

A Fault-Resistant Asynchronous Clock Function

Ezra N. Hoch, Michael Ben-Or, Danny Dolev

The Hebrew University of Jerusalem
{ezraho,benor,dolev}@cs.huji.ac.il

Abstract. Consider an asynchronous network in a shared-memory environment consisting of n nodes. Assume that up to f of the nodes might be *Byzantine* ($n > 12f$), where the adversary is full-information and dynamic (sometimes called adaptive). In addition, the non-*Byzantine* nodes may undergo transient failures. Nodes advance in atomic steps, which consist of reading all registers, performing some calculation and writing to all registers.

The three main contributions of the paper are: first, the clock-function problem is defined, which is a generalization of the clock synchronization problem. This generalization encapsulates previous clock synchronization problem definitions while extending them to the current paper's model. Second, a randomized asynchronous self-stabilizing *Byzantine* tolerant clock synchronization algorithm is presented.

In the construction of the clock synchronization algorithm, a building block that ensures different nodes advance at similar rates is developed. This feature is the third contribution of the paper. It is self-stabilizing and *Byzantine* tolerant and can be used as a building block for different algorithms that operate in an asynchronous self-stabilizing *Byzantine* model.

The convergence time of the presented algorithm is exponential. Observe that in the asynchronous setting the best known full-information dynamic *Byzantine* agreement also has an expected exponential convergence time.

1 Introduction

When tackling problems in distributed systems, there are many previously developed building blocks that assist in solving the problem. Some of these building blocks allow one to design a solution under “easy” assumptions, then automatically transform them to a more realistic environment. For example, it is easier to construct an algorithm in the synchronous model, then add an underlying synchronizer (see [4]) to adapt the solution to an asynchronous model. Similarly, developing a self-stabilizing algorithm can be challenging; instead, one can develop a non-self-stabilizing algorithm, and use a stabilizer ([1]) to address transient errors.

Among the different models of distributed systems, specific models received more attention than others; and therefore the availability and versatility of building blocks differ from one model to another. For example, the synchronous no-failures model can automatically be extended in many different directions: asynchronous no-failures, synchronous self-stabilizing, asynchronous self-stabilizing, etc. Zooming-in to the world of self-stabilizing, there are various model-convertors: between shared-memory and message-passing, from an id-based to uniform system, etc. (see [11]).

However, when moving away from the commonly researched models, the availability of such model-converters diminishes. In the current paper we are interested in an asynchronous network with *Byzantine* nodes and transient failures. That is, we aim at solving a problem in a way that is *Byzantine* tolerant, self-stabilizing and operates in an asynchronous network. The *Byzantine* adversary is assumed to be full-information and dynamic (sometimes called adaptive). There are few previous works that operate in similar models [18,19,17]. In these works non-faulty neighbors of *Byzantine* nodes may reach undesired states. However, as far as we know, this is the first work operating in such a setting in which non-*Byzantine* nodes reach their desired state even if they have *Byzantine* neighbors.

The problem we solve in the current work is clock-synchronization. Our solution assumes two simplifying assumptions: a) a “centralized daemon”, *i.e.*, each node can run the entire algorithm as an atomic step; b) an “en masse scheduler” that adheres to the following: if p gets scheduled twice, then $n - 2f$ other non-faulty nodes get scheduled in between (formally defined in Definition 2). Under these assumptions we define and solve the clock synchronization problem in an asynchronous network while tolerating both *Byzantine* and transient faults. The solution is a randomized algorithm with expected convergence time of $O(3^{n-2f})$.

Both assumptions can be seen as “building blocks that do not yet exist”. When constructing a self-stabilizing asynchronous algorithm (without *Byzantine* nodes), it is reasonable to assume a centralized daemon due to the mutual exclusion algorithm of Dijkstra (see [8]). Thus, once an equivalent algorithm can be devised for this paper’s model, the first assumption can be removed. In Section 6 we provide an algorithm that implements the second assumption, thus allowing its usage without reducing the generality of a solution that uses it.

Due to our dependence on the first assumption, we consider this work as a step towards a full solution of the clock synchronization in an asynchronous network that is self-stabilizing and *Byzantine* tolerant. We hope it leads to further research of this model, one which will produce an equivalent of Dijkstra’s algorithm operating in the current work’s model.

Related Work Being able to introduce consistent “time” in a distributed system is an important task, however difficult it may be in some models. For many distributed tasks the crux of the problem is to synchronize the operations of the different nodes. One method of doing this is using some sort of “time-awareness” at each node, ensuring that different “clocks” advance in a relatively synchronized manner. Therefore, it is interesting to devise such algorithms that are highly robust.

In the past, various models were considered. Ranging from synchronous systems (see [10,12,15]), in which all nodes receive a common signal simultaneously at regular intervals; through bounded-delay systems (see [9,?]), in which only a bound on the message delivery time is given; to completely asynchronous systems (see [7,14]), in which only an eventual (but not bounded) delivery of messages

is assumed. Independent of the timing model, different fault tolerance assumptions are considered: the self-stabilizing fault paradigm, in which all nodes follow their protocol but may start with arbitrary values of their variables and program counter (see [11]). Another commonly assumed faults are the Fail-stop faults, in which some of the nodes may crash and cease to participate in the protocol. Lastly, *Byzantine* faults are considered to be the most severe fault model, as they assume that the faulty nodes can behave arbitrarily and even collude in trying to keep the system from reaching its designated goals (see [3,16]).

“Knowing what time it is” acquires different flavors in different models. In systems without any faults, it is usually assumed that each node has a physical clock, and these clocks differ from node to node. The main issue is to synchronize the different clocks as close as possible. In a synchronous, self-stabilizing and *Byzantine* tolerant model, this problem was termed “digital clock synchronization”, and consisted of having all nodes agree on some bounded integer and increase it every round (see [2,10,12,15]).

The traditional concept of “clock synchronization” does not hold in an asynchronous environment. Therefore, previous work has defined “phase clocks” or “unison” (see [7,14]), which states that each node has an integer valued clock, and neighboring nodes should be at most ± 1 from each other. It is shown (for example, see [14]) how such “synchronization” is sufficient in solving many problems.

Most previous works in the asynchronous model considered self-stabilizing or *Byzantine* faults, but not both. In the current work, we consider both fault models. However, defining what “telling the time” means in an asynchronous, self-stabilizing and *Byzantine* tolerant manner is a bit tricky. To address that, a new notion of “knowing what time it is” is introduced: *a clock function*.

All previous clock-synchronization (or phase-clock, or unison, etc.) algorithms can be viewed in the following way: each time a non-faulty node is running, it executes some piece of code (“function”) that returns a value (“the clock value”) and there are constraints on the range of different non-faulty nodes’ values. In the synchronous digital clock synchronization problem, the function returns an integer value, and we require that all nodes executing the function at the same round receive exactly the same value and a node executing the function in consecutive rounds receives consecutive values. In an asynchronous network (*i.e.*, in [14]), different nodes may execute their clock-functions at different times and at different rates. The constraint on the returned values can be described as follows: given a configuration of the system, if p would execute its clock-function and receive a value v , then any neighbor of p that would execute its clock-function at the same configuration, would receive a value that differs by at most ± 1 from v .

In the current work it is assumed that the network is fully connected, which means that every node is connected to every other node. Therefore, the constraint of the clock-function is simplified, informally requiring any two non-faulty nodes that execute the clock-function to receive values that are at most one apart.

In synchronous networks the problem of self-stabilizing *Byzantine* tolerant clock synchronization is equivalent to the problem of *Byzantine* agreement, in the sense that any solution to the self-stabilizing *Byzantine* tolerant clock synchronization problem is also a solution to the (non-self-stabilizing) *Byzantine* agreement problem. In asynchronous networks the best known full-information dynamic *Byzantine* agreement has expected exponential convergence time (see [5]). While the synchronous equivalence between clock synchronization and *Byzantine* agreement does not transfer to the asynchronous setting (as it strongly uses the fact that all nodes agree on the exact same clock value), it raises the possibility that improving the result of this paper will require usage of new techniques. That is, it is not known yet if the self-stabilizing clock synchronization of the current work can be used to solve *Byzantine* agreement. However, if it can be used, then improving the exponential convergence of the current work would lead to an improvement of the best known asynchronous *Byzantine* agreement against a dynamic full-information adversary.

Contribution Our contribution is three-fold. First, we define the clock-function problem, which is a generalization of the clock synchronization problem. This definition provides a meaningful extension of the clock synchronization problem to the asynchronous self-stabilizing *Byzantine* tolerant model.

Second, we provide an algorithm that solves the clock-function problem in the above model. Using shared memory, it has an expected $O(3^{n-2f})$ convergence time, independent of the wraparound value of the clock. Notice that for synchronous networks, the first two contributions were already presented in [12]. Our contribution is with respect to asynchronous networks.

Lastly, in Section 6 we construct a building block that bounds the relative rates at which different non-faulty nodes progress with respect to other non-faulty nodes. More specifically, between any two atomic steps of a non-faulty node p , there are guaranteed to be atomic steps of $n - 2f$ other non-faulty nodes. (See the “en masse scheduler” assumption described in the introduction). We postulate that this building block can be used in other asynchronous self-stabilizing *Byzantine* tolerant settings.

Overview We start by defining the model (see Section 2). A subset of all possible runs is defined and denoted “en masse” (see Definition 2). Section 3 discusses different aspects of defining clock synchronization in an asynchronous, self-stabilizing, *Byzantine* tolerant environment; and defines a clock function, which is a generalization of the clock synchronization problem.

Section 4 introduces ASYNC-CLOCK, an algorithm that solves the problem at hand. Section 5 contains the correctness proof for ASYNC-CLOCK. Both Section 4 and Section 5 are correct only for en masse runs, for which ASYNC-CLOCK requires fault redundancy of $n > 6f$.

In [Section 6](#) the algorithm ENMASSE is presented, which transforms any run into an en masse run. Leading to the correctness of ASYNC-CLOCK for any run. However, the transformation done by ENMASSE increases the fault redundancy of ASYNC-CLOCK to $n > 12f$. Lastly, [Section 7](#) concludes with a discussion of the results.

2 Distributed Model

The system is composed of a set of n nodes denoted by \mathcal{P} . Every pair of nodes $p, q \in \mathcal{P}$ communicates via shared memory (*i.e.*, a fully-connected communication graph), in an asynchronous manner. That is, p and q share two registers: $R_{p,q}, R_{q,p}$.¹ Register $R_{p,q}$ is written by p and read by q .² A configuration \mathcal{C} describes the global state of the system and consists of the states of each node and the state of each register. A run of the system is an infinite sequence of configurations $\mathcal{C}_0 \rightarrow \mathcal{C}_1 \rightarrow \dots \rightarrow \mathcal{C}_r \rightarrow \dots$, such that the configuration \mathcal{C}_{r+1} is reachable from configuration \mathcal{C}_r by a single node’s atomic step. In the context of the current paper, an atomic step consists of reading all registers, performing some calculation and then writing to all registers.

The system is assumed to start from an arbitrary initial configuration \mathcal{C}_0 . We show that eventually - in the presence of continuous *Byzantine* behavior - the system becomes synchronized.

In addition to transient faults, up to f of the nodes may be *Byzantine*. The *Byzantine* adversary has full information, *i.e.*, it can read the values in every node’s memory³ and in the shared registers between any two nodes. There are no private channels and the adversary is computationally unbounded. Moreover, the adversary is dynamic, which means it may choose to “capture” a non-faulty node at any stage of the algorithm. However, once the adversary has “captured” f nodes in some run, it cannot affect other nodes and in a sense becomes static. The results of this paper can be extended to the setting in which the adversary continues to be dynamic throughout the run, as long as the adversary is limited by the rate at which it can release and capture non-faulty nodes. We do not present this extension in the current paper for the sake of clarity. However, one can easily be convinced that it applies, once the main points of the work are explained.

The adversary also has full control of the scheduling of atomic steps and can use its full information knowledge in this scheduling. However, for a clock synchronization algorithm to be meaningful, runs in which some of the non-faulty

¹ Pair-wise communication is used to allow *Byzantine* nodes to present different values to different nodes; as opposed to assuming a single register per-node that can be read by all other nodes.

² For simpler presentation we assume that p writes and reads $R_{p,p}$.

³ Actually, the presented algorithm stores all its state in the shared registers.

nodes never get to perform atomic steps should be excluded. Thus, throughout the paper only fair runs are considered:

Definition 1. A run is **fair** if every non-faulty node performs infinitely many atomic steps.

A subset of all fair runs is defined:

Definition 2. A run \mathcal{T} is **en masse with respect** to node p if for any 2 atomic steps p performs during \mathcal{T} (say at configurations \mathcal{C} and \mathcal{C}' , respectively) there are at least $n - 2f$ non-faulty nodes that perform atomic steps between \mathcal{C} and \mathcal{C}' .

A run \mathcal{T} is **en masse** if it is fair and it is en masse with respect to all non-faulty nodes.

As stated in the overview, en masse runs are needed for ASYNC-CLOCK to operate correctly. Assuming all runs are en masse runs, the fault tolerance redundancy required is $n > 6f$ (see Lemma 3 for an example of the necessity of $n > 6f$). However, in Section 6 we show how to remove the requirement of en masse runs, at the cost of increasing the fault tolerance redundancy to $n > 12f$.

3 Problem Definition

Before formally defining the problem at hand, consider the properties a distributed clock synchronization algorithm should have in an asynchronous setting:

1. (*clock-value*) a means of locally computing the current clock value at any non-faulty node;
2. (*agreement*) if different non-faulty nodes compute the clock value close (in time) to each other, they should obtain similar values;
3. (*liveness*) if non-faulty nodes continuously recompute clock values, then they should obtain increasing values.

For example, in a synchronous network, the clock synchronization problem is usually formulated as: (*clock-value*) each node p has a bounded integer counter $Clock_p$; (*agreement*) for any two non-faulty nodes p, q it holds that $Clock_p = Clock_q$; (*liveness*) if $Clock_p = z$ at round r then $Clock_p = z + 1$ at round $r + 1$. Since the clock is bounded, the previous sentence is slightly modified: “if $Clock_p = z$ at round r , then $Clock_p = z + 1 \pmod{k}$ at round $r + 1$ ”; where k represents the wrap-around value of the clock.

NOTATION 31 Denote by $a \oplus_k b$ the value $(a + b \pmod{k})$.

In an asynchronous setting, it is impossible to ensure that all nodes update their clocks simultaneously. Thus, the “agreement” property requires a relaxed version as opposed to the synchronous setting’s stricter version. In addition, the “liveness” property is somewhat tricky to define, due to the *Byzantine* presence.

To illustrate the difficulty, consider a set of f *Byzantine* nodes that “behave as if” they were non-faulty, and they repeatedly recompute the clock value. According to the definition above, the clock value will increase continuously, even though non-faulty nodes did not perform a single step. Therefore, such a clock synchronization algorithm is useless, as the *Byzantine* nodes can make it reach any clock value; in other words, the *Byzantine* nodes “control” the clock value.

It is not immediately clear how these “benign” *Byzantine* nodes can be differentiated from the non-faulty nodes. The following definitions address such difficulties, and present a formalization of the clock synchronization problem in this paper’s model.

Definition 3. *A value v' is at most d ahead of v if there exists j , $0 \leq j \leq d$, such that $v \oplus_k j = v'$. Denote “ v' is at most d ahead of v ” by $v \preceq_d v'$.*

Definition 4 addresses the “clock-value” property:

Definition 4. *A clock-function \mathcal{F} is an algorithm that when executed during an atomic step returns a value in the range $\{0, \dots, k - 1\}$. Denote by $\mathcal{F}_p(\mathcal{C})$ the value returned when p executes \mathcal{F} during an atomic step at configuration \mathcal{C} .*

Consider the “agreement” property: it requires that different non-faulty nodes that compute the clock value simultaneously, receive similar values. What does “simultaneously” mean in an asynchronous setting? It can be captured by requirements on the clock values computed in different runs. In addition, the interference caused by *Byzantine* nodes in different runs needs to be captured.

Informally, “agreement” requires the following from \mathcal{F} : given a configuration \mathcal{C} , no matter what the adversary does, if different non-faulty nodes execute \mathcal{F} they receive values that are close to each other. Definition 5, Definition 6 and Definition 7 formally state the “agreement” requirement. First, “no matter what the adversary does” is formally defined:

Definition 5. *An adversarial move from a configuration \mathcal{C} is any configuration reachable by an arbitrary sequence of atomic steps of faulty nodes only.*

Second, “different non-faulty nodes execute \mathcal{F} ” is divided into two cases. Let p, q be non-faulty nodes. The first case considers the computed value of p (when calculating \mathcal{F} on \mathcal{C}) as opposed to the computed value of q (see Definition 6). The second case considers the computed value of q after p has computed its value (see Definition 7). Both cases require the computed values of p and q to be close to each other.

Definition 6. *A configuration \mathcal{C} is ℓ -well-defined (with respect to some clock-function \mathcal{F}) if there is a value v s.t. for any non-faulty node p and every adversarial move \mathcal{C}' from \mathcal{C} it holds that $v \preceq_\ell \mathcal{F}_p(\mathcal{C}')$. v is called “a defined value” at \mathcal{C} . (There may be more than one such v).*

Informally, [Definition 6](#) says that \mathcal{C} is ℓ -well-defined if there is an intrinsic value v such that any adversarial move cannot increase the clock-value by more than ℓ . Thus, any two non-faulty nodes p, q (in different run extensions from \mathcal{C}) that execute \mathcal{F} on \mathcal{C} (no matter what the adversary has done) will receive values in the range $\{v, \dots, v + \ell\}$; *i.e.*, p and q 's values are at most ℓ apart.

Suppose \mathcal{C}_0 is ℓ -well-defined with value v , and that a non-faulty node p performs an atomic step at \mathcal{C}_0 resulting in \mathcal{C}_1 and then a non-faulty node q performs an atomic step at \mathcal{C}_1 . [Definition 6](#) does not imply any constraint on the value of $\mathcal{F}_p(\mathcal{C}_0)$ with respect to $\mathcal{F}_q(\mathcal{C}_1)$, therefore the following definition is required:

Definition 7. *A run is ℓ -well-defined (w.r.t. a clock-function \mathcal{F}) if: a) every configuration \mathcal{C} in the run is ℓ -well-defined; b) for two consecutive configurations $\mathcal{C}, \mathcal{C}'$, if v is a defined value of \mathcal{C} and v' is a defined value of \mathcal{C}' then $v \preceq_\ell v'$.*

[Definition 7](#) states that the values of a clock-function \mathcal{F} on consecutive configurations cannot be arbitrary. That is, they must be at most ℓ apart from the previous configuration. However, there is no requirement that they actually increase; *i.e.*, “liveness” is not captured by the previous definitions.

Definition 8. *A run is ℓ -clock-synchronized (w.r.t. some clock-function \mathcal{F}), if it is ℓ -well-defined (w.r.t. \mathcal{F}) and the defined values of consecutive configurations change infinitely many times. (I.e., for infinitely many consecutive configurations $\mathcal{C}, \mathcal{C}'$ the defined values of \mathcal{C} differ from the defined values of \mathcal{C}').*

Notice that [Definition 7](#) already requires that defined values of consecutive configurations are non-decreasing (assuming that ℓ is sufficiently small with respect to k). Thus, combined with [Definition 8](#), it implies that in an ℓ -clock-synchronized run, infinitely many configurations are configurations with increasing defined values (informally, “increasing” means that one defined value is achieved by adding less than $\frac{k}{2}$ to a previous defined value).

Remark 1. [Definition 7](#) and [Definition 8](#) impose requirements on the *defined* values of consecutive configurations. However, a specific node p might compute a clock value that is decreasing between consecutive configuration. *i.e.*, p 's clock might “go backward”. For example, let $\mathcal{C}, \mathcal{C}'$ be two consecutive configurations, and let the defined value of both configurations be v . It is possible that p will compute the clock value to be $v + 1$ for \mathcal{C} , while computing the clock value to be v for \mathcal{C}' .

However, this possibility is immanent to an asynchronous *Byzantine* tolerant clock synchronization that has a wraparound value k . Consider a setting in which all nodes but one are advanced in a synchronous manner, while a single node p performs atomic steps only once every $k - 1$ rounds. In such a setting, p should update its clock value to be slightly below its previous value (alternatively, it can be seen as increasing the value by $k - 1$).

```

Algorithm ASYNC-CLOCK /* executed on node q */


---


01: do forever:

    /* read all registers */
02:   for  $i := 1$  to  $n$ 
03:     set  $val_i := \text{read } R_{p_i, q} \bmod k$ ;

    /* some internal definitions */
04:   let  $\#v$  denote the number of times  $v$  appears in  $\{val_i\}_{i=1}^n$ ;
05:   let  $\text{count}(v, l)$  denote  $\sum_{j=0}^l \#(v \oplus_k j)$ ;
06:   let  $\text{pass}(l, a)$  denote  $\{v | \text{count}(v, l) \geq a\}$ ;

    /* update my_val */
07:   if  $\text{pass}(0, n - f) \neq \emptyset$  then
08:     set  $my\_val_q := 1 \oplus_k \max\{\text{pass}(0, n - f)\}$ ;
09:   else if  $\text{pass}(1, n - f) \neq \emptyset$  then
10:     set  $my\_val_q := 1 \oplus_k \max\{\text{pass}(1, n - f)\}$ ;
11:   else if  $\text{pass}(1, n - 2f) \neq \emptyset$  then
12:     let  $low \notin \text{pass}(1, n - 2f)$  be such that  $low \oplus_k 1 \in \text{pass}(1, n - 2f)$ ;
13:     let  $relative\_median = \min\{l | l \geq 0 \ \& \ \text{count}(low, l) > \frac{n}{2}\}$ ;
14:     set  $my\_val_q := low \oplus_k relative\_median$ ;
15:   else set  $my\_val_q :=$  randomly select a value from  $\text{pass}(1, n - 3f) \cup \{0\}$ ;

    /* write my_val to registers */
16:   for  $i := 1$  to  $n$ 
17:     write  $my\_val_q$  into  $R_{q, p_i}$ ;
18: od;

```

Fig. 1. A self-stabilizing *Byzantine* tolerant algorithm solving the 5-Clock-Synchronization problem.

Definition 9. An algorithm \mathcal{A} solves the ℓ -clock-synchronization problem if there is a clock-function \mathcal{F} s.t. any fair run starting from any arbitrary configuration has a suffix that is ℓ -clock-synchronized with respect to \mathcal{F} .

An ideal protocol would solve the 0-clock-synchronization problem. However, due to the asynchronous nature of the discussed model, the best that can be expected is to solve the 1-clock-synchronization problem. We aim at solving the ℓ -clock-synchronization problem for as many values of $\ell \geq 1$ as possible. Clearly, if \mathcal{A} solves the ℓ_1 -clock-synchronization problem, then \mathcal{A} also solves the ℓ_2 -clock-synchronization problem for any $\frac{k}{2} > \ell_2 \geq \ell_1$.

Therefore, the rest the paper concentrates on solving the 5-clock-synchronization problem; thus, solving the ℓ -clock-synchronization problem for all $\frac{k}{2} > \ell \geq 5$. In [Section 7.1](#) we show how to use any $\frac{k-1}{2}$ -clock-synchronization problem to solve the 1-clock-synchronization problem, thus solving the ℓ -clock-synchronization problem for all $\frac{k}{2} > \ell \geq 1$.

4 Solving the 5-Clock-Synchronization Problem

An atomic step consists of reading all registers, performing some calculations and writing to all registers. Thus, an atomic step consists of executing once an entire “loop” of ASYNC-CLOCK (see Figure 1).

Each non-faulty node p has a bounded integer variable, my_val_p , which represents the current clock value of p . When p performs an atomic step, it reads all of its registers, thus getting an impression of the clock values of the other nodes. It then computes its own new clock value (which is saved in my_val_p) and writes my_val_p to all registers.

ASYNC-CLOCK operates in a similar fashion to many other *Byzantine* tolerant algorithms. It first gathers information regarding the clock value of the other nodes in the system. Then it uses various thresholds to decide on the clock value for the next step. If no threshold works (*i.e.*, no clear majority is found), it chooses a random value from a small set of options.

To ensure all values read during Line 02–03 are in the range $[0, \dots, k - 1]$, the algorithm applies “mod k ” to the values read. This is a standard way of dealing with uninitialized values.

The crux of ASYNC-CLOCK is in the exact thresholds and their application (Lines 07–15). In these lines, node p considers different possibilities. Either it sees a decisive majority towards some clock value (Line 07 and Line 09) in which case p updates its local clock value to coincide with the majority clock value it has seen. Alternatively, if no clear majority exists (Line 15), p randomly selects a new clock value. The interesting case is when p sees a “partial” majority (Line 11), in which case p takes the relative median of the clock values it has seen. We call this a “relative median” since the clock values are “mod k ” and thus the median is not well defined.

The full ASYNC-CLOCK algorithm appears in Figure 1. ASYNC-CLOCK solves the ℓ -Clock-Synchronization problem for $\ell = 5$; combined with the discussion at the end of Section 3, it shows how to solve the ℓ -Clock-Synchronization problem for any $\frac{k}{2} > \ell \geq 5$.

NOTATION 41 The value of any variable var at configuration C_r is denoted by $C_r(var)$. For a node q that does not perform the atomic step that changes C_r to C_{r+1} , the value of my_val_q , denoted $C_r(my_val_q)$, is the same before and after the atomic step.

For node p that performs the atomic step at configuration C_r , my_val is the only variable that is not deterministically determined by the values of the registers at the beginning of p ’s atomic step. In such a case, for my_val , the notation $C_r(my_val_p)$ denotes the value of my_val before p starts its atomic step, and $C_{r+1}(my_val_p)$ denotes the value of my_val after p finishes its atomic step. For all other variables, $C_r(var)$ will denote the value of variable var , as computed for configuration C_r ; *i.e.*, $C_r(\text{pass}_p(1, n - 2f))$ denotes the value that p computes for $\text{pass}(1, n - 2f)$ during its atomic step at C_r .

5 Correctness Proof

In the following discussion we consider the system only after all transient faults ended and each non-faulty node has taken at least one atomic step. We consider only runs of the system that begin after that initial sequence of atomic steps.

Informally, a round is a portion of a run such that each node that is non-faulty throughout the round performs an atomic step at least once. The first round (of a run \mathcal{T}) is the minimal prefix \mathcal{R} of the run \mathcal{T} such that each node that is non-faulty throughout \mathcal{R} performs an atomic step at least once. Consider the suffix \mathcal{T}' of \mathcal{T} after the first round was removed. The second round of \mathcal{T} is the first round of \mathcal{T}' ; the definition continues so recursively.

Consider any fair run of the system $\mathcal{C}_0 \rightarrow \mathcal{C}_1 \rightarrow \dots \rightarrow \mathcal{C}_r \rightarrow \dots$, and consider the transition from configuration \mathcal{C}_r to configuration \mathcal{C}_{r+1} , due to some (possibly faulty) node p 's atomic step. Since we consider only runs after each non-faulty node q has taken at least one atomic step past the end of the transient faults events, the value of my_val_q reflects the latest value written to all of q 's write-registers. This property is true for all configurations that we consider. Thus, regarding a non-faulty p that performs an atomic step, for all non-faulty q it holds that $R_{q,p} = my_val_q$.

The proof outline is as follows. First, define a tight configuration:

Second, we show that if a configuration \mathcal{C}_r is tight then so is \mathcal{C}_{r+1} . Third, if \mathcal{C}_r is not tight, then we show that with probability $\frac{1}{3^{n-2f}}$ some configuration within 2 rounds from \mathcal{C}_r will be tight. Concluding that after an expected $O(3^{n-2f})$ rounds the system reaches a tight configuration; and all following configurations are tight as well. At this stage, we need to show that the value v that a configuration is tight around continuously increases.

To do so, we show that given that all configurations are tight, different non-faulty nodes that perform atomic steps can have values from a set containing (at most) 3 consecutive values. Moreover, for consecutive configurations, the minimal value among these 3 values can increase by at most 3. Lastly, by closely analyzing the behavior of ASYNC-CLOCK, we conclude that within 4 rounds the minimal value above increases. That is, the clock function value changes, and changes again within at most 4 rounds, *i.e.*, the clock value changes infinitely many times.

The reason behind the increase of the aforementioned minimal value lies in the following claim: one of two things can happen, either the minimal value increases, or all the non-faulty nodes' clock values become at most 1 apart. In the second scenario, after one round, the minimal value will increase. Concluding that the clock value changes infinitely many times, as required.

Remark 2. The en masse property is used in the proof that if \mathcal{C}_r is not tight, then with probability $\frac{1}{3^{n-2f}}$ a configuration within 2 rounds from \mathcal{C}_r will be tight. Since in an en masse run some set of $n - 2f$ different non-faulty nodes are required to take atomic steps in a consecutive manner. Together with a claim stating that

each such step has probability of $\frac{1}{3}$ to flip a coin “in the right direction”, we get that with probability $\frac{1}{3^{n-2f}}$ a tight configuration is reached.

Lemma 1. *If $v_1 \preceq_d v'$ and $v_2 \preceq_d v'$ then either $v_2 \preceq_d v_1$ or $v_1 \preceq_d v_2$.*

Proof. By definition, there are j_1, j_2 ($0 \leq j_1, j_2 \leq d$) such that $v' = v_1 \oplus_k j_1$ and $v' = v_2 \oplus_k j_2$. Thus, $v_2 \oplus_k j_2 = v_1 \oplus_k j_1$, which means that $v_2 = v_1 \oplus_k (j_1 - j_2)$. Clearly, $|j_1 - j_2| \leq d$. If $j_1 - j_2 \geq 0$ then v_2 is at most $j_1 - j_2 \leq d$ ahead of v_1 . Otherwise, $j_2 - j_1 > 0$, meaning that $v_1 = v_2 \oplus_k (j_2 - j_1)$. That is, v_1 is at most $j_2 - j_1 \leq d$ ahead of v_2 .

We are interested in the set of non-faulty nodes that are “close” to each other with respect to their value of `my_val`.

Definition 10. $H(\mathcal{C}_r, v, d)$ is the set containing any non-faulty node q , such that $\mathcal{C}_r(\text{my_val}_q)$ is at most d ahead of v . Formally, $H(\mathcal{C}_r, v, d) = \{\text{non-faulty } q \mid v \preceq_d \mathcal{C}_r(\text{my_val}_q)\}$.

Definition 11. $H(\mathcal{C}_r, p, v, d)$ is the set containing any node q , such that $\mathcal{C}_r(R_{q,p})$ is at most d ahead of v . Formally, $H(\mathcal{C}_r, p, v, d) = \{q \mid v \preceq_d \mathcal{C}_r(R_{q,p})\}$.

Notice that $H(\mathcal{C}_r, v, d)$ contains only non-faulty nodes, while $H(\mathcal{C}_r, p, v, d)$ may contain faulty nodes. The difference stems from $H(\mathcal{C}_r, p, v, d)$ representing what p “perceives” at configuration \mathcal{C}_r , as opposed to $H(\mathcal{C}_r, v, d)$ which says “what is true” in configuration \mathcal{C}_r .

Lemma 2. $|H(\mathcal{C}_r, v, d)| \geq |H(\mathcal{C}_r, p, v, d)| - f$ and $|H(\mathcal{C}_r, p, v, d)| \geq |H(\mathcal{C}_r, v, d)|$.

Remark 3. Notice that $H(\mathcal{C}_r, p, v, d)$ contains all the nodes (including faulty nodes) whose registers’ value (in \mathcal{C}_r) is at most d ahead of v . Thus, $v \in \mathcal{C}_r(\text{pass}_p(d, x)) \neq \emptyset$ if (and only if) $|H(\mathcal{C}_r, p, v, d)| \geq x$.

Definition 12. A configuration \mathcal{C}_r is **tight** around value v if $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$; a configuration is **tight** if it is tight around some value.

Lemma 3. *If a configuration \mathcal{C}_r is tight around value v and around value $v' \neq v$, then either $v \preceq_1 v'$, or $v' \preceq_1 v$.*

Proof. By the lemma’s assumption, it holds that $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$ and $|H(\mathcal{C}_r, v', 1)| \geq n - 2f$. Since $n > 6f$, there is some non-faulty node $q \in H(\mathcal{C}_r, v, 1) \cap H(\mathcal{C}_r, v', 1)$. Thus, $\mathcal{C}_r(\text{my_val}_q)$ is at most 1 ahead of v and at most 1 ahead of v' . The rest follows from [Lemma 1](#).

Remark 4. Following the same line of proof as in [Lemma 3](#) shows that $\mathcal{C}_r(\text{pass}_p(1, n - 2f))$ can contain at most 2 values, and these values are consecutive values.

Lemma 4. *If in configuration \mathcal{C}_r , non-faulty node p performs Line 12 then $\mathcal{C}_r(\text{low}_p)$ is well defined, for $k \geq 3$.*

Proof. By [Remark 4](#), if p passes the condition of Line 11 then $\mathcal{C}_r(\text{pass}_p(1, n-2f))$ contains at most 2 values, which are consecutive. Thus, if $k \geq 3$ then $\mathcal{C}_r(\text{low}_p)$ is well defined.

Lemma 5. *If p passes the condition in Line 11 and $|H(\mathcal{C}_r, v, 1)| \geq n - 3f$ then $v \preceq_1 (\text{low} \oplus_k \text{relative_median})$, for $k > 4$.*

Proof. p passed the condition in Line 11, thus $\mathcal{C}_r(\text{pass}_p(1, n-2f)) \neq \emptyset$. Thus, for some v' it holds that $|H(\mathcal{C}_r, p, v', 1)| \geq n - 2f$ (see [Remark 3](#)), and therefore $|H(\mathcal{C}_r, v', 1)| \geq n - 3f$ (see [Lemma 2](#)). By the lemma's assumption, $|H(\mathcal{C}_r, v, 1)| \geq n - 3f$. Since $n > 6f$, there is some non-faulty node $q \in H(\mathcal{C}_r, v', 1) \cap H(\mathcal{C}_r, v, 1)$. That is $v' \preceq_1 \mathcal{C}_r(\text{my_val}_q)$ and $v \preceq_1 \mathcal{C}_r(\text{my_val}_q)$. By [Lemma 1](#) either $v \preceq_1 v'$ or $v' \preceq_1 v$.

According to Line 12 and [Remark 4](#), $\text{low} \oplus_k 1 \preceq_1 v'$. Thus, in both scenarios ($v \preceq_1 v'$ or $v' \preceq_1 v$) it holds that $\text{low} \preceq_3 v$. Informally, low is “before” v , and relative_median (see Line 13) is increased until there are more than $\frac{n}{2}$ nodes in the range $[\text{low}, \text{low} \oplus_k \text{relative_median}]$. Since there are $\geq n - 3f > \frac{n}{2}$ copies of “ v ”, relative_median will be such that $\text{low} \oplus_k \text{relative_median} \in \{v, v \oplus_k 1\}$.

Formally, relative_median is the minimal value such that $\text{count}_p(\text{low}, \text{relative_median})$ contains more than $\frac{n}{2}$ nodes. Since $|H(\mathcal{C}_r, v, 1)| \geq n - 3f > \frac{n}{2}$, at least one copy of “ v ” is counted towards the sum of $\text{count}_p(\text{low}, \text{relative_median})$. Since $k > 4$ (by the lemma's assumption), copies of v will not be counted in $\text{count}_p(\text{low}, \text{relative_median}')$ for $\text{relative_median}'$ such that $\text{low} \oplus_k \text{relative_median}' \notin \{v, v \oplus_k 1\}$. On the other hand, $\text{count}_p(\text{low}, \text{relative_median}') \geq n - 2f$ for $\text{relative_median}'$ such that $\text{low} \oplus_k \text{relative_median}' = v \oplus_k 1$. Thus, $\text{relative_median} \oplus_k \text{low} = v$ or $\text{relative_median} \oplus_k \text{low} = v \oplus_k 1$. In both cases $v \preceq_1 \text{low} \oplus_k \text{relative_median}$.

Lemma 6. *If a configuration \mathcal{C}_r is tight then so is \mathcal{C}_{r+1} .*

Proof. If p is faulty, its update of my_val_p and/or its write-registers do not affect the “tightness” of the configuration \mathcal{C}_{r+1} . Thus, the rest of the proof assumes that p is non-faulty.

First, notice that if $\mathcal{C}_r(\text{pass}_p(0, n-f)) \neq \emptyset$ then $\mathcal{C}_r(\text{pass}_p(0, n-f))$ contains a single value. This is because $\mathcal{C}_r(\text{pass}_p(0, n-f))$ contains all the values v that appear at least $n-f$ times in the registers read by p . If two values appear more than $n-f$ times they must be the same value (since $n > 6f$).

Consider p updating my_val_p . If p updates it in Line 08, then it must have passed the “if” in Line 07. Thus, $\mathcal{C}_r(\text{pass}_p(0, n-f)) \neq \emptyset$, which means that $\mathcal{C}_r(\text{pass}_p(0, n-f)) = \{v\}$. Thus, my_val_p is updated to $v \oplus_k 1$. From [Remark 3](#) it holds that $|H(\mathcal{C}_r, p, v, 0)| \geq n-f$, and from [Lemma 2](#) it holds that $|H(\mathcal{C}_r, v, 0)| \geq n-2f$. Thus, after p 's update of my_val_p to $v \oplus_k 1$ (and p 's writing my_val_p to all of p 's write-registers), $|H(\mathcal{C}_{r+1}, v, 1)| \geq n-2f$ holds. Thus, configuration \mathcal{C}_{r+1} is tight.

If p updates my_val_p in Line 10, then $\mathcal{C}_r(\text{pass}_p(1, n-f)) \neq \emptyset$. Denote by $v := \max\{\text{pass}_p(1, n-f)\}$. (in Line 10, my_val_p is updated to $v \oplus_k 1$). Notice

that $v \preceq_1 \text{my_val}_p$. In addition, $v \in \mathcal{C}_r(\text{pass}_p(1, n - f))$, thus (by [Remark 3](#)), $|H(\mathcal{C}_r, p, v, 1)| \geq n - f$. Therefore, after p 's update of my_val_p , $p \in H(\mathcal{C}_{r+1}, v, 1)$, which means that $|H(\mathcal{C}_{r+1}, v, 1)| \geq |H(\mathcal{C}_r, v, 1)|$ (because p may not be counted for in $H(\mathcal{C}_r, v, 1)$). By [Lemma 2](#), $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$, thus we have that $|H(\mathcal{C}_{r+1}, v, 1)| \geq n - 2f$, *i.e.*, configuration \mathcal{C}_{r+1} is tight.

We are left to consider updates of my_val_p in Line 14 and Line 15. Notice that since \mathcal{C}_r is tight, $|H(\mathcal{C}_r, p, v, 1)| \geq n - 2f$, for some v . Thus, $\mathcal{C}_r(\text{pass}_p(1, n - 2f)) \neq \emptyset$. Therefore, p passes the condition of Line 11, and p does not perform Line 15.

According to the lemma's assumption \mathcal{C}_r is tight around some value v , and by [Lemma 5](#) we have that $v \preceq_1 \text{relative_median} \oplus_k \text{low}$. That is, p updates my_val_p such that $p \in H(\mathcal{C}_{r+1}, v, 1)$. Since, $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$ and since p is the only node changing its my_val_p at \mathcal{C}_r , it holds that $|H(\mathcal{C}_{r+1}, v, 1)| \geq n - 2f$. That is, \mathcal{C}_{r+1} is tight.

The following lemmas assume all runs are en masse. In [Section 6](#) this assumption is removed (at the cost of reducing the fault tolerance to $n > 12f$).

Lemma 7. *Consider an en masse run, and a non-faulty node p performing an atomic step at configuration \mathcal{C}_r . Denote by S_i the set of non-faulty nodes that have performed an atomic step between \mathcal{C}_r and \mathcal{C}_{r+i} . Let m be the minimal m s.t. $|S_m| = n - 2f$, then each node $q \in S_m$ performed an atomic step exactly once between \mathcal{C}_r and \mathcal{C}_{r+m} .*

Proof. First, since we consider only fair runs, eventually S_i will contain all non-faulty nodes. Thus, m , as defined in the lemma is well defined. Assume by way of contradiction that some node q in S_m performed two atomic steps between \mathcal{C}_r and \mathcal{C}_{r+m} . In that case, there must be $n - 2f$ non-faulty nodes that perform atomic steps between q 's two atomic steps. Thus, all of these nodes must be in S_m , leading to the fact that $|S_{m-1}| \geq n - 2f$ which contradicts m being the minimal m s.t. $|S_m| = n - 2f$. Therefore, there is no such $q \in S_m$, and all nodes in S_m perform a single atomic step between \mathcal{C}_r and \mathcal{C}_{r+m} .

Lemma 8. *Let \mathcal{C}_{r_1} denote the first configuration of some round \mathcal{R} , and \mathcal{C}_{r_2} denote the last configuration of round $\mathcal{R} + 1$. With probability at least $\frac{1}{3^{n-2f}}$ configuration \mathcal{C}_{r_2} is tight.*

Proof. Consider non-faulty nodes performing atomic steps between \mathcal{C}_{r_1} and \mathcal{C}_{r_2} . If some configuration $\mathcal{C}_r, r_1 \leq r \leq r_2$ is tight, then - by using [Lemma 6](#) - every configuration after \mathcal{C}_r is tight. Thus, \mathcal{C}_{r_2} is tight. Therefore, our target is proving that with probability at least $\frac{1}{3^{n-2f}}$ some configuration \mathcal{C}_r is tight.

Let $\mathcal{C}_r, r_1 \leq r \leq r_2$ be some configuration, and let p be a non-faulty node performing an atomic step on \mathcal{C}_r . p performs exactly one of the following: Line 08, Line 10, Line 14 or Line 15. If p performs Line 08 or Line 10, then $\mathcal{C}_r(\text{pass}_p(1, n - f)) \neq \emptyset$. That is, for some value $v \in \mathcal{C}_r(\text{pass}_p(1, n - f))$ it holds that $|H(\mathcal{C}_r, p, v, 1)| \geq n - f$ which means that $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$ (see [Lemma 2](#)); that is, \mathcal{C}_r is tight

around v . Therefore, if any non-faulty node performs Line 08 or Line 10 during rounds $\mathcal{R}, \mathcal{R}+1$, then \mathcal{C}_{r_2} is tight.

The rest of the proof assumes no non-faulty node performs either Line 08 or Line 10 on any configuration $\mathcal{C}_r, r_1 \leq r \leq r_2$. Consider the first $n - 2f$ non-faulty nodes performing atomic steps in round \mathcal{R} . (By Lemma 7 these nodes perform exactly one atomic step, *i.e.*, the adversary cannot reschedule a node if “it does not like” the outcome of that node’s random coin). If they all perform only Line 15, then there is some probability that they all choose the same value of my_val , as they all choose from a set that contains “0”. Each node chooses from a set $pass(1, n - 3f) \cup \{0\}$, which contains at most 3 items. Thus, with probability at least $\frac{1}{3^{n-2f}}$ all non-faulty nodes choose the same value, leading to a tight configuration.

The proof continues under the assumption that some non-faulty node p performs Line 14 on some configuration \mathcal{C}_r during round \mathcal{R} . Using the notations \mathcal{S}_m of Lemma 7, there are $n - 2f$ non-faulty nodes that perform atomic steps between \mathcal{C}_r and $\mathcal{C}_{r'} = \mathcal{C}_{r+m}$. Notice that since all non-faulty nodes perform an atomic step during round $\mathcal{R} + 1$ then configuration $\mathcal{C}_{r'}$ is reached in round \mathcal{R} or in round $\mathcal{R} + 1$, and in any case $\mathcal{C}_{r'}$ is reached before \mathcal{C}_{r_2} .

Since p performs Line 14 it passed the condition of Line 11 and it holds that $\mathcal{C}_r(pass_p(1, n - 2f)) \neq \emptyset$. By Lemma 2 and Remark 3 $|H(\mathcal{C}_r, v, 1)| \geq n - 3f$ for some value v . According to Lemma 5, $v \preceq_1 \mathcal{C}_{r+1}(my_val_p)$; thus, $|H(\mathcal{C}_{r+1}, v, 1)| \geq n - 3f$.

The proof continues by showing that if $|H(\mathcal{C}_{r''}, v, 1)| \geq n - 3f$ for $r'', r + 1 \leq r'' \leq r'$, then with probability at least $\frac{1}{3}$ it holds that $|H(\mathcal{C}_{r''+1}, v, 1)| \geq n - 3f$; and if node q performing an atomic step on $\mathcal{C}_{r''}$ is non-faulty then also $q \in H(\mathcal{C}_{r''+1}, v, 1)$. Assume that $|H(\mathcal{C}_{r''}, v, 1)| \geq n - 3f$ and consider a node q performing an atomic step on $\mathcal{C}_{r''}$. If q is faulty then its action does not change the value of $H(\mathcal{C}_{r''+1}, v, 1)$, thus $|H(\mathcal{C}_{r''+1}, v, 1)| \geq n - 3f$. If q is non-faulty and it performs Line 15 then since $v \in \mathcal{C}_{r''}(pass_q(n - 3f, 1))$, with probability at least $\frac{1}{3}$, q selects v as its value of my_val_q (as it is selected from a set containing at most three items). On the other hand, if q performs Line 14, then since $|H(\mathcal{C}_{r''}, v, 1)| \geq n - 3f$ by Lemma 5 $q \in H(\mathcal{C}_{r''+1}, v, 1)$ and $|H(\mathcal{C}_{r''+1}, v, 1)| \geq n - 3f$. Thus, in either case with probability at least $\frac{1}{3}$ it holds that $q \in H(\mathcal{C}_{r''+1}, v, 1)$ and $|H(\mathcal{C}_{r''+1}, v, 1)| \geq n - 3f$.

Therefore, if at some configuration \mathcal{C}_r it holds that $|H(\mathcal{C}_r, v, 1)| \geq n - 3f$, then any non-faulty node q operating in a configuration $\mathcal{C}_{r''}, r'' \geq r$ will have $\mathcal{C}_{r''+1}(my_val_q) \in H(\mathcal{C}_{r''+1}, v, 1)$ (with probability $\geq \frac{1}{3}$). Therefore, once $n - 2f$ non-faulty nodes perform an atomic step, they are all in $H(\mathcal{C}_{r'}, v, 1)$ with probability at least $\frac{1}{3^{n-2f}}$. Thus, $\mathcal{C}_{r'}$ is tight with probability $\geq \frac{1}{3^{n-2f}}$.

From this point on, the discussion assumes that all configurations are tight. Therefore, Line 15 will never be executed.

Definition 13. Denote by $\mathcal{V}(\mathcal{C}_r)$ the set containing any value v of a non-faulty node p , such that p “helps” in the configuration \mathcal{C}_r being tight around v . Formally,

$$\mathcal{V}(\mathcal{C}_r) = \bigcup_{v \text{ s.t. } |H(\mathcal{C}_r, v, 1)| \geq n-2f} \{\mathcal{C}_r(\text{my_val}_p) | p \in H(\mathcal{C}_r, v, 1)\}.$$

Lemma 9. If $k \geq 6$, then $\mathcal{V}(\mathcal{C}_r)$ is exactly one of the following: $\{v\}$, $\{v, v \oplus_k 1\}$ or $\{v, v \oplus_k 1, v \oplus_k 2\}$, for some value v .

Proof. From Lemma 3 it follows that $|\mathcal{V}(\mathcal{C}_r)| \leq 3$. Moreover, if $k \geq 6$ then for any two values $v, v' \in \mathcal{V}(\mathcal{C}_r)$ it holds that $v \preceq_2 v'$ or $v' \preceq_2 v$, which is proved by way of contradiction. Assume that neither hold; notice that $v \in \mathcal{V}(\mathcal{C}_r)$ due to some value \bar{v} such that $\bar{v} \preceq_1 v$ and $|H(\mathcal{C}_r, \bar{v}, 1)| \geq n - 2f$; for similar reasons $v' \in \mathcal{V}(\mathcal{C}_r)$ due to $\bar{v}' \preceq_1 v'$. Thus, if neither $v \preceq_2 v'$ or $v' \preceq_2 v$, we have that $H(\mathcal{C}_r, \bar{v}, 1) \cap H(\mathcal{C}_r, \bar{v}', 1) = \emptyset$, leading to $|H(\mathcal{C}_r, \bar{v}, 1) \cup H(\mathcal{C}_r, \bar{v}', 1)| \geq 2(n - 2f) = 2n - 4f > n$. From the above discussion, if $|\mathcal{V}(\mathcal{C}_r)| = 3$, it must be of the form $\mathcal{V}(\mathcal{C}_r) = \{v, v \oplus_k 1, v \oplus_k 2\}$.

If $|\mathcal{V}(\mathcal{C}_r)| = 2$, then $\mathcal{V}(\mathcal{C}_r)$ can either be $\{v, v \oplus_k 1\}$ or $\{v, v \oplus_k 2\}$. In the second option, no non-faulty node has a value of $v \oplus_k 1$, that is, $H(\mathcal{C}_r, v, 1) \cap H(\mathcal{C}_r, v \oplus_k 2, 1) = \emptyset$. As before, we reach a contradiction from $|H(\mathcal{C}_r, v, 1) \cup H(\mathcal{C}_r, v \oplus_k 2, 1)| \geq 2(n - 2f) = 2n - 4f > n$.

Therefore, $\mathcal{V}(\mathcal{C}_r)$ is exactly one of the following: $\{v\}$, $\{v, v \oplus_k 1\}$ or $\{v, v \oplus_k 1, v \oplus_k 2\}$.

Lemma 9 leads to defining the “minimal” and “maximal” values of $\mathcal{V}(\mathcal{C}_r)$ in the following way: $\mathcal{V}_{\min}(\mathcal{C}_r) := \{v | v \in \mathcal{V}(\mathcal{C}_r) \ \& \ v \oplus_k -1 \notin \mathcal{V}(\mathcal{C}_r)\}$ and $\mathcal{V}_{\max}(\mathcal{C}_r) := \{v | v \in \mathcal{V}(\mathcal{C}_r) \ \& \ v \oplus_k 1 \notin \mathcal{V}(\mathcal{C}_r)\}$. By the above lemma both $\mathcal{V}_{\min}(\mathcal{C}_r)$ and $\mathcal{V}_{\max}(\mathcal{C}_r)$ are well defined (for $k \geq 6$).

Lemma 10. Let $k > 6$ and let $\mathcal{C}_r, \mathcal{C}_{r+1}$ be two consecutive configurations. Then, $\mathcal{V}_{\min}(\mathcal{C}_r) \preceq_3 \mathcal{V}_{\min}(\mathcal{C}_{r+1})$.

Proof. Let p be the node that performs an atomic step between \mathcal{C}_r and \mathcal{C}_{r+1} . If p is faulty, then its update of my_val_p does not affect the value of $\mathcal{V}_{\min}(\mathcal{C}_{r+1})$, and we have that $\mathcal{V}_{\min}(\mathcal{C}_r) = \mathcal{V}_{\min}(\mathcal{C}_{r+1})$, which means that $\mathcal{V}_{\min}(\mathcal{C}_r) \preceq_3 \mathcal{V}_{\min}(\mathcal{C}_{r+1})$.

The rest of the proof assumes p is non-faulty. p updates my_val_p due to Line 08, Line 10 or Line 14. If p performs Line 08 then my_val_p is updated to $v \oplus_k 1$, where $v = \max\{\text{pass}_p(0, n - f)\}$. By definition, $v \in \mathcal{V}(\mathcal{C}_r)$; thus, $\mathcal{V}_{\max}(\mathcal{C}_r) \preceq_1 \mathcal{V}_{\max}(\mathcal{C}_{r+1})$.

Similarly, if p performs Line 10, then my_val_p is updated to $v \oplus_k 1$, where $v = \max\{\text{pass}_p(1, n - f)\}$. Again, by definition $v \in \mathcal{V}(\mathcal{C}_r)$; thus, $\mathcal{V}_{\max}(\mathcal{C}_r) \preceq_1 \mathcal{V}_{\max}(\mathcal{C}_{r+1})$.

Consider p performs Line 14. By definition, $|H(\mathcal{C}_r, \mathcal{V}_{\min}(\mathcal{C}_r), 1)| \geq n - 3f$; thus, by Lemma 5 $\mathcal{V}_{\min}(\mathcal{C}_r) \preceq_1$ (*low \oplus_k relative_median*). Thus, $\mathcal{V}_{\max}(\mathcal{C}_r) \preceq_1 \mathcal{V}_{\max}(\mathcal{C}_{r+1})$.

In all 3 scenarios it was shown that $\mathcal{V}_{max}(\mathcal{C}_r) \preceq_1 \mathcal{V}_{max}(\mathcal{C}_{r+1})$. However, $|\mathcal{V}(\mathcal{C}_r)| \leq 3$ (see [Lemma 9](#)) and thus $\mathcal{V}_{min}(\mathcal{C}_r) \preceq_2 \mathcal{V}_{max}(\mathcal{C}_r)$. Therefore, $\mathcal{V}_{min}(\mathcal{C}_r) \preceq_3 \mathcal{V}_{max}(\mathcal{C}_{r+1})$. Since $\mathcal{V}_{min}(\mathcal{C}_{r+1}) \preceq_2 \mathcal{V}_{max}(\mathcal{C}_{r+1})$, we have that $\mathcal{V}_{min}(\mathcal{C}_r) \preceq_3 \mathcal{V}_{min}(\mathcal{C}_{r+1})$; as required.

Lemma 11. *If a non-faulty node p performs an atomic step between $\mathcal{C}_r, \mathcal{C}_{r+1}$ then $\mathcal{C}_{r+1}(my_val_p) \in \mathcal{V}(\mathcal{C}_{r+1})$.*

Proof. If p performs Line 08 or Line 10 then $\mathcal{C}_{r+1}(my_val_p) = v \oplus_k 1$ for some $v \in \mathcal{C}_r(pass_p(1, n - f))$. By [Remark 3](#) and [Lemma 2](#), $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$. Therefore, $p \in H(\mathcal{C}_{r+1}, v, 1)$ and $|H(\mathcal{C}_{r+1}, v, 1)| \geq n - 2f$, which means that $\mathcal{C}_{r+1}(my_val_p) \in \mathcal{V}(\mathcal{C}_{r+1})$.

Consider p performing Line 14. Since \mathcal{C}_r is tight there is some value v such that $|H(\mathcal{C}_r, v, 1)| \geq n - 2f$. By [Lemma 5](#) we have that $v \preceq_1 \mathcal{C}_{r+1}(my_val_p)$. Thus, $p \in H(\mathcal{C}_{r+1}, v, 1)$ and $|H(\mathcal{C}_{r+1}, v, 1)| \geq n - 2f$, which means that $\mathcal{C}_{r+1}(my_val_p) \in \mathcal{V}(\mathcal{C}_{r+1})$.

Lemma 12. *Starting from a tight configuration \mathcal{C}' , within 4 rounds there are two consecutive configurations $\mathcal{C}_r, \mathcal{C}_{r+1}$ for which $\mathcal{V}_{min}(\mathcal{C}_r) \neq \mathcal{V}_{min}(\mathcal{C}_{r+1})$.*

Proof. Consider configurations $\mathcal{C}' = \mathcal{C}_{r_1}, \mathcal{C}_{r_2}, \mathcal{C}_{r_3}, \mathcal{C}_{r_4}, \mathcal{C}_{r_5}$ such that $\mathcal{C}_{r_{i+1}}$ is one round after \mathcal{C}_{r_i} . Let $\mathcal{C}_{r'}$, $r_1 \leq r' < r_5$ be some configuration, and let p be a non-faulty node performing an atomic step on $\mathcal{C}_{r'}$. If $\mathcal{V}_{min}(\mathcal{C}_{r'}) \neq \mathcal{V}_{min}(\mathcal{C}_{r'+1})$, we are done. Otherwise, assume by way of contradiction that for all $r_1 \leq r' < r_5$ it holds that $\mathcal{V}_{min}(\mathcal{C}_{r'}) = \mathcal{V}_{min}(\mathcal{C}_{r'+1})$; denote $\mathcal{V} = \mathcal{V}_{min}(\mathcal{C}_{r_1})$. Therefore, by [Lemma 9](#) and [Lemma 11](#), $\mathcal{C}_{r'+1}(my_val_p) \in H(\mathcal{C}_{r'+1}, \mathcal{V}, 2)$. Since all non-faulty nodes have performed an atomic step between \mathcal{C}_{r_1} and \mathcal{C}_{r_2} , for any $\mathcal{C}_{r'}$, $r_2 \leq r' < r_5$ it holds that $|H(\mathcal{C}_{r'}, \mathcal{V}, 2)| = n - f$. We continue to consider only configurations $\mathcal{C}_{r'}$ such that $r_2 \leq r' < r_5$.

Notice that if p performs Line 08 or Line 10 then my_val_p is updated to be at least “+1” from \mathcal{V} . This is because only $\mathcal{V}, \mathcal{V} \oplus_k 1, \mathcal{V} \oplus_k 2$ may be in $\mathcal{C}_{r'}(pass_p(1, n - f))$ or $\mathcal{C}_{r'}(pass_p(0, n - f))$. Thus, taking the maximum of these sets and adding “1” produces a value that is at least “+1” from \mathcal{V} .

We divide the proof into two scenarios: 1) for some configuration $\mathcal{C}_{r'}$, $r_2 \leq r' \leq r_4$ it holds that $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| < \frac{n}{2} - f$; 2) for all $\mathcal{C}_{r'}$, $r_2 \leq r' \leq r_4$ it holds that $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| \geq \frac{n}{2} - f$. Consider the first case, and let $\mathcal{C}_{r'}$ be some configuration s.t. $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| < \frac{n}{2} - f$. Clearly, if p performs Line 08 or Line 10 then it updates my_val_p to be “greater” than \mathcal{V} . If p performs Line 14, then because values that are not $\mathcal{V} \oplus_k 1, \mathcal{V} \oplus_k 2$ can appear at most $\frac{n}{2}$ times, p must update my_val_p to be “greater” than \mathcal{V} . Therefore, if $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| < \frac{n}{2} - f$, then also $|H(\mathcal{C}_{r'+1}, \mathcal{V}, 0)| < \frac{n}{2} - f$. Moreover, $\mathcal{C}_{r'+1}(my_val_p) \in \{\mathcal{V} \oplus_k 1, \mathcal{V} \oplus_k 2\}$.

Thus, if for some configuration $\mathcal{C}_{r'}$, $r_2 \leq r' \leq r_4$ it holds that $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| < \frac{n}{2} - f$, then starting from \mathcal{C}_{r_5} (at least one round after $\mathcal{C}_{r'}$), no non-faulty node has my_val equal to \mathcal{V} , which means that $\mathcal{V}_{min}(\mathcal{C}_{r_5}) \neq \mathcal{V} = \mathcal{V}_{min}(\mathcal{C}_{r_1})$.

We continue under the assumption that $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| \geq \frac{n}{2} - f$, for all $r_2 \leq r' \leq r_4$. Recall that all non-faulty nodes have values from the set $\{\mathcal{V}, \mathcal{V} \oplus_k 1, \mathcal{V} \oplus_k 2\}$. Thus, $|H(\mathcal{C}_{r'}, \mathcal{V} \oplus_k 1, 1)| \leq \frac{n}{2}$. Therefore, if p passes Line 08 or Line 10 then $\mathcal{C}_{r'}(\text{pass}_p(0, n - f))$ and $\mathcal{C}_{r'}(\text{pass}_p(1, n - f))$ do not contain $\mathcal{V} \oplus_k 1, \mathcal{V} \oplus_k 2$; which means they may contain $\mathcal{V} \oplus_k -1$ or \mathcal{V} . Thus, p updates my_val_p to \mathcal{V} or $\mathcal{V} \oplus_k 1$. On the other hand, if p performs Line 14 it updates my_val_p to be either \mathcal{V} or $\mathcal{V} \oplus_k 1$ (recall that $\mathcal{V}_{\min}(\mathcal{C}_{r'}) = \mathcal{V}$ which means that $|H(\mathcal{C}_{r'}, \mathcal{V}, 1)| \geq n - 2f > \frac{n}{2}$). Thus, in all cases, p updates my_val_p to be either \mathcal{V} or $\mathcal{V} \oplus_k 1$. Therefore, $|H(\mathcal{C}_{r_3}, \mathcal{V}, 1)| = n - f$, and $|H(\mathcal{C}_{r'}, \mathcal{V}, 1)| = n - f$ for all $r_3 \leq r' \leq r_4$.

Thus, any non-faulty node performing an atomic step on configuration $\mathcal{C}_{r'}, r_3 \leq r' \leq r_4$, either passes the condition of Line 08 or Line 10. In both cases, it updates my_val_p to be “greater” than \mathcal{V} . Thus, starting from \mathcal{C}_{r_4} it holds that $|H(\mathcal{C}_{r'}, \mathcal{V}, 0)| < \frac{n}{2} - f$. And we are back to the previous case, in which we have shown that \mathcal{V}_{\min} must change within 1 round. Thus, for some configuration $\mathcal{C}_{r'}, r_4 \leq r' < r_5$ it holds that $\mathcal{V}_{\min}(\mathcal{C}_{r'}) \neq \mathcal{V}$. In other words, within 4 rounds there is some configuration \mathcal{C}_r such that $\mathcal{V}_{\min}(\mathcal{C}_r) \neq \mathcal{V}_{\min}(\mathcal{C}_{r+1})$.

Following is the main result of the paper, which is shown to be true assuming that the runs are en masse. In the following section en masse runs are constructed from fair runs. Thus, the theorem can be updated to only require that the run is fair.

Theorem 1. *ASYNC-CLOCK solves the 5-clock-synchronization problem within expected $O(3^{n-2f})$ rounds, for any en masse run and wrap-around value greater than 6 (i.e., $k > 6$).*

Proof. Define the clock-function \mathcal{F} executed by non-faulty node p at configuration \mathcal{C}_r to be the value of $\mathcal{C}_{r+1}(\text{my_val}_p)$ as updated by ASYNC-CLOCK when executed as an atomic step. Combining Lemma 9, Lemma 10 and Lemma 11 shows that a tight configuration is 5-well-defined with respect to \mathcal{F} ; where $\mathcal{V}_{\min}(\mathcal{C}_r)$ a defined value at \mathcal{C}_r . In addition, these lemmas show that any fair run \mathcal{T} consisting of only tight configurations is 5-well-defined. By Lemma 12, run \mathcal{T} is also 5-clock-synchronized.

Given any en masse run \mathcal{T}' and any initial configuration \mathcal{C}_0 , Lemma 8 states that with probability $\geq \frac{1}{3^{n-2f}}$ there is some configuration $\mathcal{C}_r \in \mathcal{T}'$ (within two rounds from \mathcal{C}_0) that is tight. By Lemma 6 every configuration after \mathcal{C}_r is also tight. Thus, every fair run \mathcal{T}' has a suffix \mathcal{T} that consists of only tight configurations; and this suffix is reached within $O(3^{n-2f})$ rounds in expectation. From the above paragraph, \mathcal{T} is 5-clock-synchronized.

Thus, ASYNC-CLOCK solves the 5-clock-synchronization problem.

6 Ensuring En Masse Runs

Our goal is to ensure that if a non-faulty node p performs a step, at least $n - 2f$ non-faulty nodes have performed a step since p 's last step. That is, given an

algorithm \mathcal{A} we want to ensure that if some non-faulty node performs two steps of \mathcal{A} then there are at least $n - 2f$ different non-faulty nodes that also perform steps of \mathcal{A} . To ensure this, we present an algorithm ENMASSE that ensures that a specific action, denoted “act”, is executed twice by the same non-faulty node p only if there are at least $n - 4f$ other non-faulty nodes that have also executed “act”. By setting “act” to execute an atomic step of \mathcal{A} , we achieve the required goal. *I.e.*, ASYNC-CLOCK will be executed entirely every time “act” appears in ENMASSE.

As the algorithm we present ensures only $n - 4f$ nodes execute “act” in between two “acts” of every non-faulty node, we must reduce the *Byzantine* tolerance by half ($n > 12f$) to use ENMASSE as a subcomponent of ASYNC-CLOCK. That is, ASYNC-CLOCK requires a threshold of $\frac{2}{3}n$ non-faulty nodes ($n - 2f$ threshold for $n > 6f$); ENMASSE ensures a threshold of $n - 4f$. Therefore, by reducing the fault tolerance to $n > 12f$ we ensure that $n - 4f > \frac{2}{3}n$, as required by ASYNC-CLOCK.

Our solution borrows many ideas from [13]. Due to our model’s atomicity assumptions, each node can read all registers and write to all registers in a single atomic step. Thus, the problems that [13] encounters do not exist in the current paper at all. However, in the current model there are additional faults (*Byzantine* and self-stabilizing) which do not exist in [13]. Interestingly, the same ideas used in [13] can be adapted to the self-stabilizing *Byzantine* tolerant setting.

For each node p , there is a set of labels $Labels_p$ associated with p . In addition, each node p has a variable $label_p$ from the set $Labels_p$; Also, p has an ordering vector $order_p$, of length $|Labels_p|$, which induces an order on the labels in $Labels_p$. Lastly, each node p has a time-stamp $time_p$, which is a vector of n entries, consisting of a single label $time_p[q] \in Labels_q$ for each node q .

Definition 14. A label b is of **type** p if $b \in Labels_p$.

Definition 15. Two labels b, c of type p are compared according to $order_p$, where $b <_p c$ if b appears before c in the vector $order_p$. The inequalities $\leq_p, >_p, \geq_p, =_p$ are similarly defined.

Definition 16. Given two time-stamps $time_p, time_q$, and a set of nodes I , we say that $time_p >_I time_q$ if $p, q \in I$ and for every entry $i \in I$, $time_p[i] \geq_i time_q[i]$, $time_p[q] =_q time_q[q]$ and $time_p[p] >_p time_q[p]$.

To simplify notations, when it is clear from the context, we write $p >_I q$ instead of $time_p >_I time_q$. That is, when comparing nodes (according to $>_I$), we actually compare the nodes’ time stamps.

Definition 17. A set I of nodes is **comparable** if for any $p, q \in I$ either $p >_I q$ or $q >_I p$.

Lemma 13. If I is a comparable set, and $p, q, w \in I$, and $p >_I q, q >_I w$ then $p >_I w$.

Proof. Since I is comparable, either $p >_I w$ or $w >_I p$. Suppose by way of contradiction that $w >_I p$, thus $time_p[w] <_w time_w[w]$. However, since $p >_I q$ we have that $time_p[w] \geq_w time_q[w]$, and since $q >_I w$ we have that $time_q[w] =_w time_w[w]$. Thus, $time_p[w] \geq_w time_w[w]$, contradicting $time_p[w] <_w time_w[w]$. Therefore, it is not true that $w >_I p$, leaving only one other option: $p >_I w$.

Lemma 14. *Let I, I' be comparable sets, then $I \cap I'$ is a comparable set. Moreover, for $p, q \in I \cap I'$, $p <_{I \cap I'} q$ iff $p <_I q$.*

Proof. Let I, I' be comparable sets, and let $p, q \in I \cap I'$. Since $p, q \in I$ and I is comparable, either $p <_I q$ or $q <_I p$, similarly, either $p <_{I'} q$ or $q <_{I'} p$. Suppose by way of contradiction that $p <_I q$ and $q <_{I'} p$. Due to $p <_I q$ it holds that $time_q[q] >_q time_p[q]$ and due to $q <_{I'} p$ it holds that $time_q[q] =_q time_p[q]$; leading to a contradiction. Thus, either $p <_I q$ and $p <_{I'} q$ or $p >_I q$ and $p >_{I'} q$.

Assume that $p <_I q$ and $p <_{I'} q$. Therefore, for all $i \in I \cup I'$ it holds that $time_p[i] \leq_i time_q[i]$, $time_p[p] = time_q[p]$ and $time_p[q] < time_q[q]$. Thus, for all $i \in I \cap I'$ it holds that $time_p[i] \leq_i time_q[i]$, and by definition we have that $p <_{I \cap I'} q$. Similarly, if $p >_I q$ and $p >_{I'} q$ then $p >_{I \cap I'} q$.

It was shown that only two options exist: 1) $p <_I q$ and $p <_{I'} q$, 2) $p >_I q$ and $p >_{I'} q$. If option 1 occurs, then $p <_{I \cap I'} q$; if option 2 occurs then $p >_{I \cap I'} q$. Thus, any $p, q \in I \cap I'$ either $p >_{I \cap I'} q$ or $p <_{I \cap I'} q$ holds. *i.e.*, $I \cap I'$ is comparable. Moreover, we have shown that $p <_{I \cap I'} q$ iff $p <_I q$, as required.

Remark 5. Notice that proof of the lemma above also implies that $p <_I q$ iff $p <_{I'} q$.

Notice that a comparable set I induces a total order among the elements in I , therefore we can refer to the index of an element in I .

Definition 18. *A node $p \in I$ is said to be the k th highest (in I) if $|\{q \in I | q >_I p\}| = k - 1$. Let $I_{\#}(p) = k$ if $p \in I$ is the k th highest in I .*

The 1st highest in I is the node that is larger than all other nodes. The 2nd highest node in I is the node that has only one node larger than it; (and so on).

Informally, we wish to show that given two intersecting comparable sets I, I' , if a node p is the i th highest item in I , it is at most $i + \ell$ highest in I' ; where ℓ changes according to I, I' . The following lemma formally bounds the difference between $I_{\#}(p)$ and $I'_{\#}(p)$.

Lemma 15. *Let I, I' be comparable sets, and denote $\ell = |I| - |I \cap I'|$. If $p \in I \cap I'$ then $I_{\#}(p) \leq I'_{\#}(p) + \ell$.*

Proof. Let $p \in I \cap I'$. By definition $I_{\#}(p) = |\{q \in I | q >_I p\}| + 1$ and $I'_{\#}(p) = |\{q \in I' | q >_{I'} p\}| + 1$. Therefore, it is enough to show that $|\{q \in I | q >_I p\}| \leq |\{q \in I' | q >_{I'} p\}| + \ell$. Consider the set $A = \{q \in I \cap I' | q >_{I \cap I'} p\}$, be [Lemma 14](#), it holds that $A \subseteq \{q \in I | q >_I p\}$ and $A \subseteq \{q \in I' | q >_{I'} p\}$. Clearly, $\{q \in I | q >_I p\}$

```

01: do forever:

    /* read all registers and initialize structures */
02: for each node  $p$ , read  $time_p$  and  $order_p$ ;
03: set  $\mathcal{I} := \emptyset$ ;
04: for each set  $W \subseteq \mathcal{P}$  s.t.  $|W| \geq n - f$  and  $q \in W$ :
05:   construct  $I := \{time_p \mid p \in W\}$ ;
06:   if  $W$  is comparable then  $\mathcal{I} := \mathcal{I} \cup \{I\}$ ;

    /* decide whether to execute "update" and whether to execute "act" */
07:   if for some  $I \in \mathcal{I}$ , it holds that  $I_{\#}(q) \geq n - 3f$  then
08:     update  $time_q, order_q$  and "act";
09:   if  $\mathcal{I} = \emptyset$  then update  $time_q, order_q$ ;
10:   write  $time_q$  and  $order_q$ ;
11: od;

```

Updating $time_q$ is done by setting $time_q[p] = label_p$, for every $p \in \mathcal{P}$.
Updating $order_q$ consists of changing the order induced by $order_q$ such that $label_q$ is first and for other labels the order is preserved.

Fig. 2. A self-stabilizing *Byzantine* tolerant algorithm ensuring en masse runs.

$p\}$ – A contains only items in $I - I \cap I'$. Thus, $|\{q \in I \mid q >_I p\} - A| \leq |I| - |I \cap I'|$ and since $A \subseteq \{q \in I \mid q >_I p\}$ it holds that $|\{q \in I \mid q >_I p\}| - |A| \leq |I| - |I \cap I'|$. That is, $|\{q \in I \mid q >_I p\}| \leq |A| + \ell$. Since, $A \subseteq \{q \in I' \mid q >_{I'} p\}$ it holds that $|A| \leq |\{q \in I' \mid q >_{I'} p\}|$. Thus, $|\{q \in I \mid q >_I p\}| \leq |\{q \in I' \mid q >_{I'} p\}| + \ell$, as required.

Corollary 1. *Let I, I' be comparable sets, and denote $\ell' = \max\{|I|, |I'|\} - |I \cap I'|$. If $p \in I \cap I'$ then $I'_{\#}(p) - \ell' \leq I_{\#}(p) \leq I'_{\#}(p) + \ell'$.*

6.1 Algorithm EnMasse

This section proves general properties of comparable sets. It discusses “static” sets, that do not change over time. The following algorithm considers comparable sets that change from step to step. However, during each atomic step, the comparable sets that are considered do not change, and the claims from the previous section hold. That is, when reasoning about the progress of the algorithm, the comparable sets that are considered are all “static”.

In the following algorithm, instead of storing both $label_p$ and $time_p$, each node stores just $time_p$ and the value of $label_p$ is the entry $time_p[p]$. In addition, during each atomic step, the entire algorithm is executed, *i.e.*, a node reads all time stamps and all order vectors of other nodes, and can update its own time stamp during an atomic step.

When a node q performs an update, it changes the value of $time_q$ and $order_q$ in the following way: a) $order_q$ is updated such that $time_q[q]$ is larger than any

other label in $Labels_q$. b) $time_q[p]$ is set to be $time_p[p]$, for all p . Notice that the new $order_q$ does not affect the relative order of labels in $Labels_q$ that are not $time_q[q]$. That is, if $l_1, l_2 \neq time_q[q]$ and $l_1 \leq_q l_2$ before the change of $order_q$, it holds that $l_1 \leq_q l_2$ also after updating the $order_q$.

Intuitively, the idea of ENMASSE is to increase the time stamp of a node q only if q sees that most of the other nodes are ahead of q . When the time stamp is increased, q also performs “act”. This leads to the following dynamics: a) If q has performed an “act” twice, *i.e.*, updated its time stamp twice, then after the first update, q is ahead of all other nodes. b) However, since q is ahead of all non-faulty nodes, if q updates its time stamp again it must mean that many nodes have updated their time stamps after q ’s first update. *i.e.*, between two “act” of q many other nodes have performed “act” as well.

We continue with an overview of the proof. First, consider the set of non-faulty nodes, and consider the set of time stamps of these nodes. The proof shows that if this set is comparable for some configuration \mathcal{C}_r then it is comparable for any configuration $\mathcal{C}_{r'}$ where $r' > r$. Second, we consider an arbitrary starting state, and consider the set Y_r containing non-faulty nodes that have updated their time stamp by the end of round r . It is shown that if $|Y_r| \geq n - 2f$ then $|Y_{r+1}| \geq n - f$. Moreover, if $|Y_r| < n - 2f$ then $|Y_{r+1}| \geq |Y_r| + 1$. Thus, we conclude that within $O(n)$ rounds all non-faulty nodes have performed an update.

Once all non-faulty nodes have performed an update since the starting state, it holds that the set of all non-faulty nodes’ time stamps is comparable. Thus, during every round at least $2f$ nodes perform an update (as they see themselves in the lower $3f$ part of the comparable set). This ensures that within $\frac{n}{2f}$ rounds some node will perform “act” twice. That is, there is no deadlock in the ENMASSE algorithm. To conclude the proof, it is shown that when the set of all non-faulty nodes values is comparable and some non-faulty node performs “act” twice, it must be that another $n - 4f$ non-faulty nodes have performed “act” in between.

Definition 19. *Let Z be a set of non-faulty nodes, and consider an atomic step on configuration \mathcal{C}_r . Denote by $\mathcal{TS}_{\mathcal{C}_r}(Z)$ the set of time-stamps of nodes in Z , as they are at the beginning of the atomic step. When the configuration \mathcal{C}_r is clear from the context, we simply say “ Z is comparable”, instead of “ $\mathcal{TS}_{\mathcal{C}_r}(Z)$ is comparable”.*

Lemma 16. *Let Z be a set of non-faulty nodes and consider any atomic step on configuration \mathcal{C}_r . If $\mathcal{TS}_{\mathcal{C}_r}(Z)$ is comparable, then $\mathcal{TS}_{\mathcal{C}_{r'}}(Z)$ is comparable for any $r' \geq r$.*

Proof. First, notice that whether $\mathcal{TS}_{\mathcal{C}_{r'}}$ is comparable or not depends only on the values of nodes in Z and is not affected by nodes not in Z . Therefore, only changes incurred by nodes in Z matter. Consider the first node $p \in Z$ to perform an update at some configuration $\mathcal{C}_{r''}$, $r'' \geq r$. Thus, p sets $time_p[q] = time_q[q]$

for all nodes $q \in Z$ and also p ensures that $time_p[p] >_p time_q[p]$ for all nodes $q \in Z, q \neq p$. Thus, $p >_Z q$ for all nodes $q \in Z, q \neq p$.

Consider two nodes $q_1, q_2 \neq p$ in Z . p 's update does not change the value of $time_{q_1}[q]$ and $time_{q_2}[q]$ for all $q \in Z, q \neq p$. What about the relative order of $time_{q_1}[p]$ and $time_{q_2}[p]$? W.l.o.g. $q_1 <_Z q_2$ before p 's update. According to the way p changes $order_p$, after p 's update $time_{q_1}[p] \leq_p time_{q_2}[p]$. Therefore, $q_1 <_Z q_2$.

Thus, for any pair of nodes $q_1, q_2 \in Z$ either $q_1 <_Z q_2$ or $q_1 <_Z q_2$ after p 's update. Repeating the above line of proof for any node in Z inductively proves that $\mathcal{TS}_{\mathcal{C}_r}(Z)$ is comparable.

Consider the system starts in an arbitrary state. Denote by U_r the set of non-faulty nodes that have not performed any ‘‘update’’ by the end of round r , and by Y_r the set of non-faulty nodes that have performed ‘‘update’’ by the end of round r .

Lemma 17. *The set Y_r is comparable during the last configuration of round r .*

Proof. Consider the order at which non-faulty nodes performed updates by the end of round r : let p_1 denote the first node to perform an update, p_2 the second, \dots, p_m be the m th (and last) non-faulty node to perform an update. A node may appear more than once in that order, for example, if some node was the 2nd and 5th to perform an update, $p_2 = p_5$. Denote by A_i the set containing all non-faulty nodes in $\{p_1, \dots, p_i\}$, where p_i is the i th node performing an update. Clearly, $A_m = Y_r$, and it is left to show that the set of time-stamps of nodes from A_m is comparable. Notice that if $p_{i+1} \in A_i$ then $A_i = A_{i+1}$, that is, if the node performing the ‘‘next’’ update has already performed an update, $A_i = A_{i+1}$.

We show something stronger: for all $0 \leq i \leq m$ let \mathcal{C}_i be the configuration after p_i performs an atomic step, then $\mathcal{TS}_{\mathcal{C}_i}(A_i)$ is comparable. The proof is by induction on i . For $i = 0, 1$, clearly A_i is comparable. We are left to show that if A_i is comparable so is A_{i+1} .

If $p_{i+1} \in A_i$ (i.e., p_{i+1} already performed an update) then by [Lemma 16](#) it holds that A_{i+1} is comparable. Consider p_{i+1} such that $p_{i+1} \notin A_i$. According to the way p_{i+1} updates its registers, for any two nodes q_1, q_2 , if $q_1 <_{A_i} q_2$ then also $q_1 <_{A_{i+1}} q_2$. Moreover, for any node $q \in A_i$ it holds that $q <_{A_i} p_{i+1}$. Thus, for any pair of nodes $p, q \in A_{i+1}$ either $p <_{A_{i+1}} q$ or $q <_{A_{i+1}} p$.

Thus, A_{i+1} is comparable. To complete the proof, recall that $A_m = Y_r$.

Lemma 18. *Let q be a node in U_r . Whenever q performs an atomic step before the end of round r , it holds that $\mathcal{I} \neq \emptyset$, and for all $I \in \mathcal{I}$ it holds that $I_{\#}(q) < n - 3f$.*

Proof. If $\mathcal{I} = \emptyset$ during q 's atomic step, then q will perform an update, and $q \notin U_r$. Similarly, if for some $I \in \mathcal{I}$ it holds that $I_{\#}(q) \geq n - 3f$ then q will also perform an update. Since $q \in U_r$, by definition, q does not perform an update until the end of round r .

Lemma 19. *Let q be a non-faulty node and consider q 's atomic step during round r'' , $r < r'' \leq r'$. For any comparable set $I \in \mathcal{I}$ that q considers in Line 07, and for any $p' \in U_{r'}$, $p'' \in Y_r$: if $p', p'' \in I$ then $I_{\#}(p'') < I_{\#}(p')$.*

Proof. Since p', p'' are both in I , either $p' <_I p''$ or $p'' <_I p'$. Since $p' \in U_{r'}$ it has not yet performed an update, while p'' has performed an update before the end of round r . Thus, $time_{p''}[p'] = time_{p'}[p']$ and therefore it cannot be that $p'' <_I p'$, leaving us with $p' <_I p''$. Since I is totally ordered, any node w such that $p'' <_I w$ also holds that $p' <_I w$. Therefore, there are more nodes in I that are larger than p' than there are nodes that are larger than p'' . *i.e.*, $I_{\#}(p'') < I_{\#}(p')$.

Lemma 20. *If $|Y_r| \geq n - 2f$ then $|Y_{r+1}| \geq n - f$.*

Proof. If $U_{r+1} = \emptyset$ we are done. Otherwise, assume by way of contradiction that $q \in U_{r+1}$. By Lemma 18 during q 's atomic steps in round $r+1$ it holds that $\mathcal{I} \neq \emptyset$ and for all $I \in \mathcal{I}$ we have that $I_{\#}(q) < n - 3f$. Consider such an $I \in \mathcal{I}$; since $|Y_r| \geq n - 2f$, it holds that $|I \cap Y_r| \geq n - 3f$. That is, I contains at least $n - 3f$ nodes that have performed an update. Thus, by Lemma 19, $I_{\#}(q) \geq n - 3f$ which contradicts the fact that $I_{\#}(q) < n - 3f$. Thus, $q \notin U_{r+1}$ and $U_{r+1} = \emptyset$.

Lemma 21. *If $|Y_r| < n - 2f$ then $|Y_{r+1}| \geq |Y_r| + 1$.*

Proof. Assume by way of contradiction that $|Y_{r+1}| < |Y_r| + 1$. Therefore, $|Y_{r+1}| \leq |Y_r| < n - 2f$ and $|U_{r+1}| \geq f + 1$. Thus, for any set I containing at least $n - f$ nodes, it holds that $I \cap U_{r+1} \neq \emptyset$.

Let $q \in U_{r+1}$, by Lemma 18 during q 's atomic steps in round $r+1$ it holds that $\mathcal{I} \neq \emptyset$ and for all $I \in \mathcal{I}$ we have that $I_{\#}(q) < n - 3f$. By Lemma 19, for any node $q' \in U_{r+1}$ and any node $p' \in Y_r$ it holds that if $q', p' \in I$ then $I_{\#}(p') < I_{\#}(q')$. Therefore, for any node $p \in I \cap U_{r+1}$ it holds that $I_{\#}(p) > |I \cap Y_r|$. As there are $|I \cap U_{r+1}| > 0$ nodes from U_{r+1} in I , there is a node $p \in I \cap U_{r+1}$ such that $I_{\#}(p) \geq |I \cap Y_r| + |I \cap U_{r+1}|$.

Notice that $Y_r \subseteq Y_{r+1}$, and since $|Y_{r+1}| \leq |Y_r|$ it holds that $Y_r = Y_{r+1}$. Moreover, $U_{r+1} \cap Y_{r+1} = \emptyset$ and $|U_{r+1} \cup Y_{r+1}| = n - f$. Thus, $|I \cap Y_r| + |I \cap U_{r+1}| = |I \cap (Y_r \cup U_{r+1})| \geq n - 2f$. That is, there is a node $p \in I \cap U_{r+1}$ such that $I_{\#}(p) \geq n - 2f$. *i.e.*, there are at most $3f - 1$ nodes p' that have the following property: $time_{p'}[p'] <_{p'} time_p[p']$. Since p does not perform an update, this property can change only if p' performs an update, which will reduce the number of nodes such that $time_{p'}[p'] <_{p'} time_p[p']$. Therefore, throughout round $r + 1$, if p considers a comparable set $I \in \mathcal{I}$, it will always have $I_{\#}(p) \geq n - 3f$.

Consider an atomic step by p during round $r+1$, by Lemma 18 for any $I' \in \mathcal{I}$ that p considers in Line 07, $I'_{\#}(p) < n - 3f$, which contradicts the above fact that $I_{\#}(p) \geq n - 3f$. Thus, we conclude that $|Y_{r+1}| > |Y_r|$; as required.

Corollary 2. *For any round $r \geq n - 2f + 2$, it holds that $U_r = \emptyset$ and $|Y_r| = n - f$.*

Proof. Apply [Lemma 21](#) during the first $n - 2f$ rounds, then apply [Lemma 20](#) for round $n - 2f + 1$.

Lemma 22. *For any round $r > n - 2f + 2$, any non-faulty node q considers $Y_r \in \mathcal{I}$ during atomic steps of round r .*

Proof. By [Corollary 2](#), $|Y_{r-1}| = n - f$ and $U_{r-1} = \emptyset$, leading to $q \in Y_{r-1}$. By [Lemma 17](#) and [Lemma 16](#) Y_r is comparable, and is viewed as comparable by q during any atomic step of round r . Thus, when q performs an atomic step during round r , q adds Y_r to \mathcal{I} at Line 06.

Corollary 3. *For any round $r > n - 2f + 2$, no non-faulty node passes the condition of Line 09.*

Proof. By [Lemma 22](#), a non-faulty node q performing an atomic step during round $r > n - 2f + 2$ has $\mathcal{I} \neq \emptyset$. Thus, the condition of Line 09 does not hold.

Lemma 23. *Let r be any round, $r > n - 2f + 2$. During round r at least $2f$ non-faulty nodes perform update.*

Proof. Let $r > n - 2f + 2$ be any round. Consider the set $W = \{time_q | q \in Y_r\}$ before the first atomic step of round r . Denote by $Z = \{p \in W | W_{\#}(p) \geq n - 3f\}$, that is, Z contains all nodes that are at most $n - 3f$ highest in W . Since $|W| = n - f$ it holds that $|Z| = 2f$.

For each node $q \in Z$, consider q 's first atomic step in round r . By the proof of [Lemma 17](#), since q did not perform an update since the beginning of round r , when it performs its first atomic step, it holds that $W_{\#}(q) \geq n - 3f$ and q will pass the condition in Line 07 and perform an update (and “act”) in Line 08. This holds for all $q \in Z$, that is, for at least $2f$ nodes.

Lemma 24. *Starting from round $n - 2f + 3$, every non-faulty node p performs an update at least once every $\frac{n}{2f}$ rounds.*

Proof. By the proof of [Lemma 23](#), every round the lowest $2f$ nodes perform an update. Thus, if p does not perform an update during round r , there are at least $2f$ nodes higher than it. Consider round $r + i$, if p does not perform an update there are $2f \cdot i$ nodes higher than p . Therefore, after at most $\frac{n}{2f}$ rounds p will perform an update.

Lemma 25. *Consider a non-faulty node p performing an update twice, then there are at least $n - 4f$ other nodes that have performed update in between.*

Proof. Consider the comparable set Y_r after p 's first update. By the way p does an update, $Y_{r\#}(p) = 1$. When p performs its second update, it has some comparable set $I \in \mathcal{I}$, such that $I_{\#}(p) \geq n - 3f$. Therefore, at least $n - 4f$ non-faulty nodes have become larger than p . Thus, they all must have performed an update.

Theorem 2. *Starting from round $n - 2f + 3$, between any non-faulty node’s two consecutive “act”s, there are $n - 4f$ non-faulty nodes that perform “act”. Moreover, every non-faulty node performs an “act” at least once every $\frac{n}{2f}$ rounds.*

Proof. By [Corollary 3](#), non-faulty nodes perform update only if they also perform an “act”. By [Lemma 25](#), between a non-faulty node’s two consecutive updates there are $n - 4f$ non-faulty nodes that perform an update. By [Lemma 24](#) every non-faulty node performs an update at least once every $\frac{n}{2f}$ rounds. Combining these two claims yields the required result.

[Theorem 2](#) states that using ENMASSE one can ensure that nodes executing ASYNC-CLOCK will have the following properties: 1) every non-faulty node p executes an atomic step of ASYNC-CLOCK once every $\frac{n}{2f}$ rounds; 2) if non-faulty p executes 2 atomic steps of ASYNC-CLOCK, then at least $n - 4f$ non-faulty nodes execute atomic steps of ASYNC-CLOCK in between. By setting $n > 12f$, these properties ensure that a fair run \mathcal{T} is an en-masse run \mathcal{T}' with respect to ASYNC-CLOCK, s.t. each round of \mathcal{T}' consists of at most $\frac{n}{2f}$ rounds of \mathcal{T} .

7 Discussion

7.1 Solving the 1-Clock-Synchronization Problem

First, the 5-clock-synchronization problem was solved using ASYNC-CLOCK while assuming en masse runs. Second, the assumption of en masse runs was removed in [Section 6](#). In this subsection we complete the paper’s result by showing how to transform a 5-clock-synchronization algorithm to a 1-clock-synchronization algorithm.

Given any algorithm \mathcal{A} that solves the ℓ -clock-synchronization problem, one can construct an algorithm \mathcal{A}' that solves the 1-clock-synchronization problem. Denote by $k_{\mathcal{A}'}$ the desired wraparound value of \mathcal{A}' , and let $k_{\mathcal{A}} = k_{\mathcal{A}'} \cdot \ell$ be the wraparound value for \mathcal{A} .

The construction is simple: each time \mathcal{A}' is executed, it runs \mathcal{A} and returns the clock value of \mathcal{A} divided by ℓ (that is, $\lfloor \frac{\mathcal{F}_{\mathcal{A}}}{\ell} \rfloor$). The intuition behind this construction is straightforward: \mathcal{A} solves the ℓ -clock-synchronization problem, thus, the values it returns are at most ℓ apart. Therefore, the values that \mathcal{A}' returns are at most 1 apart from each other.

7.2 Future Work

The current paper has a few drawbacks, each of which is interesting to resolve.

First, is it possible to reduce the atomicity requirements; that is, can an atomic step be defined as a single read or a single write (and not as “read all registers and write all registers”)?

Second, can the current algorithm be transported into a message passing model?

Third, can different coin-flipping algorithms that operate in the asynchronous setting (*i.e.*, [6]) be used to reduce the exponential convergence time to something more reasonable? Perhaps even expected constant time?

Fourth, can the ratio between *Byzantine* and non-*Byzantine* nodes be reduced? *I.e.*, can $n > 3f$ be achieved?

Fifth, can the problem of asynchronous *Byzantine* agreement be reduced to the problem of clock synchronization presented in the current work? (This will show that the expected exponential convergence time is as good as is currently known).

Lastly, the building block ENMASSE is interesting by itself. It would be interesting to find a polynomial solution to ENMASSE.

8 Acknowledgements

Michael Ben-Or is the incumbent of the Jean and Helena Alfassa Chair in Computer Science, and he was supported in part by the Israeli Science Foundation (ISF) research grant. Danny Dolev is Incumbent of the Berthold Badler Chair in Computer Science. Danny Dolev was supported in part by the Israeli Science Foundation (ISF) Grant number 0397373.

References

1. Y. Afek and S. Dolev. Local stabilizer. In *Proc. of the 5th Israeli Symposium on Theory of Computing Systems (ISTCS97)*, Bar-Ilan, Israel, Jun 1997.
2. A. Arora, S. Dolev, and M.G. Gouda. Maintaining digital clocks in step. *Parallel Processing Letters*, 1:11–18, 1991.
3. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.
4. Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.
5. Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC '83*, pages 27–30, New York, NY, USA, 1983.
6. Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC '93*, pages 42–51, New York, NY, USA, 1993. ACM.
7. J. M. Couvreur, N. Francez, and M. Gouda. Asynchronous unison. *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 486–493, 1992.
8. W. Dijkstra. Self-stabilization in spite of distributed control. *Commun. of the ACM*, 17:643–644, 1974.
9. D. Dolev and E. N. Hoch. Byzantine self-stabilizing pulse in a bounded-delay model. In *SSS'07*, Paris, France, Nov 2007.
10. D. Dolev and E. N. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In *DISC'07*, Lemesos, Cyprus, Sep. 2007.
11. S. Dolev. *Self-Stabilization*. The MIT Press, 2000.
12. 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.
13. C. Dwork and O. Waarts. Simple and efficient bounded concurrent timestamping or bounded concurrent timestamp systems are comprehensible. In *STOC '92*, 1992.

14. T. Herman. Phase clocks for transient fault repair. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1048–1057, 2000.
15. E. N. Hoch, D. Dolev, and A. Daliot. Self-stabilizing byzantine digital clock synchronization. In *SSS'06*, Dallas, Texas, Nov 2006.
16. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
17. Toshimitsu Masuzawa and Sébastien Tixeuil. Bounding the impact of unbounded attacks in stabilization. In *SSS*, pages 440–453, 2006.
18. Mikhail Nesterenko and Anish Arora. Tolerance to unbounded byzantine faults. In *SRDS '02*, page 22, Washington, DC, USA, 2002. IEEE Computer Society.
19. Yusuke Sakurai, Fukuhito Ooshita, and Toshimitsu Masuzawa. A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In *OPODIS*, 2004.