p[i] = Views_{p'}[i]$ for all $0 \le i < \texttt{minH}(r) - 1$. □

The following two lemmas show that *"simultaneity"* and *"liveness"* eventually hold.

LEMMA 5.10. *"simultaneity" holds for all times* $k \ge r_c$.

Proof. By Lemma 5.8 and Lemma 5.9, for two processes p, p' it holds that $Views_p[0] = Views_{p'}[0]$. Together with Lemma 5.7 we have that every pair p and p' of non-crashed processes either both fire or both refrain from firing in round r, for every $r \ge r_c$. □

LEMMA 5.11. *"liveness" holds for all times* $k \ge 0$.

Proof. If some non-faulty process p received a request to fire at time k, then it sets $Requests^k_p[0] = 1$. Line 11 is the only place where $Requests^k_p[0]$ can be altered to 0. However, since $Views^k_p[0] \ge Horizon^k_p \ge 1$, only $Requests^k_p[i]$ for $i \ge 1$ may be set to 0 in Line 11. Thus, at time $k + 1$ it holds that $Requests^{k+1}_p[1] = 1$; and in general, if by time $k+i$ p does not set $Requests^{k+i}_p[i] = 0$ then it holds that $Requests^{k+i+1}_p[i+1] = 1$.

Notice that if p sets $Requests^{k+i}_p[i] = 0$ (for $i \ge 1$), then p executes Line 11, indicating that p fires. Notice that $Views^{k+t+1}_p[0] \le Horizon^{k+t+1}_p \le t + 1$. Thus, if by time $k + t + 1$ p has not set $Requests^{k+t+1}_p[t + 1] = 0$, then at time $k + t + 1$ p will fire. We conclude that within $t + 1$ rounds p will fire, and so *"liveness"* holds. □

We now show another property of the horizon: The algorithm ensures that, intuitively, once the current horizon is reached, current round information is guaranteed to be common knowledge. Formally:

LEMMA 5.12. *Let* $k \ge 1$, *and let* $p \in \mathcal{P}$. *If* k' *is such that* $p \in \mathsf{G}^{k'}$ *and* $k' = k + Horizon^k_p - 1$, *then* $Views^{k'}_p[0] \le Horizon^k_p$.

Proof. Let p be any process and consider time k: by Line 8 process p sets $Views^k_p[Horizon^k_p - 1] = 1$. For $Horizon^k_p = 1$, it holds that $Views^k_p[Horizon^k_p - 1] = Views^k_p[0] = 1 \le Horizon^k_p$.

The rest of the proof concentrates on the case that $Horizon^k_p > 1$. At time $k' = k + 1$ if p does not update $Views^{k+1}_p[Horizon^k_p - 2]$ due to Line 8, then $Views^{k+1}_p[Horizon^k_p - 2] \le 2$; and in general, if at time $k' = k + j$ p does not update $Views^{k+j}_p[Horizon^k_p - 1 - j]$ then $Views^{k+j}_p[Horizon^k_p - 1 - j] \le 1 + j$. Notice that if p does update $Views_p[Horizon^k_p - 1 - j]$ due to Line 8 then p has $Views_p[Horizon^k_p - 1 - j] = 1 \le 1 + j$. Therefore, for the case where $Horizon^k_p > 1$ we have shown that $Views^{k+j}_p[Horizon^k_p - 1 - j] \le 1 + j$ holds for $j \ge 1$. Setting $j = k' - k$ for $k' = k + Horizon^k_p - 1$ we have that $j = Horizon^k_p - 1$; replacing it in the above leads to the claim that $Views^{k'}_p[0] \le Horizon^k_p$ holds at time $k' = k + Horizon^k_p - 1$.

Thus, both for $Horizon^k_p = 1$ and $Horizon^k_p > 1$, the inequality $Views^{k'}_p[0] \le Horizon^k_p$ holds at time $k' = k + Horizon^k_p - 1$. □

Define $\texttt{minHG}(\mathcal{F}, k) = \min_{p \in \mathsf{G}} Horizon^k_p$ and use it to define $\texttt{bestH}(\mathcal{F}, k) = \min_{k' \ge k}\{k' + \texttt{minHG}(\mathcal{F}, k' + 1)\}$. If $\mathcal{F}$ is clear from the context, we use $\texttt{bestH}(k)$.

Notice that $\mathtt{minHG}$ is similar to $\mathtt{minH}$ except that $\mathtt{minHG}$ considers only *Horizon* values of processes that never crash, while $\mathtt{minH}$ considers processes that haven't crashed yet. Also, notice that $\mathtt{bestH}$ is the equivalent of the publication time π with respect to FIRE-SQUAD (recall that π is computed according to CCFS). The following lemma shows that $\mathtt{bestH}$ does not exceed the publication time π.

LEMMA 5.13. $\mathtt{bestH}(k) \leq \pi(k)$, *for every* $k \geq 0$.

Proof. Consider the value of $\pi(k) = \min_{k' \geq k}\{\mathtt{absH}(k')\}$, and denote by k'' the latest time for which the minimum is reached. *I.e.,* $\pi(k) = \mathtt{absH}(k'') = k'' + t + 1 - x(k'')$, and for all $k' > k''$ it holds that $\mathtt{absH}(k') > \pi(k)$. Thus, $\delta(k''+1) = \delta(k'')$ (otherwise, $\mathtt{absH}(k''+1) \leq \mathtt{absH}(k'')$, contradicting the choice of k'').

Since $\delta(k''+1) = \delta(k'')$ it holds that no new failed processes are discovered at round $k''+1$. Consider two cases: $k'' \geq 1$ and $k'' = 0$. When $k'' \geq 1$ we have that $k''+1 \geq 2$ and therefore every non-faulty process p at time $k''+1$ has $Horizon_p^{k''+1} = t + 1 - \delta(k'')$. Thus, $\mathtt{minHG}(k''+1) = t + 1 - \delta(k'')$ leading to $k'' + \mathtt{minHG}(k''+1) = k'' + \mathtt{H\_dis}(k'') = \mathtt{absH}(k'')$.

Consider the case that $k'' = 0$. By Definition 3.1, $\delta(k'') = \delta(0) = 0$ leading to $\mathtt{H\_dis}(k'') = t + 1$. Since $Horizon_p^1 \leq t + 1$ it follows that $k'' + \mathtt{minHG}(k''+1) \leq k'' + \mathtt{H\_dis}(k'') = \mathtt{absH}(k'')$.

For both $k'' \geq 1$ and $k'' = 0$ we conclude that $k'' + \mathtt{minHG}(k''+1) \leq \mathtt{absH}(k'')$. Since $\pi(k) = \mathtt{absH}(k'')$ we conclude that $\mathtt{bestH}(k) \leq \pi(k)$. □

Based on the last two lemmas, we can now show:

LEMMA 5.14. *"safety" holds at all times* $k \geq \pi(0)$.

Proof. Let $p \in \mathsf{G}$ be a process such that $Horizon_p^{k'+1} = \mathtt{minHG}(k'+1)$. Since $k'+1 \geq 1$ and $p \in \mathsf{G}^{k''}$, we have, by Lemma 5.5, that $Horizon_p^{k''} \leq Horizon_p^{k'+1}$ holds for all $k'' \geq k'+1$.

Consider time $k'+i$ (for $i \geq 1$). We have, by Lemma 5.12, that $Views_p^{k''}[0] \leq Horizon_p^{k'+i} \leq Horizon_p^{k'+1}$ holds for every time $k'' = k' + i + Horizon_p^{k'+i} - 1$. Thus, for every time $k'' \geq k' + Horizon_p^{k'+1} = \mathtt{bestH}(0)$ it holds that $Views_p^{k''}[0] \leq Horizon_p^{k'+1} \leq \mathtt{bestH}(0)$.

By Lemma 5.13 we have that $\mathtt{bestH}(0) \leq \pi(0)$. Hence, for every time $k'' \geq \pi(0)$ it holds that $Views_p^{k''}[0] \leq \pi(0)$. Consider time $\pi(0)$. Since $r_c \leq \pi(0)$, *"simultaneity"* holds by Lemma 5.10. Therefore, if some process fires then all processes in $\mathsf{G}^{\pi(0)}$ fire. For any process $q \in \mathsf{G}^{\pi(0)}$. If q fires, then it sets all $Requests_q^{\pi(0)}[i] = 0$ for all $i \geq Views_q^{\pi(0)}[0]$. If q does not fire, then it is because $Requests_q^{\pi(0)}[i] = 0$ for all $i \geq Views_q^{\pi(0)}[0]$. Moreover, since $Views_q^{\pi(0)}[0] \leq \pi(0)$, it holds that $Requests_q^{\pi(0)}[i] = 0$ for all $i \geq \pi(0)$.

Since for every $k'' \geq \pi(0)$ it holds that $Views_p^{k''}[0] \leq \pi(0)$, we have that if process p has $Requests_p^k[i] = 1$, it must have been set at some time ≥ 0. In other words, if a fire action occurs then there was a previous GO input received; and because $Requests_p^k[i]$ is zeroed once a fire action occurs, each GO can induce at most a single fire action. Thus, the number of times k' in the range $\mathtt{bestH}(0) \leq k' \leq k$ for which a fire action occurs is not larger than the number of times $0 \leq k' < k$ during which a GO input is received. □ The following lemma (together with Lemma 5.13) shows that FIRE-SQUAD fires by time $\pi(k)$ given a GO input at time k. The lower bound in Theorem 3.7 implies that this is optimal.

LEMMA 5.15. *Let input* $\mathcal{I}$ *be sequential w.r.t.* (FIRE-SQUAD, $\mathcal{S}, \mathcal{F}$). *If* $\mathcal{I}_p^k = 1$ *for process* p *at time* k *then* $\mathcal{O}_p^{k'} = 1$ *for some* k' *satisfying* $k < k' \leq \mathtt{bestH}(k)$.

Proof. Since $\mathcal{I}$ is sequential and $\mathcal{I}_p^k = 1$ it holds that $p \in \mathsf{G}$. Consider $\texttt{bestH}(0) = \min_i\{i + \texttt{minHG}(i+1)\}$, and denote by k' a time that satisfies $k' + \texttt{minHG}(k'+1) = \texttt{bestH}(0)$. Let $q \in \mathsf{G}$ be some process such that $\mathit{Horizon}_q^{k'+1} = \texttt{minHG}(k'+1)$. Since $k'+1 \geq 1$ and $q \in \mathsf{G}$, we have by Lemma 5.12 that $\mathit{Views}_q^{k''}[0] \leq \mathit{Horizon}_q^{k'+1}$ holds at time $k'' = k' + \mathit{Horizon}_q^{k'+1} = \texttt{bestH}(0)$.

If p fires at some time $k < k'' < \texttt{bestH}(k)$ then the claim is proved. Otherwise, at time $k'' = \texttt{bestH}(k)$ it holds that $\mathit{Views}_q^{k''}[0] \leq \mathit{Horizon}_q^{k'+1}$. Since $\mathcal{I}_k^p = 1$ and $p \in \mathsf{G}$, by time $k+1$ we have that $\mathit{Requests}_q^{k+1}[1] = 1$. Since p does not fire before time $\texttt{bestH}(k)$ and since *"simultaneity"* holds, we have that $\mathit{Requests}_q^{k''}[\mathit{Horizon}_q^{k'+1}] = 1$ holds by time $k'' = \texttt{bestH}(k)$. Therefore, at time $k'' = \texttt{bestH}(k)$ q will fire and due to *"simultaneity"* p will fire as well. We conclude that $\mathcal{O}_p^{k''} = 1$ is guaranteed to hold for some k'' satisfying $k < k'' \leq \texttt{bestH}(k)$. □

Following are the main results of the paper, stated in three different theorems.

THEOREM 5.16. FIRE-SQUAD *solves the SSFS problem.*

Proof. Consider any initial state $\mathcal{S}$, any input pattern $\mathcal{I}$ and any failure pattern $\mathcal{F}$. By definition, $\pi(\mathcal{F}, 0) \leq t+1$. Thus, by Lemma 5.14, *"safety"* holds starting from time $t+1$. Since by time $t+1$ there is a clean round, by Lemma 5.10, *"simultaneity"* holds starting from time $t+1$. Lemma 5.11 completes the proof, and we have that $\texttt{stab}(\text{FIRE-SQUAD}, \mathcal{S}, \mathcal{I}, \mathcal{F}) \leq k$ holds at time $k = t+1$. □

THEOREM 5.17. FIRE-SQUAD *optimally stabilizes.*

Proof. By Lemma 5.14, the *"safety"* property of FIRE-SQUAD holds from time $\pi(\mathcal{F}, 0)$. Moreover, by Lemma 5.10 together with the fact that by time $\pi(\mathcal{F}, 0)$ there is a clean round, the *"simultaneity"* property of FIRE-SQUAD holds from time $\pi(\mathcal{F}, 0)$. Combined with Lemma 5.11 we have that $\texttt{stab}(\text{FIRE-SQUAD}, \mathcal{S}, \mathcal{I}, \mathcal{F}) \leq \pi(\mathcal{F}, 0)$; for any state $\mathcal{S}$, input pattern $\mathcal{I}$ and failure pattern $\mathcal{F}$. *I.e.*, $\max_{\mathcal{S},\mathcal{I}}\{\texttt{stab}(\text{FIRE-SQUAD}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} \leq \pi(\mathcal{F}, 0)$.

Let $\mathcal{A}$ be any SSFS algorithm. By Theorem 3.6 for every failure pattern $\mathcal{F}$ we have that $\max_{\mathcal{S},\mathcal{I}}\{\texttt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} \geq \pi(\mathcal{F}, 0)$. Thus, for every $\mathcal{F}$: $\max_{\mathcal{S},\mathcal{I}}\{\texttt{stab}(\text{FIRE-SQUAD}, \mathcal{S}, \mathcal{I}, \mathcal{F})\} \leq \max_{\mathcal{S},\mathcal{I}}\{\texttt{stab}(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{F})\}$. □

THEOREM 5.18. FIRE-SQUAD *is optimally swift.*

Proof. Let input $\mathcal{I}$ be sequential w.r.t. $(\text{FIRE-SQUAD}, \mathcal{S}_{\text{FIRE-SQUAD}}, \mathcal{F})$. By Lemma 5.15, if $\mathcal{I}_p^k = 1$ for some process p at time k then $\mathcal{O}_p^{k'} = 1$ holds for some k' satisfying $k < k'' \leq \texttt{bestH}(k)$. Therefore, by time $\texttt{bestH}(k)$ we have that $\#[(\text{FIRE-SQUAD}, \mathcal{S}_{\text{FIRE-SQUAD}}, \mathcal{I}, \mathcal{F}), \texttt{bestH}(k)]$ is no smaller than the number of GO inputs received by time k.

Let $\mathcal{A}$ be an SSFS algorithm and let $\mathcal{I}$ be sequential w.r.t. $(\mathcal{A}, \mathcal{S}_{\mathcal{A}}, \mathcal{F})$. By Theorem 3.7, for every $k \geq 0$ for which a GO input is received in $\mathcal{I}^k$ there is no fire action in $\mathcal{O} = \mathcal{A}(\mathcal{S}_{\mathcal{A}}, \mathcal{I}, \mathcal{F})$ during times k' satisfying $k < k' < \pi(\mathcal{F}, k)$. Since $\texttt{bestH}(k) \leq \pi(k)$ (Lemma 5.13), it holds that by time $\texttt{bestH}(k)$, the value of $\#[(\mathcal{A}, \mathcal{S}_{\mathcal{A}}, \mathcal{I}, \mathcal{F}), \texttt{bestH}(k)]$ is at most equal to the number of GO inputs received by time k.

Thus, for every $\mathcal{S}_{\mathcal{A}}, \mathcal{S}_{\text{FIRE-SQUAD}}, \mathcal{F}$ and sequential $\mathcal{I}$ it holds that $\#[(\text{FIRE-SQUAD}, \mathcal{S}_{\text{FIRE-SQUAD}}, \mathcal{I}, \mathcal{F}), k] \geq \#[(\mathcal{A}, \mathcal{S}_{\mathcal{A}}, \mathcal{I}, \mathcal{F}), k]$, for all k. □

6. Conclusions and Open Problems. This paper presents FIRE-SQUAD, the first self-stabilizing firing squad algorithm. FIRE-SQUAD is optimal in two important respects: It optimally stabilizes, and is optimally swift. There are many directions in which this work can be extended. These include:

- FIRE-SQUAD assumes the crash fault model. What can be said about the omission fault model? And what about the *Byzantine* fault model? Each such extension seems to be a nontrivial step.

- FIRE-SQUAD works when we assume that failures are permanent. Being an ongoing and everlasting service, firing squad is expected to operate for long periods, in which processes may recover. A more reasonable assumption in this case is that there is a bound (of t) on the number of failures over every interval of m rounds, for some m. (Non-stabilizing) Continuous consensus has recently been studied in this model [14], and it would be interesting to see if the same can be done for self-stabilizing firing squad.

REFERENCES

[1] R. Bazzi and G. Neiger. The possibility and the complexity of achieving fault-tolerant coordination. In *PODC '92*, pages 203–214, New York, USA, 1992. ACM.

[2] J. E. Burns and N. A. Lynch. The byzantine firing squad problem. *Advances in Computing Research: Parallel and Distributed Computing*, 4:147–161, 1987.

[3] B. A. Coan, D. Dolev, C. Dwork, and L. J. Stockmeyer. The distributed firing squad problem. *SIAM J. Comput.*, 18(5):990–1012, 1989.

[4] D. Dolev, R. Reischuk, and R. H. Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.

[5] S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Real-Time Systems*, 12(1):95–107, January 1997.

[6] S. Dolev. *Self-Stabilization.* The MIT Press, 2000.

[7] S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *Journal of the ACM*, 51(5):780–799, 2004.

[8] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.

[9] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, Mass., 1995.

[10] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in PODC '84.

[11] E. N. Hoch, D. Dolev, and A. Daliot. Self-stabilizing byzantine digital clock synchronization. In *Proc. of 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'06)*, pages 350–362, Nov 2006.

[12] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

[13] T. Mizrahi and Y. Moses. Continuous consensus via common knowledge. *Distributed Computing.*, 20(5):305–321, 2008.

[14] T. Mizrahi and Y. Moses. Continuous consensus with failures and recoveries. In *DISC'08*, pages 408–422, 2008.

[15] Y. Moses and M. Raynal. Revisiting simultaneous consensus with crash failures. *J. Parallel Distrib. Comput.*, 69(4):400–409, 2009.

[16] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[17] G. Neiger and M. R. Tuttle. Common knowledge and consistent simultaneous coordination. *Distrib. Comput.*, 6(3):181–192, 1993.

[18] B. Patt-Shamir. *A Theory of Clock Synchronization.* Doctoral thesis, MIT, Oct 1994.