Timeliness, Failure-Detectors, and Consensus Performance

Idit Keidar∗ Alexander Shraer†

Abstract

We study the implication that various timeliness and failure detector assumptions have on the performance of consensus algorithms that exploit them. We present a general framework, GIRAF, for expressing such assumptions, and reasoning about the performance of indulgent algorithms.

Keywords: eventual synchrony, failure detectors, lower bounds, indulgent consensus.

1 Introduction

1.1 Background and motivation

Consensus is a widely-studied fundamental problem in distributed computing, theory and practice. Roughly speaking, it allows processes to agree on a common output. We are interested in the performance of consensus algorithms in different timing models.

Although the synchronous model provides a convenient programming framework, it is often too restrictive, as it requires implementations to use very conservative timeouts to ensure that messages are never late. For example, in some practical settings, there is a difference of two orders of magnitude between average and maximum message latencies [6, 5]. Therefore, a system design that does not rely on strict synchrony is often advocated [25, 17, 8]; algorithms that tolerate arbitrary periods of asynchrony are called indulgent [20].

As it is well-known that consensus is not solvable in asynchronous systems [18], the feasibility of indulgent consensus is contingent on additional assumptions. More specifically, such a system may be asynchronous for an unbounded period of time, but eventually reaches a Global Stabilization Time (GST) [17], following which certain properties hold. These properties can be expressed in terms of eventual timeliness of communication links [17, 10], or using the abstraction of oracle failure detectors [8]. Protocols in such models usually progress in asynchronous rounds, where, in each round, a process sends messages (often to all processes), then receives messages while waiting for some condition expressed as a timeout or as the oracle’s output, and finally performs local processing.

Recent work has focused on weakening post-GST synchrony assumptions [22, 1, 2, 3, 27], e.g., by only requiring one process to have timely communication with other processes after GST. Clearly, weakening timeliness requirements is desirable, as this makes it easier to meet them. For example, given a good choice of a leader process, it is possible to choose a fairly small timeout, so that the leader is almost always able to communicate with all processes before the timeout expires, whereas having each process usually succeed to communicate with every other processes requires a much larger timeout [5, 4]. In general, the weaker the eventual timeliness properties assumed by an algorithm are, the shorter the timeouts its implementation needs to use, and the faster its communication rounds can be.

∗Department of Electrical Engineering, Technion, Israel. idish@ee.technion.ac.il.
†Department of Computer Science, Technion, Israel. shralex@cs.technion.ac.il.

1A timely link delivers messages with a bounded latency; the bound is either known or unknown a priori.
Unfortunately, faster communication rounds do not necessarily imply faster consensus decision; the latter also depends on the number of rounds a protocol employs. A stronger model, although more costly to implement, may allow for faster decision after GST. Moreover, although formally modeled as holding from GST to eternity, in practice, properties need only hold “enough time” for the algorithm to solve the problem (e.g., consensus) [23]. But how much time is “enough” depends on how quickly consensus can be solved based on these assumptions. Satisfying a weak property for a long time may be more difficult than satisfying a stronger property for a short time. Therefore, before choosing timeliness or failure detector assumptions to base a system upon, one must understand the implication these assumptions have on the running time of consensus. This is precisely the challenge we seek to address in this paper.

1.2 GIRAF – General Round-based Algorithm Framework

This question got little attention in the literature, perhaps due to the lack of a uniform framework for comparing the performance of asynchronous algorithms that use very different assumptions. Thus, the first contribution of our work is in introducing a general framework for answering such questions. In Section 2.2, we present GIRAF, a new abstraction of round-based algorithms, which separates an algorithm’s computation (in each round) from its round waiting condition. The former is controlled by the algorithm, whereas the latter is determined by an environment that satisfies the specified timeliness or failure detector properties. In addition, the environment can provide additional “oracle output” information for the protocol. In general, rounds are not synchronized among processes. GIRAF is inspired by Gafni’s round-by-round failure detector (RRFD) [19], but extends it to allow for more expressiveness in specifying round properties, in that the oracle output can have an arbitrary range and not just a suspect list as in [19], and our rounds do not have to be synchronized among processes or communication-closed like Gafni’s.

One model that we study and cannot be expressed in RRFD ensures that each process receives messages from a majority, and in addition, provides an eventual leader oracle, $\Omega$, which eventually outputs the same correct leader process at all processes; many consensus algorithms were designed for this model, e.g., [25, 13, 21]. Note that in order to ensure communication with a majority in each round, RRFD’s suspect list must include at most a minority of the processes, and hence cannot, by itself, indicate which of the unsuspected processes is leader. Thus, additional oracle output is required. In general, the question of which systems can be implemented in RRFD was left open [19]. In contrast, we show that GIRAF is general enough to faithfully capture any oracle-based asynchronous algorithm in the model of [7] (see Section 4). Note that GIRAF can be used to express models assuming various failure patterns and is not constrained to the crash failure model (even though the models we define in this paper do assume crash failures). Moreover, GIRAF can be used to study problems other than consensus.

Since we focus on round-based computations, we replace the notion of GST with the notion of a Global Stabilization Round (GSR) [12]. Each run eventually reaches a round GSR, after which no process fails, and all “eventual” properties hold. More specifically, an environment in our model is defined using two types of properties: (1) perpetual properties, which hold in all rounds; and (2) eventual properties, which hold from GSR onward. The eventual counterpart of a perpetual property $\phi$ is denoted $\Diamond \phi$.

Since one can define different round properties, and organize algorithms into rounds where these properties hold, one can prove upper and lower bounds for the number of rounds of different types (i.e., satisfying different properties such as all-to-all communication in each round or communication with a majority in each round) that are sufficient and necessary for consensus. Note that, in order to deduce which algorithm is best for a given network setting, this analysis should be complemented with a measurement study of the cost of rounds of different types in that specific setting. The latter is highly dependant on the particularities of the given network and has no general answers (e.g., in a LAN, all-to-all communication may well cost

---

2In communication-closed rounds, each message arrives in the round in which it is sent.
the same as communication with majority, whereas in a WAN it clearly does not [5, 4]). GIRAF provides generic analysis, which can be combined with a network-specific measurements to get a network-specific bottom line.

We use GIRAF to revisit the notion of oracle (or model) reducibility. Traditionally, reducibility defines when one model can be implemented in another [8, 7], without taking complexity into account. In Section 3, we define \( k \)-round reducibility, which captures reductions that incur a \( k \)-round penalty in running time; i.e., the GSR of a run in the emulated model is at most \( k \) rounds later than that of the run in the original model. Additionally, we define the more general notion of \( \alpha \)-reducibility, where the GSR of the emulated model is at most a function \( \alpha(x) \) of that of the original model in the same run. Thus, \( k \)-round reducibility is simply \( \alpha \)-reducibility with \( \alpha(x) = x + k \). Gafni [19] has posed as an open problem the question of finding a notion of an equivalence relation between models with regard to the extent in which one model “resembles” another. We hope that our notion of \( \alpha \)-reducibility (and \( k \)-round reducibility) provides a convenient instrument to describe such relations.

1.3 Results

We use GIRAF to analyze consensus performance in different models. In this paper, we consider a crash-failure model, where up to \( t < n/2 \) out of \( n \) processes may crash (before GSR). Our performance measure is the number of rounds until global decision, i.e., until all correct processes decide, after GSR.

Dutta et al. [12] have shown that in the Eventual Synchrony (ES) [17] model, where all links are timely from GSR onward, GSR+2, i.e., three rounds including round GSR, is a tight lower bound for global decision. We are interested in the implications of weakening the ES assumptions. Following the observation that in some settings communication with a leader or a majority can be achieved with significantly shorter timeouts than required for timely communication with all processes [5, 4], we focus on leader-based and majority-based models.

The first model we define is Eventual Leader-Majority, \( \diamond \text{LM} \) (see Table 1, row 2). In this model, processes are equipped with a leader oracle, \( \Omega \) [7]. We further require that the leader be a \( \diamond n \)-source, where a process \( p \) is a \( \diamond j \)-source if it has \( j \) timely outgoing links in every round starting from GSR [2] (the \( j \) recipients include \( p \), and are not required to be correct)\(^3\). Finally, we require that each correct process eventually have timely incoming links from a majority of correct processes (including itself) in each round; this property is denoted \( \diamond (\lfloor \frac{n}{2} \rfloor + 1) \)-destination, where the subscript \( v \) denotes that the incoming links of a process can change in each round. \( \diamond \text{LM} \) does not impose any restrictions on the environment before GSR. One might expect that weakening the ES model in this way would hamper the running time. Surprisingly, in Section 5, we present a leader-based consensus algorithm for \( \diamond \text{LM} \), which achieves the tight bound for ES, i.e., global decision by GSR+2. Our result suggests that eventually perfect failure detection is not required for optimal performance; \( \Omega \) and timely communication with a majority suffice.

We then turn to see whether we can replace \( \Omega \) with an equivalent (in the “classical” sense) failure detector, \( \diamond S \) [8]. \( \diamond S \) outputs a list of suspected processes at each process, so that eventually, every correct process suspects every faulty process, and there exists one correct process that is not suspected by any process. In Section 6, we show that in runs in which less than \( \frac{n}{2} - 1 \) processes fail, replacing \( \Omega \) with \( \diamond S \) entails a lower bound of \( n \) rounds from GSR onward. In the literature, \( \diamond S \) is typically used with the assumptions that links are reliable (although not timely), and that messages from a majority arrive in each round, including before GSR [8, 28]. We therefore prove our lower bound for a stronger model, which we call Eventually Strong-Reliable, \( \diamond \text{SR} \). In this model, all links are reliable, processes are equipped with a \( \diamond S \) failure detector, the unsuspected correct process is a \( \diamond n \)-source, and every process is a perpetual

\(^3\)In [2], the link from \( p \) to itself is not counted; hence a \( j \)-source in our terminology is a \((j - 1)\)-source in theirs.
(n − f − 1)-destination.\(v\), where \(f \leq t\) is the number of actual failures in a given run, which is at least a majority whenever \(f < \frac{n}{2} − 1\) (see Table 1, row 3).

The \(Ω\) oracle is clearly a powerful tool; our result for \(\Diamond LM\) shows that \(Ω\) effectively eliminates the need for communication with all processes, and renders communication with majority sufficient. This raises the question of whether timely communication with a majority can suffice for constant-time consensus even without an oracle. To answer this question, we define the Eventual All-from-Majority, \(\Diamond AFM\), model, where each correct process eventually has incoming timely links from a majority of processes, and outgoing timely links to a majority (including itself). It is possible for processes to have fewer outgoing links, and in return have additional incoming ones (see Table 1, row 4). In Section 7, we give a consensus algorithm for this model that decides in constant time after GSR (five or six rounds, depending on the number of outgoing versus incoming timely links). We are not aware of any previous algorithm for the \(\Diamond AFM\) model, nor any other constant-time (from GSR) oracle-free algorithm in a timing model other than ES.

Finally, we examine whether one can weaken the model even further. Can we relax the assumption that all correct processes have incoming timely links from a majority, and allow a minority of the processes to have one fewer timely link? (In case \(n = 3\), only one timely link is removed). In Section 8, we show that the answer to this question is no, as this renders the problem unsolvable in bounded time. We define a family of models, Eventual Majority-from-Majority, \(\Diamond MFM(m)\), where, roughly speaking, a majority of the processes have incoming timely links from a majority, and the rest have incoming timely links from a minority. In order to strengthen the lower bound, we add a host of additional assumptions (see Table 1, row 5): We require a minority of processes to be \(\Diamond n\)-sources. We replace \(j\)-destination assumptions with \(j\)-accessibility [1, 27], i.e., the existence of bidirectional timely links with \(j\) correct processes. Finally, we require reliable links. We show that in the resulting models, \(\Diamond MFM(m)\), global decision cannot be achieved in bounded time from GSR. Interestingly, these models are strictly stronger than those of [2, 3, 27], which were used for solving consensus. We note, though, that in [27], local decision (of the leader and its accessible destinations) is possible in constant time, whereas in [2, 3], local decision time is unbounded as well.

As Table 1 shows, there are still several tantalizing gaps between the known upper and lower bounds in various models. Moreover, many additional models can be explored, e.g., in the middle ground between
AFM and MFM. We hope that GIRAF will allow researchers to address many such issues in future work. Section 9 provides further discussion of future research directions.

1.4 Related work

In recent years, a number of efforts have been dedicated to understanding the performance of asynchronous algorithms in runs that are synchronous (or the failure detector is perfect) from the outset [24, 13, 21, 16, 14], typically focusing on the case that all failures are initial, which corresponds to $GSR = 0$ in our model.

Only very recently, the issue of performance following asynchronous periods has begun to get attention [12, 15]. As noted above, [12] shows that $GSR + 2$ is a tight bound for global decision in ES. It uses Gafni’s RRFD [19] framework. In [15], Dutta et al. focus on actual time rather than rounds, again in ES; they present an algorithm that decides by $GST + 17\delta$, where $\delta$ is a bound on message delay from GST onward (but no matching lower bound). This result gives a more accurate assessment of the actual running time after GST than out round-count offers. Nevertheless, a similar assessment might be obtained in our model if one can quantify the time it takes the environment’s synchronization to establish GSR after GST; this is an interesting subject for future study. We believe that the clean separation we offer between round synchronization and the consensus algorithm’s logic allows for more abstract and easier to understand protocol formulations and complexity analyses, as well as for proving lower bounds.

The only previous algorithm presented in the $\Diamond LM$ model, Paxos [25], may require a linear number of rounds after GSR [11]. Most other $\Omega$-based protocols, e.g., [13, 21], wait for messages from a majority in each round (including before GSR), which is undesirable, as it may cause processes to be in arbitrarily unsynchronized rounds when some process first reaches round GSR, causing GSR itself to take a long time. Dutta et al. [11] allow processes to “skip” rounds in order to re-synchronize in such situations. Implementing such approach in our framework yields an algorithm that requires one more round than Algorithm 2.

2 Model and Problem Definition

2.1 Distributed computation model

We consider an asynchronous distributed system consisting of a set $\Pi$ of $n > 1$ processes, $p_1, p_2, \ldots, p_n$, fully connected by communication links. Processes and links are modeled as deterministic I/O automata [26]. An automaton’s transitions are triggered by actions, which are classified as input, output, and internal. Action $\pi$ of automaton $A$ is enabled in state $s$ if $A$ has a transition of the form $(s, \pi, s')$. The transitions triggered by input actions are always enabled, whereas those triggered by output and internal actions are preconditioned on the automaton’s current state.

A run of I/O automaton $A$ is an infinite sequence of alternating states and actions $s_0, \pi_1, s_1, \ldots$, where $s_0$ is $A$’s initial state, and each triple $(s_{i-1}, \pi_i, s_i)$ is a transition of $A$. We only consider fair runs, where no action is enabled without occurring in an infinite suffix.

A process $p_i$ interacts with its incoming link from process $p_j$ via the $receive(m)_{i,j}$ action, and with its outgoing link to $p_j$ via the $send(m)_{i,j}$ action. Communication links do not create, duplicate, or alter messages (this property is called integrity). Messages may be lost by links.

A threshold of $t$ of the processes may fail by crashing. The failure of process $p_i$ is modeled using the action $\text{crash}_i$, which disables all locally controlled actions of $p_i$. A process that does not fail is correct. The actual number of failures occurring in a run is denoted $f$. Process $p_i$ is equipped with a failure detector oracle, which can have an arbitrary output range [7], and is queried using the $\text{oracle}_i$ function.
onward. More formally, in every run there is a round GSR(r) so that from round GSR(r) onward, no

<table>
<thead>
<tr>
<th>States:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k_i \in N, \text{initially } 0 ) /&quot;round number&quot;/*</td>
</tr>
<tr>
<td>( \text{sent}_i[\Pi] \in \text{Boolean array}, \text{initially } \forall p_j \in \Pi : \text{sent}_i[j] = true )</td>
</tr>
<tr>
<td>( DF_i \in \text{OracleRange}, \text{initially arbitrary} )</td>
</tr>
<tr>
<td>( M_i[N][\Pi] \in Messages \cup { \bot }, \text{initially } \forall k \in N \forall p_j \in \Pi : M_i[k][j] = \bot )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions and Transitions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>input ( \text{receive}(\langle m, k \rangle)_{i,j}, k \in N )</td>
</tr>
<tr>
<td>Effect: ( M_i[k][j] \leftarrow m )</td>
</tr>
<tr>
<td>Precondition: ( \text{sent}_i[j] = true )</td>
</tr>
<tr>
<td>Effect: ( \text{sent}_i[j] \leftarrow false )</td>
</tr>
<tr>
<td>output ( \text{send}(\langle M_i[k_i][i], k_i \rangle)_{i,j} )</td>
</tr>
<tr>
<td>Precondition: ( \text{sent}_i[j] = false )</td>
</tr>
<tr>
<td>Effect: ( \text{sent}_i[j] \leftarrow true )</td>
</tr>
</tbody>
</table>

input \( \text{end-of-round}_i \)
Effect: \( DF_i \leftarrow \text{oracle}_i(k_i) \)
if \( (k_i = 0) \) then \( M_i[1][i] \leftarrow \text{initialize}(DF_i) \)
else \( M_i[k_i + 1][i] \leftarrow \text{compute}(k_i, M_i, DF_i) \)
\( k_i \leftarrow k_i + 1 \)
\( \forall p_j \in \Pi : \text{sent}_i[j] \leftarrow false \)

Algorithm 1: GIRAF: Generic algorithm for process \( p_i \) (I/O automaton).

### 2.2 GIRAF – General Round-based Algorithm Framework

Algorithm 1 presents GIRAF, a generic round-based distributed algorithm framework. To implement a specific algorithm, GIRAF is instantiated with two functions: \text{initialize}(), and \text{compute}(). Both are passed the oracle output, and \text{compute}() also takes as parameters the set of messages received so far and the round number. These functions are executed atomically as part of one automaton action, and are not allowed to block or wait.

Each process’s computation proceeds in rounds. The advancement of rounds is controlled by the environment via the \text{end-of-round} input action. It first occurs in round 0, whereupon it queries the oracle and calls \text{initialize}(), which creates the message for sending in the first round (round one). Subsequently, during each round, the process sends a message to all processes and receives messages available on incoming links, until the \text{end-of-round} action occurs, at which point the oracle is queried and \text{compute}() is called, which returns the message for the next round. We say that an event of process \( p_i \) occurs in round \( k \) of run \( r \), if there are exactly \( k \) invocations of \text{end-of-round}_i before that event in \( r \). For simplicity, we have the algorithm send the same message to all processes in each round; this is without loss of generality as we are not interested in message complexity as a performance metric. The outgoing message is stored in the incoming message buffer, \( M_i[k_i + 1][i] \), hence self-delivery is ensured. The environment might decide not to send the message of a round to any subset of processes, i.e., it might invoke \text{end-of-round}_i in round \( k \) without a \text{send}(m)_{i,j} action ever happening in round \( k \) for a process \( p_j \). However, some of our environment definitions below will restrict this behavior and require messages to be sent. In any case, self-delivery is always preserved.

Our framework can capture any asynchronous oracle-based message-passing algorithm in the general model of [7] (see Section 3). Thus, GIRAF does not restrict the allowed algorithms in any way, but rather imposes a round structure that allows for analyzing them.

Each run is determined by the algorithm automaton’s state transitions, and the environment’s actions, consisting of (i) scheduling \text{end-of-round} actions; (ii) oracle outputs; and (iii) \text{send} and \text{receive} actions of the communication links. Environments are specified using round-based properties, restricting the oracle outputs or message arrivals in each round. We consider two types of environment properties: perpetual properties, which hold in each round, and eventual properties, which hold from some (unknown) round onward. More formally, in every run \( r \) there is a round GSR(r) so that from round GSR(r) onward, no
process fails, and all eventual properties hold in each round. GSR($r$) is the first round in $r$ that satisfies this requirement. (We henceforth omit the ($r$) where it is clear from the context).

Note that although, in general, rounds are not synchronized among processes, we specify below environment properties that do require some synchronization, e.g., that some messages are received at one process at the same round in which they are sent by another. Therefore, an implementation of an environment that guarantees such properties needs to employ some sort of round or clock synchronization mechanism (e.g. [17, 30], or using GPS clocks).

2.3 Environment properties

We define several environment properties in GIRAF, mostly in perpetual form. Prefixing a property with ♦ means that it holds from GSR onward.

**Communication Properties** Every process has a “link” with itself, and though it is not an actual physical link, it counts toward the $j$ timely links in the definitions below. Some of the properties that require $j$ timely links may appear with a subscript $v$ (variable), which indicates that the set of $j$ timely links is allowed to change in each round. Note that link integrity is assumed by the model. When characterizing a link, we denote the source process of the link by $p_s$, and the recipient by $p_d$.

**reliable link:** if end-of-round$_s$ occurs in round $k$ and $p_d$ is correct, then $p_d$ receives the round $k$ message of $p_s$.

**timely link:** if end-of-round$_s$ occurs in round $k$ and $p_d$ is correct, then $p_d$ receives the round $k$ message of $p_s$, in round $k$.

**$j$-source:** process $p$ is a $j$-source if there are $j$ processes to which it has timely outgoing links in every round; $p$ is a $j$-source$_v$ if in every round it has $j$ timely outgoing links. (Correctness is not required from the recipients.)

**$j$-destination:** correct process $p$ is a $j$-destination if there are $j$ correct processes from which $p$ has timely incoming links in every round; $p$ is a $j$-destination$_v$ if it has $j$ timely incoming links from correct processes in every round.

**$j$-accessible:** correct process $p$ is $j$-accessible if there are $j$ correct processes with which $p$ has timely bidirectional links in every round. (We do not consider variable $j$-accessibility in this paper.)

Note that the reliable and timely link properties assure that the environment sends messages on the link, i.e., the end-of-round$_s$ action in round $k$ is preceded by a send($m$)$_{s,d}$ action in round $k$.

**Failure Detector Properties** We next define several oracle properties [7, 8]. The range of the oracle() function for $S$ (and ♦$S$) is $2^\Pi$ – a group of suspected processes. For leader (and $\Omega$), the range is $\Pi$.

$S$ failure detector: ♦$SC$ (strong completeness) – eventually every faulty process is suspected by every correct process, and $WA$ (weak accuracy) – some correct process is not suspected.

leader: $\exists$ correct $p_i$ s.t. for every round $k \in N$ and every $p_j \in \Pi$, oracle$_j(k) = i$.

$\Omega$ failure detector: ♦ leader.
2.4 Consensus and global decision

A consensus problem is defined for a given value domain, Values. In this paper, we assume that Values is a totally ordered set. In a consensus algorithm, every process $p_i$ has a read-only variable $prop_i \in Values$ and a write-once variable $dec_i \in Values \cup \{\bot\}$. In every run $r$, $prop_i$ is initialized to some value $v \in Values$, and $dec_i$ is initialized to $\bot$. We say that $p_i$ decides $d \in Values$ in round $k$ of $r$ if $p_i$ writes $d$ to $dec_i$ when $k_i = k$ in $r$.

An algorithm $A$ solves consensus if in every run $r$ of $A$ the following three properties are satisfied: (a) (validity) if a process decides $v$ then $prop_i = v$ for some process $p_i$, (b) (agreement) no two correct processes decide differently, and (c) (termination) every correct process eventually decides.

We say that a run of $A$ achieves global decision at round $k$ if (1) every process that decides in that run decides at round $k$ or at a lower round; and (2) at least one process decides at round $k$.

3 Complexity of Reductions

In discussing different models, the question of reducibility naturally arises – one is often interested whether one model is stronger than another, or how “close” two models are. The classical notion of reducibility among models/oracles [8, 7] does not take complexity into account. We use GIRAF to provide a more fine-grained notion of similarity between models.

We first explain how classical reducibility is expressed for GIRAF models. Reducibility (in the “classical” sense) means that one model can be emulated in another. A simulation from a GIRAF model $M_1$ to another (GIRAF or non-GIRAF) model $M_2$, must work within the $initialize()$ and $compute()$ functions in $M_1$, which must be non-blocking. Simulating a GIRAF model $M_2$ means invoking the $initialize_A()$ and $compute_A()$ functions of some algorithm $A$ that works in $M_2$, while satisfying the properties of $M_2$. In particular, if $M_1$ and $M_2$ are both GIRAF models, then a reduction algorithm $T_{M_1 \rightarrow M_2}$ instantiates the $initialize()$ and $compute()$ functions, denoted $initialize_T()$ and $compute_T()$, and invokes $initialize_A()$ and $compute_A()$ in model $M_1$. If algorithm $T_{M_1 \rightarrow M_2}$ exists, we say that $M_2$ is reducible to $M_1$ (or weaker than $M_1$), and denote this by $M_1 \geq M_2$. $M_1$ is equivalent to $M_2$ if $M_1 \geq M_2$ and $M_2 \geq M_1$.

We next extend the notion of reducibility, and introduce $\alpha$-reducibility, which takes the reduction time (round) complexity into account. Note that the definition of a run’s GSR is model-specific: $GSR(r) = k$ in model $M$ if $k$ is the first round from which onward no process fails and the eventual properties of $M$ are satisfied. We denote GSR in model $M$ and run $r$ by $GSR_M(r)$.

**Definition ($\alpha$-reducibility).** Model $M_2$ is $\alpha$-reducible ($\alpha : N \rightarrow N$) to model $M_1$, denoted $M_1 \geq_\alpha M_2$, if there exists a reduction algorithm $T_{M_1 \rightarrow M_2}$ s.t. for every run $r$, $GSR_{M_2}(r) \leq \alpha(GSR_{M_1}(r))$.

**Definition ($k$-round reducibility).** Model $M_2$ is $k$-round reducible ($k \in N$) to model $M_1$, denoted $M_1 \geq_k M_2$, if $M_1 \geq_\alpha M_2$ s.t. $\alpha(x) = x + k$.

In particular, if $M_1 \geq_0 M_2$ then model $M_2$ can be simulated in model $M_1$ with no performance penalty. In Section 6, we use the notion of $k$-round reducibility to prove that $\diamond S$ is 0-round reducible to $\diamond n$-source.

4 Generality of GIRAF

In this section we show how GIRAF relates to the framework of [7]. A computation step in the model of [7] consists of (i) receiving a message, (ii) consulting the oracle, (iii) using the process’s algorithm ($A(p)$) in their
Lemma 1. Every model $M_1$ in the framework of [7] is equivalent to a GIRAF model $M_2$, where the only environment properties are the same oracle properties as in $M_1$ and reliable links.

Proof. We prove that the framework of [7] with model $M_1$ can be used to implement the environment for GIRAF resulting in model $M_2$. Note that in [7], $A(p)$ is invoked upon every message receipt after the oracle is queried, and the oracle output and incoming message are available to it. To run the generic GIRAF algorithm in model $M_1$, we have $A(p)$ first invoke initialize(), and subsequently invoke compute() every time it is called to take a step. $A(p)$ passes to these functions the oracle output. To compute(), it also passes the set of messages received thus far, and a counter of the number of times compute() is called. compute() or initialize() returns a message, which is sent immediately afterwards. Every perpetual property guaranteed by the oracle of $M_1$ holds starting from the first round in $M_2$, and every eventual property of the oracle is eventually true in $M_2$, guaranteeing that the properties of $M_2$’s oracle are preserved.

We next prove that GIRAF with model $M_2$ can be used to simulate the framework of [7]. Given an algorithm $A(p)$ designed for the framework of [7] and model $M_1$, we make the compute() function of GIRAF invoke a series of steps of $A(p)$ – one invocation for every message $m$ added to $M$ since the previous time compute() was activated. compute() then aggregates all the messages that these steps return into one composite message which is returned to GIRAF generic algorithm (to be sent in the next round). When a step of $A(p)$ queries the oracle, it is given the oracle output passed to compute(). Note that each step of $A(p)$ is atomic, and therefore compute() is atomic, as required by GIRAF. Since the message arrival order is arbitrary in [7], this a valid run in $M_1$. Every perpetual property of $M_2$’s oracle is preserved starting from the first round and is therefore true starting from the first activation of $A(p)$, and every eventual property will hold starting from GSR in $M_2$ and therefore eventually holds in $M_1$. 

5 Optimal Leader-Based Algorithm in $\diamond LM$

The $\diamond LM$ model is strictly weaker than ES: it requires that each process have only a majority of incoming timely links (from GSR onward), which can vary in each round, and an $\Omega$ oracle that selects a correct $\diamond n$-source as leader (it is easy to implement $\Omega$ in ES, e.g., by choosing the lowest-id correct process). Formally:

$\diamond LM$ (Leader-Majority): $t < n/2$, $\Omega$ failure detector, the leader is a $\diamond n$-source, and every correct process is a $\diamond (\lceil n/2 \rceil + 1)$-destination.

Algorithm. Algorithm 2 presents a leader-based consensus algorithm for $\diamond LM$, which reaches global decision by round GSR+2. In runs with GSR = 0, this means that consensus is achieved in 2 rounds, which is tight [9, 24]. In runs with GSR > 0, global decision is reached in 3 rounds, numbered GSR, GSR+1, and GSR+2, which also matches the lower bound for ES [12].

Algorithm 2 works in GIRAF, and therefore implements only the initialize() and compute() functions. These function are passed leader, the leader trusted by the oracle.

The main idea of the algorithm, which ensures fast convergence, is to trust the leader even if it competes against a higher bid of another process. In contrast, Paxos [25] initiates a new “ballot”, that is, aborts any pending attempts to decide on some value, whenever a higher timestamp is observed, potentially leading to linear running time after GSR [11]. In order to ensure that the leader does not propose a value that contradicts previous agreement, the lastApproval variable (and message-field) conveys the “freshness” of the leader’s proposed value, and the leader’s proposals are not accepted if it is not up-to-date.
1: Additional state
2:  \( est_i \in Values, \text{initially} \ prop_i \)
3:  \( ts_i, maxTS_i, lastApproval_i \in N, \text{initially} 0 \)
4:  \( prevLD_i, newLD_i \in \Pi \)
5:  \( msgType_i \in \{ \text{PREPARE, COMMIT, DECIDE} \}, \text{initially} \text{PREPARE} \)
6: Message format
7:  \( (msgType \in \{ \text{PREPARE, COMMIT, DECIDE} \}, \text{est} \in Values, ts \in N, \text{leader} \in \Pi, \text{lastApproval} \in N) \)
8: procedure initialize(leader,)
9:  \( prevLD_i \leftarrow newLD_i \leftarrow \text{leader} \)
10: return message \( (msgType_i, est_i, ts_i, newLD_i, \text{lastApproval}_i) \) /*round 1 message*/
11: procedure compute(\( k_i, M[[*]][*], \text{leader} \),)
12: if \( \text{dec} = \perp \) then
13: /*Update variables*/
14:  \( prevLD_i \leftarrow newLD_i; \text{newLD}_i \leftarrow \text{leader} \)
15:  \( maxTS_i \leftarrow \max \{ m.ts | m \in M[k][*] \} \)
16: if \( \{ j | M[k][j][*] \neq \perp \} > \lceil n/2 \rceil \) then
17:  \( \text{lastApproval}_i \leftarrow k_i \)
18: /*Round Actions*/
19: if \( \exists m \in M[k][*] \) s.t. \( m.msgType = \text{DECIDE} \) then /*decide-1*/
20: \( dec_i \leftarrow est_i; msgType_i \leftarrow \text{DECIDE} \)
21: else if \( \{ j | M[k][j][*] = \text{COMMIT} \} > \lceil n/2 \rceil \)
22: and \( (M[k][prevLD][*].msgType = M[k][i].msgType = \text{COMMIT}) \) then /*decide-2*/
23: \( dec_i \leftarrow est_i; msgType_i \leftarrow \text{DECIDE} \)
24: else if \( \{ j | M[k][j][*].leader = prevLD[i] \} \geq \lceil n/2 \rceil \) /*commit-1*/
25: and \( (M[k][prevLD][i].lastApproval = \text{leader} = prevLD_i) \) /*commit-2*/
26: and \( (\text{newLD} = prevLD_i) \) then /*commit-3*/
27: \( est_i \leftarrow M[k][prevLD][i].est; ts_i \leftarrow \text{leader}; msgType_i \leftarrow \text{COMMIT} \)
28: \( ts_i \leftarrow \text{maxTS}_i; msgType_i \leftarrow \text{PREPARE} \)
29: return message \( (msgType_i, est_i, ts_i, newLD_i, \text{lastApproval}_i) \) /*round \( k_i + 1 \) message*/

Algorithm 2: Optimal leader–based algorithm for \( \diamond LM \), code for process \( p_i \).

We now describe the protocol in more detail. Process \( p_i \) maintains the following local variables: an estimate of the decision value, \( est_i \) initialized to the proposal value (\( prop_i \)); the timestamp of the estimated value, \( ts_i \), and the maximal timestamp received in the current round, \( maxTS_i \), both initialized to 0; the index of the last round in which \( p_i \) receives a message from a majority of processes, \( lastApproval_i \), initialized to 0; the leader provided by the oracle at the end of the previous round, \( prevLD_i \), and in the current round, \( newLD_i \); and the message type, \( msgType_i \), which is used as follows: If \( p_i \) sees a possibility of decision in the next round, then it sends a COMMIT message. Once \( p_i \) decides, it sends a DECIDE message in all subsequent rounds. Otherwise, the message type is PREPARE.

We now describe the computation of round \( k_i \). If \( p_i \) has not decided, it updates its variables as follows. It saves its previous leader assessment in \( prevLD_i \), and its new leader (as passed by the oracle) in \( newLD_i \) (line 14). It stores the highest timestamp received in \( maxTS_i \). If \( p_i \) receives a message from a majority, it sets \( lastApproval_i \) to the round number, \( k_i \). It then executes the following conditional statements:

- If \( p_i \) receives a DECIDE message then it decides on the received estimate by writing that estimate to \( \text{dec}_i \) (line 20).
- If \( p_i \) receives COMMIT messages from a majority of processes, including itself and its leader, then \( p_i \) decides on its own estimate (line 22).
Let $prevLD_i$ be the leader indicated in $p_i$’s round $k_i$ message. Consider the following three conditions (line 23): commit-1: $p_i$ receives round $k_i$ messages from a majority of processes that indicate $prevLD_i$ as their leader; commit-2: $p_i$ receives a message from $prevLD_i$ that has $prevLD_i$ as the leader, and $lastApproval$ set to $k_i - 1$; and commit-3: $prevLD_i = newLD_i$. If all three conditions are satisfied, then $p_i$ sets its message type (for the round $k_i + 1$ message) to COMMIT, adopts the estimate received from $prevLD_i$, say $est'$, and sets its timestamp to the current round number $k_i$ (line 24). We say that $p_i$ commits in round $k_i$ with estimate $est'$.

Otherwise, $p_i$ adopts the estimate and the timestamp of an arbitrary message with the highest timestamp $maxTS_i$, and sets the message type to PREPARE (lines 26–27).

Finally, $p_i$ returns the message for the next round.

**Correctness.** We formally prove Algorithm 2’s correctness in Appendix A. Our main lemma (Lemma 11) shows that no two processes decide differently, by showing that if some process decides $x$ in round $k$, then from round $k - 1$ onward, the only committed estimate is $x$. (This proves agreement since a decision is made when either a DECIDE or a majority of COMMITS is received.) We now intuitively explain why this is correct. The claim is proven by induction on round number. Let $p_1$ be the first process that decides, and denote its decision value by $x$, and the decision round by $k$. (the decision is by rule decide-2; rule decide-1 is not applicable since $p_1$ is the first process to decide). Therefore, in round $k$, $p_i$ hears COMMIT from majority $M$, including itself and its round $k$ $prevLD_i$, $p_i$, and decides on its own estimate, $x$. Let us first examine round $k - 1$. Processes of $M$ commit in this round. Rules commit-1 and commit-3 ensure that all COMMIT messages sent in this round have the same estimate and leader fields, namely, $x$, and $p_1$. Additionally, it is easy to see that a process’s timestamp never decreases. Thus, since processes of $M$ commit in round $k - 1$, they have timestamps of at least $k - 1$ in all ensuing rounds. Now consider round $k$. Any process that commits in round $k$ hears from a majority with the same leader, and since this majority intersects $M$, the leader is $p_i$. Therefore, any commitment in round $k$ is made with the estimate of $p_i$, i.e., $x$.

We now consider the inductive step, i.e., round $k' > k$. If $p_i$ commits in round $k'$, it commits on the estimate of its leader. If that leader sends a COMMIT message, by induction, its estimate is $x$. Otherwise, the leader sends a PREPARE message. By commit-2, that leader’s $lastApproval$ field is set to $k' - 1 ≥ k$, implying that the leader receives a message from a majority of processes in round $k' - 1$. Therefore, it receives at least one message from a process in $M$ with timestamp at least $k - 1$. Since the highest timestamp received is adopted, the leader adopts timestamp $ts ≥ k - 1$ and some estimate $z$. It is easy to see that if a message (other than DECIDE) is sent with timestamp $ts$ and estimate $z$, then some process commits $z$ in round $ts$. Therefore, some process commits $z$ in a round $≥ k - 1$. By induction, we get that $z = x$. Therefore, the leader adopts $x$ with the maximal timestamp in round $k' - 1$, and $p_i$ commits $x$ in round $k'$.

**Performance.** We now give an intuitive explanation why in round GSR+1, every correct process $p_i$ that does not decide by the end of that round evaluates the three commit rules (line 23) to true (this is formally proven in Appendix A). Since $p_i$ does not decide by the end of GSR+1, all the processes it hears from in this round do not decide by round GSR. By definition of $\diamond LM$, from round GSR onward, each correct process receives messages from a majority of correct processes, including its leader, $p_l$. Therefore, the $lastApproval$ field of every round GSR+1 message is GSR (notice for the case of GSR=0 that $lastApproval$ is initialized to 0). Moreover, it is assured by the $\Omega$ failure detector, that from round GSR onward, all processes trust the same leader, $p_l$. Therefore, from round GSR+1 onward, all running processes (including the leader $p_l$) send the same leader identifier in their messages. (Note that rule commit-3 is assured to be true only starting at round GSR+1, since $prevLD_i$ of round $k_i = GSR$ is based on the oracle’s output in round GSR−1, in which
it is not assured that all processes trust the same leader.) We conclude that in round $GSR+2$ every correct process sends a COMMIT or DECIDE message, and by the end of that round, every correct process decides.

6 Linear Bound for $\Diamond SR$

We use the notion of $k$-round reducibility, to prove that at least $n$ rounds starting at $GSR$ are needed to solve consensus in the $\Diamond SR$ model. We formally define the $\Diamond SR$ model as follows:

$\Diamond SR$ (Strong-Reliable): $t < n/2$, reliable links, $\Diamond S$ failure detector, the unsuspected process is $\Diamond n$-source and all correct processes are $(n - f - 1)$-destinations, where $f < n/2 - 1$.

Lemma 2. Any model $M_{\Diamond S}$ that requires a $\Diamond S$ failure detector and environment properties $\mathcal{P}$ is $0$-round reducible to a model $M_{\Diamond n}$ that assumes a correct $\Diamond n$-source process and $\mathcal{P}$, i.e. $M_{\Diamond n} \geq_0 M_{\Diamond S}$.

Proof. We implement the reduction algorithm $T_{M_{\Diamond n} \rightarrow M_{\Diamond S}}$ as follows: $compute_T()$ receives a multi-set of messages $M$ received so far, and the current round number $k$, but no oracle output (since $M_{\Diamond n}$ does not include an oracle) and produces the set of suspected processes $FD_T$ as follows: $FD_T \leftarrow \{ j \mid M[k][j] = \perp \}$. It then passes $M, k$ and $FD_T$ to $compute_A()$. $initialize_T()$ calls $initialize_A()$ with $\emptyset$ as the set of suspected processes. Since in every round $k' \geq GSR_{M_{\Diamond n}}$ there exists one process (the $\Diamond n$-source correct process) whose $k'$ round message reaches every correct process by the end of round $k'$, this process is not included in any of the $FD_T$ sets produced by algorithm $T_{M_{\Diamond n} \rightarrow M_{\Diamond S}}$ at any process in round $k'$, i.e. is not suspected. Since no faulty process enters round $GSR_{M_{\Diamond n}}$, no such process sends a round $k'$ message, and thus every faulty process is suspected from round $GSR_{M_{\Diamond n}}$ onward. Therefore, the produced set $FD_T$ satisfies the specification of $\Diamond S$ in our framework such that the eventual properties of $\Diamond S$ are satisfied from $GSR_{M_{\Diamond n}}$ onward. Since $M$ (the message set) and $k$ are not altered by $T_{M_{\Diamond n} \rightarrow M_{\Diamond S}}$, all the other properties $\mathcal{P}$ are still preserved from round $GSR_{M_{\Diamond n}}$ onward. Therefore, $GSR_{M_{\Diamond n}} = GSR_{M_{\Diamond S}}$ and $M_{\Diamond n} \geq_0 M_{\Diamond S}$. \qed

From Lemma 2, it follows that suffices to prove the lower bound for a model just like $\Diamond SR$, but without the assumption of $\Diamond S$. We denote this model by $\Diamond SR \setminus \Diamond S$.

We prove the lower bound using the impossibility of consensus in the mobile failure model [29], in which no process crashes, and in each communication step there is one process whose messages may be lost.

Below we denote the prefix of length $l$ rounds of a run $r$ by $r(l)$.

Lemma 3. For any $k \in N$, let $r$ be a run in the mobile failure model. There exists a run $r'$ in $\Diamond SR \setminus \Diamond S$ with $GSR(r) = k$ and $f = 0$ such that $r'(k + n - 2) = r(k + n - 2)$.

Proof. We construct $r'$ as follows: (i) $f = 0$ and $GSR(r') = k$, (ii) $r'$ is identical to $r$ in the first $k + n - 2$ rounds, except that messages are delayed to round $k + n - 1$ instead of being lost, and (iii) from round $k + n - 1$ onward, $r'$ is synchronous (all links are timely).

We show that $r'$ is a run in model $\Diamond SR \setminus \Diamond S$. In each round of $r'(k + n - 2)$, a subset of messages sent by at most one process is delayed and all other messages arrive in the same round in which they are sent, and from round $k + n - 1$ onward, no message is delayed in $r'$. Therefore, in $r'$, each process receives messages from at least $n - 1$ processes in every round and is therefore an $(n - f - 1)$-destination, (recall that $f = 0$ in $r'$). Since $r'(k + n - 2)$ lasts only $n - 1$ rounds starting from $GSR(r')$ (and there are $n$ processes), there exists some correct process whose messages are not delayed in any round from $GSR(r')$. This process is a correct $\Diamond n$-source in $r'$. Finally, since every message sent before round $k + n - 1$ in $r'$ arrives at the latest in round $k + n - 1$ and every message sent in later round arrives in the same round in which it is sent, we conclude that links are reliable in $r'$.
We strengthen the lower bound by proving that it is impossible to reach global decision in less than \( n \) rounds from GSR in the \( \Diamond SR \setminus \Diamond S \) model, even with an algorithm especially tailored for some specific GSR.

Lemma 4. For \( k \in N, k \geq 1 \), no algorithm exists that in every run \( r \) in which \( GSR(r) = k \) achieves global decision before round \( GSR(r) + (n - 1) \), in the \( \Diamond SR \setminus \Diamond S \) model.

Proof. For \( k \in N, k \geq 1 \), assume there exists an algorithm \( A_k \) that solves consensus in \( \Diamond SR \setminus \Diamond S \), and in every run with GSR\( = k \) reaches global decision by round \( k + n - 2 \). Then we run \( A_k \) in the mobile model for \( k + n - 2 \) rounds. Denote this run by \( r \). From Lemma 3, there is a run \( r' \) in \( \Diamond SR \setminus \Diamond S \) with \( GSR(r') = k \) and \( f = 0 \), such that \( r'(k + n - 2) = r(k + n - 2) \). Therefore, \( A_k \) cannot distinguish \( r \) from \( r' \) in the first \( k + n - 2 \) rounds and decides by round \( k + n - 2 \) in \( r \) as it does in \( r' \). We conclude that \( A_k \) reaches a global decision for every run \( r \) in the mobile failure model. A contradiction to [29].

Note that our proof (combined with Algorithm 2, which achieves global decision by GSR+2 in \( \Diamond LM \)) immediately implies that \( \Diamond SR \not\preceq_k \Diamond LM \) for any \( k < n - 3 \), since otherwise, we could use the reduction algorithm to simulate \( \Diamond LM \) in \( \Diamond SR \) in any run \( r \) with \( GSR_{\Diamond LM} < GSR_{\Diamond SR}(r) + n - 3 \) and use Algorithm 2 on top of the reduction algorithm. Since Algorithm 2 assures global decision by \( GSR_{\Diamond LM}(r) + 2 \) we get that there exists an algorithm that for any run \( r \) achieves global decision before round \( GSR_{\Diamond SR}(r) + n - 1 \), a contradiction to our lower bound.

7 Constant-Time Algorithm in \( \Diamond AFM \)

In this section, we investigate whether constant time decision is possible without an oracle in a model weaker than ES. We are not aware of any previous constant-time algorithms for such a model.

In the \( \Diamond AFM \) model, each process has timely incoming links from a correct majority of processes, and a majority of timely outgoing links (from GSR onward), both can vary in each round. The number of outgoing links may decrease if more incoming links are timely. Formally:

\( \Diamond AFM \) (All-From-Majority): \( t < n/2, \exists m \in N, f \leq m < n/2 \) such that every correct process is a \( \Diamond(n - m) \)-destination, and a \( \Diamond(m + 1) \)-source. Note that \( m \) can be different in each run.

Algorithm. Algorithm 3 is a majority-based algorithm for \( \Diamond AFM \), which always reaches global decision by round GSR+5. At the end of this section we present an optimization of the algorithm for the case of \( n = 2m + 1 \) (i.e., when both \( m + 1 \) and \( n - m \) are majorities), and in Appendix B we prove that the optimized algorithm reaches global decision by round GSR+4 for \( n = 2m + 1 \) and by round GSR+5 for other values of \( f \leq m < n/2 \). The code used for optimization is marked in gray in Algorithm 3 and should be ignored until its explanation at the end of this section.

In general, Algorithm 3 is similar to Algorithm 2. We therefore focus mainly on the differences from Algorithm 2. Since \( \Diamond AFM \) does not assume a failure detector, the oracle’s output is not a parameter for \( compute() \).

The variables maintained by each process \( p_i \) are similar to those of Algorithm 2. A new variable, \( maxEST_i \), holds the maximal estimate received with timestamp \( maxTS_i \) in the current round (recall that \( Values \) is a totally ordered set). A new message type is introduced, \( PRE-COMMIT \). Intuitively, pre-committing is similar to a committing, but without increasing the timestamp. An estimate must be pre-committed by some process before it is committed.

Pre-commit is needed, since, unlike \( \Diamond LM \), where the leader is a \( \Diamond n \)-source, \( \Diamond AFM \) never assures that a process is able to convey information to all other processes in a single round. If we hadn’t introduced \( PRE-COMMIT \), it would have been possible for two different estimates to be committed in alternating rounds,
where a majority of processes hear and adopt estimate $est_1$, (which has the maximal timestamp) but some other process does not hear $est_1$ and commits to $est_2$, increasing its timestamp. In the next round the situation flips, and $est_2$ is adopted by a majority while $est_1$ is committed, and so on, precluding decision.

In $\Diamond AFM$, in every round from GSR onward, each process hears from $(n - m)$ correct processes, and its outgoing message reaches $m + 1$ processes. Note that the $m + 1$ processes the message reaches overlaps the set of $(n - m)$ correct processes every other process hears from in the next round, allowing information to propagate to all correct processes in two rounds. Thus, a single pre-commit phase suffices to eliminate races as described above, where two different values are repeatedly committed after GSR.

We now describe $p_i$’s computation. If $p_i$ does not decide, it evaluates the following two conditions: pre-commit (line 23): $p_i$ receives messages from a majority of processes with $maxEST_i$ as their estimate; and commit (line 24): at least one COMMIT or PRE-COMMIT message is received with $maxEST_i$. If both conditions are true, then $p_i$ sets its message type (for the round $k_i + 1$ message) to COMMIT, adopts the estimate $maxEST_i$, and sets its timestamp to the current round number $k_i$ (line 25). We say that $p_i$ commits in round $k_i$ with estimate $maxEST_i$. If, however, only the first condition holds, then $p_i$ sets its message type to PRE-COMMIT, adopts the estimate $maxEST_i$, and sets its timestamp to $maxTS_i$ (line 27). We say that $p_i$ pre-commits in round $k_i$ with estimate $maxEST_i$. If neither condition holds, $p_i$ prepares (sets his

Algorithm 3: Majority-based algorithm for $\Diamond AFM$ model. Code for process $p_i$. Optimization for $n = 2m + 1$ is marked in gray.
Correctness. A process may commit with different estimates in different rounds. However, we show (in Appendix B) that starting from a round $k$ in which a majority of processes $M$ commit with some estimate $x$ onward, every commit is with estimate $x$. Note that this implies agreement, since decision is impossible before a majority of processes commit (see decision rules). To understand why this is true, note first that by rule pre-commit, all COMMIT and PRE-COMMIT messages sent in the same round are with the same estimate. This explains why a commitment with $y \neq x$ is impossible in round $k$. Additionally, note that a process’s timestamp never decreases, and therefore the processes in $M$ have timestamps $\geq k$ in subsequent rounds. Suppose that a process $p_i$ commits in round $k' > k$. Rule pre-commit ensures that $p_i$ hears from a majority. Since every two majorities intersect, $p_i$ hears from at least one process in $M$. Since $p_i$ commits on $\text{max}EST_i$, which has the maximal timestamp, $p_i$ commits with a timestamp $\geq k$. Using an inductive argument, we get that $\text{max}EST_i = x$. Since no decision is made before a majority commits, and every decision is either on the value of a previous decision (rule decide-1), or on the value sent in COMMIT messages (rule decide-2), which equals $x$ from round $k$ onward, all decisions are with $x$.

Performance. We now explain why the algorithm decides by round GSR+5 (a formal proof appears in Appendix B). First, if some process decides by round GSR+3, then its decide message reaches every process by the end of round GSR+5. Assume no process decides by GSR+3. Second, if no process commits in round GSR, the maximum timestamp sent in GSR is the same as the maximum timestamp sent in round GSR+1, and it reaches every correct process by the end of round $k_1 = \text{GSR}+1$, at which point all processes have the same $\text{max}EST$. Finally, if a process commits in GSR, the use of pre-commit ensures that no different value is committed in GSR+1, and thus this value has the highest timestamp among those sent in round GSR+2, and this timestamp and its estimate reach every process by the end of round $k_2 = \text{GSR}+2$. In both cases, every process has the same $\text{max}EST$ at the end of round $k = k_1$ or $k = k_2$. Thus, all processes send the same estimate in round $k+1$, and in the ensuing round, a majority of processes receives it and pre-commits (at least). In round $k+2$, every correct process receives the same estimate from majority and a PRE-COMMIT or COMMIT message, and commits. Finally, by round $k+3$, which is at most GSR+5, every process decides by rule decide-2.

Optimization for $n = 2m+1$. We present an optimization of Algorithm 3 for the case of $n = 2m+1$ (i.e., when both $(m+1)$ and $(n-m)$ are majorities). The additional code used for the optimization is marked in gray in Algorithm 3. In Appendix B, we prove that the optimized algorithm reaches global decision by round GSR+4 (five rounds) for $n = 2m+1$ and by round GSR+5 (six rounds) for other values of $f \leq m < n/2$.

The optimization relies on the IgotCommit and gotCommit variables, that are used for “gossiping” about COMMIT messages. Whenever a process receives a COMMIT message, it indicates this in its next round message by setting IgotCommit to true. In order to have all processes learn about commits, we use the gotCommit message field. A process includes in the gotCommit set that it sends in round $k+1$, all processes that it knows have gotten COMMIT messages in round $k-1$ (based on IgotCommit indications sent in round $k$). Thus, in round $k+1$, the incoming gotCommit sets from different processes can give $p_i$ a better picture about which processes got COMMIT messages in round $k-1$. In Appendix A, we prove that if the union of the gotCommit groups that a process gets exceeds $\lfloor n/2 \rfloor$, it is safe for the process to decide on $\text{max}EST$ (rule decide-3) and this optimization allows us to speed up global decision to be by round GSR+4 instead of by round GSR+5. We formally prove the correctness of the optimized Algorithm 3 in Appendix B.
8 Impossibility of Bounded Time Global Decision in \( \Diamond MFM \)

We define the \( \Diamond MFM \) family of models, for \( m \in \mathbb{N}^+, f \leq m < n/2 \), as follows:

\( \Diamond MFM(m) \) \((\text{Majority-From-Majority})\): \( t < n/2 \), reliable links, every correct process is a \( \Diamond (n - m) \)-source and \( \Diamond m \)-accessible, \( m \) correct processes are \( \Diamond n \)-sources, and \( (n - m) \) correct processes are \( \Diamond (n - m) \)-accessible.

Note that these models are only slightly weaker than \( \Diamond A FM \), where we have shown that constant-time decision is attainable. We show that the time for global decision after GSR in all of these models is unbounded.

**Lemma 5.** For any \( f \leq m < n/2 \) \((m \in \mathbb{N}^+)\), there exists no consensus algorithm that reaches global decision in bounded time from GSR in \( \Diamond MFM(m) \).

**Proof.** Assume by contradiction that an algorithm \( A \) reaches global decision by round \( GSR(r) + T_A \) in every run \( r \). We partition the processes into three groups: a group \( P \) of \( m \) processes, a group \( Q \) of \( m \) processes, and a group \( R \) of the remaining \( n - 2m \) \((\geq 1\), since \( m < n/2 \)) processes.

We construct three runs in \( \Diamond MFM(m) \), in which no process fails \((f = 0)\), and processes of each group have perpetually timely bidirectional links to all other processes of the same group. For each run we state which inter-group links are \( \Diamond \)timely. These links are timely only from GSR onward, and delay until round GSR all messages sent before that round.

Each one of the three runs is a run in \( \Diamond MFM(m) \): in every run, either groups \( Q \) and \( R \) or groups \( P \) and \( R \) are fully connected with timely links from GSR onward. The number of processes in the resulting group is \( n - m \). Therefore, in every run there are \( n - m \) processes that are \( \Diamond (n - m) \)-accessible (and therefore \( \Diamond (n - m) \)-source and \( \Diamond m \)-accessible, since \( n - m > m \)). The other \( m \) processes are correct and fully interconnected with timely links from the start, i.e. \( m \)-accessible, and have \( \Diamond \)timely outgoing links to every process, i.e. \( m \) correct \( \Diamond n \)-source processes. Therefore, the requirements of the model are fulfilled in each one of the three runs below.

We construct a run \( \sigma_0 \) in which from round \( GSR(\sigma_0) = 1 \) onward (i) processes of \( P \) have timely outgoing links to all other processes, and (ii) processes of \( Q \) and \( R \) have timely links among them. All other links between groups deliver messages only after round \( GSR(\sigma_0) + T_A = T_A + 1 \). All processes propose 0. Since algorithm \( A \) always reaches global decision by round \( GSR(r) + T_A \), processes of \( P \) decide 0 (by validity) by round \( T_A + 1 \).

We next construct a run \( \sigma_1 \) in which from round \( GSR(\sigma_1) = T_A + 2 \) onward (i) processes of \( Q \) have timely outgoing links to all other processes, and (ii) processes of \( P \) and \( R \) have timely links among them. All other links between groups deliver messages only after round \( GSR(\sigma_1) + T_A \). All processes propose 1. Since algorithm \( A \) always reaches global decision by round \( GSR(r) + T_A \), processes of \( Q \) decide 1 (by validity) by round \( GSR(\sigma_1) + T_A \).

Finally, we construct a run \( \sigma_2 \) in which, like in \( \sigma_1 \), starting from round \( GSR(\sigma_2) = T_A + 2 \) onward (i) processes of \( Q \) have timely outgoing links to all other processes, and (ii) processes of \( P \) and \( R \) have timely links among them. All other links between groups deliver messages only after round \( GSR(\sigma_2) + T_A \). In \( \sigma_2 \), processes of \( Q \) propose 1 and processes of \( P \) and \( R \) propose 0. Note that processes of \( Q \) decide 1 since they cannot distinguish this run from \( \sigma_1 \). Processes of group \( P \) cannot distinguish \( \sigma_2 \) from \( \sigma_0 \) by round \( T_A + 1 \) and hence decide 0, violating agreement. A contradiction.

Note that our notion of timely links is more abstract than the real-time-based definition used in \([2, 3, 27]\), where messages arrive within bounded latency. Nevertheless, since we never explicitly reason about time duration in constructing our runs, our proof is applicable even if all messages on timely links in these runs are delivered within bounded latency, and hence covers these models.
9 Conclusions and Future Directions

We have focused on the question of which timeliness or failure detector guarantees one should attempt to implement in a distributed system. While it is obvious that weaker timeliness/failure detector guarantees can be practically satisfied using shorter timeouts and cheaper hardware than stronger ones, it was not previously established what implications the use of weaker properties has on algorithm performance. Although from a theoretical perspective it is interesting to discover the weakest conditions that can be used to ensure eventual decision, in practice, timely decision is of essence. System designers are often willing to spend more on hardware, if this can ensure better performance. Likewise, implementations are better off using longer timeouts if this can lead to faster decision overall.

We have presented a general framework GIRAF, to answer such questions. GIRAF does not restrict the set of allowed algorithms, models or failure patterns, but rather organizes algorithms in a “round” structure, which allows for analyzing their complexity. We used our framework to show that some previously suggested guarantees were too weak to solve consensus in a timely manner. We have further shown that it is possible to strengthen a model in which consensus is not solvable in bounded time (♦MFM(m) for n = 2m + 1) to get a model in which consensus is solvable in constant time (♦AFM) by adding just one ♦timely incoming link per process, for a minority of processes. In such situations, it is worthwhile to increase timeouts and/or buy faster hardware in order to implement stronger guarantees. On the other hand, we have shown that the strong ES model (which requires timely communication among all pairs of correct processes) can be weakened in ways that are significant from a performance standpoint (as shown in [4, 5]), and yet with little (for ♦AFM) or no (for ♦LM) penalty on performance of the consensus algorithm.

We believe that GIRAF has the potential to further enhance the understanding of performance tradeoffs between different models, and opens vast opportunities for future work. We now point out several exemplar directions for future research.

- One can use our new notion of α-reducibility (and k-round reducibility) to compare various models more meaningfully than with the classical notion of reducibility, by considering the time (round) complexity of the reduction.

- While this paper focuses on the performance of the algorithm after synchronization, an important complementary direction for future study is understanding the performance of the environment’s synchronization mechanism, that is, the actual time it takes to reach GSR in various timing models. Whereas GIRAF provides generic analysis of the cost of algorithms in terms of different round-types (e.g., all-to-all communication in each round or communication with a majority of processes), in order to deduce which algorithm is best for a given network setting, this analysis should be complemented with a measurement study of the cost of rounds of different types in that specific setting.

- It would be interesting to further study the fine line between models that allow bounded and unbounded decision times. For example, is it possible to weaken ♦AFM by making fewer processes ♦(m + 1)-sources, and still achieve constant or bounded time consensus? and what would be the effect of weakening the assumption that the leader is a ♦n-source in ♦LM, on consensus performance?

- In this paper, we have focused on global decision. It can be interesting to investigate local consensus decision [16], i.e., the number of rounds until some process decides.

- Finally, there are gaps between upper and lower bounds shown in Table 1, which might be closed.
Acknowledgments
We thank Marcos Aguilera, Partha Dutta, Rachid Guerraoui, Eshcar Hilel, Denis Krivitski, Keith Marzullo, Yoram Moses, and Neeraj Suri for many helpful discussions.

References


A Correctness of Algorithm 2

Lemma 6. A process’s timestamp at the start of round $k$ is less than $k$.

Proof. We prove the claim by induction on the round number $k'$. Base case: $k' = 1$. The claim is correct since a process’s timestamp is initialized to 0. The induction hypothesis is that the claim holds up to round $k'$. Let us inspect the possible actions of processes at the end of round $k'$. A process can decide and in this case its timestamp does not change and in round $k' + 1$ it will remain less or equal to $k' - 1$, by the induction hypothesis. Alternatively, a process may commit, and then (on line 24) it will adopt $k'$ as its new timestamp for round $k' + 1$, and the claim holds here as well. Finally, a process may adopt the timestamp of a message it received in round $k'$ (lines 26-27) and again, by induction hypothesis, the claim is true (since communication rounds are closed).

Lemma 7. A process’s timestamp is non-decreasing.

Proof. Observe that when a process decides, its timestamp does not change. It does not change in the following rounds as well. If a process $p_i$ does not decide in round $k$, then it can change its timestamp by adopting either $k$ (when committing on line 24) or the maximum timestamp received in round $k$ as its new timestamp (lines 26-27). Since $p_i$ receives its own message in round $k$, the latter is not lower than its current timestamp (recall that communication rounds are closed). In case it commits, since according to Lemma 6, its old timestamp cannot exceed $k - 1$, by adopting $k$ it can only increase.

Lemma 8. For every round $k$, no two processes commit with different estimates in round $k$, and no two processes commit with different newLDs in round $k$.

Proof. Consider two processes $p_i$ and $p_j$ that commit in round $k$ with estimates $est_i$ and $est_j$, and leader values $newLD_i$ and $newLD_j$, respectively. Also, in round $k$, let $prevLD_i$ be the leader of $p_i$ and $prevLD_j$ be the leader of $p_j$. From commit-1, each of them has received in round $k$ a majority of messages that contain $prevLD_i$ and $prevLD_j$ as leader, respectively. As two majorities intersect, $prevLD_i = prevLD_j$. Furthermore, from commit-3, $newLD_i = prevLD_i$ and $newLD_j = prevLD_j$. So, $newLD_i = prevLD_i = prevLD_j = newLD_j$. From the algorithm, $p_i$ commits with the estimate sent by $prevLD_i$, and $p_j$ commits with the estimate sent by $prevLD_j$. As $prevLD_i = prevLD_j$, $p_i$ and $p_j$ commit with the same estimate.

Lemma 9. If some process sends a PREPARE or COMMIT message with timestamp $ts > 0$ and estimate $x$ then some process commits in round $ts$ with estimate $x$.

Proof. We prove the claim by induction on the round number $k'$, starting from a round $k_0$ in which a message with the timestamp $ts$ was first sent with some estimate $x'$, by some process $p_j$.

Base Case. $k' = k_0$. From the definition of $k_0$, $p_j$ could not receive a message with $ts$ from another process in an earlier round. Thus, $p_j$ commits with timestamp $ts$ and estimate $x'$ in round $k_0 - 1$, and from the algorithm, $k_0 - 1 = ts$.

Induction Hypothesis. If any process sends a PREPARE or COMMIT message in round $k_1$, such that $k_0 \leq k_1 \leq k'$, with timestamp $ts$ and some estimate $x''$, then some process commits in round $ts$ with estimate $x''$.

Induction Step. We need to show that if, in round $k' + 1$, a process sends a PREPARE or COMMIT message with timestamp $ts$ and some estimate $x''$ then some process commits in round $ts$ with estimate $x''$. Observe,
that if a COMMIT message is sent, it would have a timestamp equal to the previous round number \( k' \), and since \( ts = k_0 - 1 < k' \) (by the base case), this case is not possible. Observe that if a PREPARE message is sent in round \( k' + 1 \) with timestamp \( ts \) and estimate \( x'' \), the sending process must have adopted the timestamp together with the estimate from some PREPARE or COMMIT message sent in round \( k' \). By the induction hypothesis, we get that some process commits in round \( ts \) and estimate \( x'' \).

Please note that the claim in Lemma 9 does not hold for DECIDE messages, since a process decides adopting only the estimate and not the associated timestamp from another DECIDE message.

**Lemma 10.** If a process \( p_i \) decides in round \( k \) by rule decide-2 on estimate \( x \), then every process that commits in round \( k \), commits with estimate \( x \).

**Proof.** Suppose for the purpose of contradiction that a process \( p_j \) that commits with \( y \neq x \) in round \( k \). Since \( p_j \) does not decide in round \( k \), it evaluates rules decide-1 and decide-2 to false. \( p_j \) commits the estimate that it receives from its leader (line 24). We denote this leader by \( p_l \). By rule commit-1, there is a majority of processes that send a round \( k \) message with \( p_l \) as leader. Let us denote this majority by \( M_1 \). Observe process \( p_i \) that decides in round \( k \). \( p_i \) receives a COMMIT message from a majority of processes, including its leader \( prevLD_1 \) and itself. We denote this majority by \( M_2 \). By Lemma 8 every COMMIT message sent by a process in \( M_2 \) has the same leader field and the same estimate \((x)\). Since \( M_1 \) and \( M_2 \) intersect (as every two majorities), the leader field indicates \( p_l \). Since \( p_i \) receives a COMMIT message from itself, it as well sends a round \( k \) message with \( p_l \) as leader. Since what \( p_i \) actually sent is now in his \( prevLD_1 \) variable, we get that \( prevLD_i = p_l \). Since \( p_i \) receives a message \( M_i[k][prevLD_i] \) with \( msgType = COMMIT \), we conclude that \( p_l \) sends a COMMIT message in round \( k \) and as was explained, this means that its message includes \( x \) as the estimate. This contradicts our assumption that \( p_j \) sees an estimate \( y \neq x \) sent by \( p_l \).

**Lemma 11 (Uniform Agreement).** No two processes decide differently.

**Proof.** Let \( k \) be the lowest numbered round in which some process decides. Suppose \( p_i \) decides \( x \) in round \( k \). Since no process decides in an earlier round, \( p_i \) decides by rule decide-2. Therefore, \( p_i \) receives a majority of COMMIT messages in round \( k \), and it decides on the estimate of one of the COMMIT messages (the one from itself). From Lemma 8, all COMMIT messages include the same estimate and leader, say \( p_l \).

Thus \( p_i \) receives a round \( k \) message of the form \( \langle COMMIT, x, k-1, pl, * \rangle \) from a majority of processes, and hence, a majority of processes commits in round \( k-1 \) with estimate \( x \). Let us denote this majority of processes by \( S_x \). Note that \( k-1 \geq 1 \) since according to the pseudo-code, the first round of the algorithm is round number 1. We claim that if any process commits or decides in round \( k' \geq k-1 \) then it commits or decides \( x \). The proof is by induction on round number \( k' \).

**Base Case.** \( k' = k-1 \). As processes in \( S_x \) commit \( x \) in round \( k-1 \), from Lemma 8, no process commits with an estimate different from \( x \) in round \( k-1 \). By definition of \( k \), no process decides in round \( k-1 \).

**Induction Hypothesis.** If any process commits or decides in any round \( k1 \) such that \( k-1 \leq k1 \leq k' \), then it commits with estimate \( x \) or decides \( x \).

**Induction Step.** decision in round \( k' + 1 \). If some process \( p \) decides in round \( k' + 1 \), then in that round either some other process sends a DECIDE message with decision value \( y \) or \( p \) sends a COMMIT message with estimate \( y \). In both cases, by the induction hypothesis, \( y = x \).

**commit in round \( k' + 1 \).** Suppose by contradiction that some process \( p_j \) commits in round \( k' + 1 \) with estimate \( z \neq x \). First, since \( p_j \) decides by rule decide-2 in round \( k \), by Lemma 10 we have that \( k' + 1 \neq k \). Since we know by the induction hypothesis that \( k' \geq k-1 \) we now get that \( k' > k-1 \). Since \( p_j \) commits,
it does not receive any DECIDE message in round \( k' + 1 \). Since commit-2 evaluated to true for \( p_j \), a message \( m = (\text{type} \neq \text{DECIDE}, z, ts_z, ld, k') \) was received by \( p_j \) in round \( k' + 1 \) from the leader \( ld \). Notice that \( ts_z \) might be different than \( \max TS_i \) of round \( k' + 1 \).

Observe the lastApproval field of the message \( m \). Its value is \( k' \). Since \( k' > k - 1 \geq 1 \) we get that \( k' > 1 \). Since the lastApproval field can become greater than 0 only on line 17 of the compute() function, this indicates that the leader received a message from a majority of processes in round \( k' \), and therefore it must have heard from at least one process \( p_a \in S_x \). Recall that every process in \( S_x \) commits in round \( k - 1 \) with estimate \( x \). Thus \( p_a \) has timestamp \( k - 1 \) at the end of round \( k - 1 \). From Lemma 7, since \( k' > k - 1 \), \( p_a \)'s timestamp is at least \( k - 1 \).

If type = COMMIT, this means that \( ts_z = k' \) (line 24). As was explained, \( k' > 1 \), and by Lemma 9 we get that some process commits in round \( k' \) with estimate \( z \neq x \). This is a contradiction to the induction hypothesis. If type = PREPARE, it means that \( ts_z \) is the maximum timestamp the leader received in any message of round \( k' \) (lines 26-27). Because it received a message from \( p_a \) and because, according to Lemma 6, the highest timestamp that can be received in round \( k' + 1 \) is \( k' \), we get that \( k - 1 \leq ts_z \leq k' \), and since (by Lemma 9) there must be a process that commits in round \( ts_z \) with estimate \( z \neq x \) (recall that \( k - 1 > 0 \)), this is a contradiction to the induction hypothesis.

**Lemma 12.** In every run \( r \), all correct processes decide by round \( GSR(r) + 2 \).

**Proof.** Observe that in our model every correct process executes an infinite number of rounds, and in particular, executes round \( GSR(r) + 2 \). We prove the lemma by contradiction. Assume that some correct process \( p_j \) does not decide by round \( GSR(r) + 2 \) in some run \( r \). Therefore, \( p_j \) couldn’t have received a COMMIT message from a majority of processes (including from itself and the leader) in round \( GSR(r) + 2 \). Since, in our model, from \( GSR(r) \) onward, every correct process receives messages from a majority of correct processes (including itself and the leader), it must have received at least one message with type \( t \) other than COMMIT. \( t \) cannot be DECIDE, since \( p_j \) didn’t decide in round \( GSR(r) + 2 \). Therefore, \( t \) must be PREPARE. Therefore, there must be a process \( p_i \) that sent in round \( GSR(r) + 2 \), a message with \( msgType = \text{PREPARE} \).

Let us now observe round \( GSR(r) + 1 \). \( p_i \) couldn’t have decided, and couldn’t have committed in this round since it sent a PREPARE message in the next round. Therefore, one of the commit rules must have been evaluated to \( false \) for \( p_i \). It couldn’t have been commit-1 or commit-3, because all correct processes agree on the identity of the leader from \( GSR(r) \) onward, and each process receives a message from a majority of processes (commit-1), and starting from round \( GSR(r) + 1 \), the rule commit-3 evaluates to \( true \) as was explained in the description of the algorithm.

Therefore, commit-2 must have been the rule that evaluated to \( false \). The only possible reason for this is that the leader indicated \( \text{lastApproval} \neq GSR(r) \) in its round \( GSR(r) + 1 \) message. If \( GSR(r) = 0 \) we get a contradiction since \( \text{lastApproval} \) is initialized to 0. Otherwise \( (GSR(r) > 0) \), notice that the leader couldn’t have decided by start of round \( GSR(r) \), since otherwise all correct processes would decide by end of \( GSR(r) \). Therefore, according to our algorithm, the leader had to set \( \text{lastApproval} = GSR(r) \) in round \( GSR(r) \) (since every process hears from a majority starting at round \( GSR(r) \)), and this is a contradiction.

}\]
Theorem 13. The algorithm solves consensus by round GSR(r) + 2.

Proof. From Lemma 12, every correct process decides by round GSR(r) + 2. Validity holds, since the decision can only be one of the initial estimates of the processes. Uniform agreement was proved in Lemma 11.

B Correctness of Algorithm 3

Lemma 14. A process's timestamp at the start of round k is less than k.

Proof. We prove the claim by induction on the round number k'. Base case: k' = 1. The claim is correct since a process's timestamp is initialized to 0. The induction hypothesis is that the claim holds up to round k'. Let us inspect the possible actions of a process at the end of round k'. A process can decide and in this case its timestamp does not change and in round k' + 1 it will remain less or equal to k' - 1, by the induction hypothesis. Alternatively, a process may commit, and then (on line 25) it will adopt k' as its new timestamp for round k' + 1, and the claim holds here as well. Finally, a process may adopt the timestamp of a message it received in round k' (on line 27 or 29) and again, by induction hypothesis, the claim is true (since communication rounds are closed).

Lemma 15. A process's timestamp is non-decreasing.

Proof. Observe that when a process decides, its timestamp does not change. It does not change in the following rounds as well. If a process p_i does not decide in round k, then it can change its timestamp by adopting either k (when committing on line 25) or the maximum timestamp received in round k as its new timestamp (on line 27 or 29). Since p_i receives its own message in round k, the latter is not lower than its current timestamp (recall that communication rounds are closed). In case it commits, since according to Lemma 14, its old timestamp cannot exceed k - 1, by adopting k it can only increase.

Lemma 16. For every round k, no two processes commit or pre-commit with different estimates in round k.

Proof. Consider two processes p_i and p_j that commit or pre-commit in round k with estimates est_i and est_j. Thus, by pre-commit rule, each of them has received in round k a majority of messages that contain est_i and est_j, respectively. As two majorities intersect, est_i = est_j. Therefore, p_i and p_j commit or pre-commit with the same estimate.

Lemma 17. If some process sends a message other than DECIDE with timestamp ts > 0 and estimate x, then some process commits in round ts with estimate x.

Proof. We prove the claim by induction on the round number k', starting from a round k_0 in which a message other than DECIDE with the timestamp ts is first sent with some estimate x' by some process p_j.

Base Case. k' = k_0. From the definition of k_0, p_j could not receive a message with ts from another process in an earlier round. Thus, p_j commits with timestamp ts and estimate x' in round k_0 - 1, and from the algorithm, k_0 - 1 = ts.

Induction Hypothesis. If any process sends a PREPARE, PRE-COMMIT or COMMIT message in round k_1, such that k_0 ≤ k_1 ≤ k', with timestamp ts and some estimate x'', then some process commits in round ts with estimate x''.

23
**Induction Step.** We need to show that if, in round $k' + 1$, a process sends a message other than DECIDE with timestamp $ts$ and some estimate $x''$, then some process commits in round $ts$ with estimate $x''$. Observe, that if a COMMIT message is sent, it has a timestamp equal to the previous round number $k'$, and since $ts = k_0 - 1 < k'$ (from the base case), this case is not possible. Observe that if a PREPARE or PRE-COMMIT message is sent in round $k' + 1$ with timestamp $ts$ and estimate $x''$, the sending process must have adopted the timestamp together with the estimate from some PREPARE, PRE-COMMIT, or COMMIT message sent in round $k'$ (this message couldn’t have been DECIDE since otherwise the $k' + 1$ round message would be DECIDE and not PREPARE). By the induction hypothesis, we get that some process commits in round $ts$ and estimate $x''$.

Please note that the claim in Lemma 17 does not hold for DECIDE messages, since a process can decide adopting only the estimate and not the associated timestamp from another DECIDE message.

**Lemma 18.** If rule decide-3 evaluates to true in some round $k$, there exists a majority of processes that receive a COMMIT message in round $k - 2$.

**Proof.** Suppose rule decide-3 evaluates to true in some round $k$ at process $p_i$. Therefore, the union of the gotCommit sets $p_i$ receives in round $k$ messages includes more than $\lceil (n/2) \rceil$ indices. These gotCommit groups were created in round $k - 1$ by the processes that sent these messages, according to the IgotCommit values that these processes received. The fact that the union of the gotCommit groups has size $\lceil (n/2) \rceil$ indicates that more than $\lceil (n/2) \rceil$ messages were received in round $k - 1$ with IgotCommit = true from different processes. A process sends a message with IgotCommit = true only when it receives a COMMIT message in the previous round. Therefore, more than $\lceil (n/2) \rceil$ (a majority) of processes received a COMMIT message in round $k - 2$.

**Lemma 19.** (a) If some process receives a COMMIT message in round $k$ with estimate $x$, and some process commits in round $k$ with estimate $z$, then $z = x$ (b) if a process $p_i$ commits to $x$ in round $k$, or receives a COMMIT message with estimate $x$ in round $k$, and does not decide in this round, then it adopts $x$ as its estimate with timestamp $ts \geq k - 1$.

**Proof.** (a) If some process commits with estimate $z$ in round $k$, it must have received a COMMIT or PRE-COMMIT message with $z$ (rule commit), and according to Lemma 16, all such messages have the same estimate, and therefore $z = x$. (b) If $p_i$ commits $x$, then it sets its timestamp to $k$ and adopts $x$ as its estimate. If $p_i$ receives a COMMIT message with estimate $x$, it cannot commit or pre-commit on a different value since according to rule pre-commit a process can commit or pre-commit only on a value received with the highest timestamp. Moreover, $p_i$ receives $x$ with the timestamp $k - 1$ (which is maximal at round $k$) and (Lemma 16) every message with this timestamp has $x$ as estimate. Since it does not commit on $x$ either, it does not commit at all in round $k$. Since $p_i$ does not decide in this round, it must either pre-commit or prepare with the estimate $x$ and adopt its timestamp: $k - 1$.

**Lemma 20 (Uniform Agreement).** Let $k$ be the first round in which there exists a group consisting of a majority of processes such that each process of the group either commits or receives a COMMIT message. Then, no decision is made before round $k + 1$, and all decisions and commitments made in rounds $k' \geq k - 1$ are with the same estimate.

**Proof.** Let $k$ be the lowest numbered round in which each one out of a majority of processes either commits $x$ (from Lemma 16 all commitments in some round are with the same value) or receives a COMMIT message with a value $x$ ($x$ is well defined according to Lemma 19). Denote this group of processes by $S_x$. According to Lemma 19, every process in $S_x$ has timestamp $\geq k - 1$ at the end of round $k$ (we prove below that a decision is not possible in round $k$). Note also, that $k - 1 > 0$. This is true since by definition of $k$, processes
in \( S_x \) either commit or receive a COMMIT message in round \( k \). Therefore, in round \( k - 1 \) some process must either commit or pre-commit and since round numbering starts from 1, we have that \( k - 1 > 0 \).

There are three decision rules in the algorithm. We show that none of them could evaluate to true for any process before round \( k + 1 \). Let \( k' \) be the first round in which any process decides. Rule decide-1 may be true only after some process has already decided, and thus cannot cause the first decision. Rule decide-2 can evaluate to true only if a majority of processes committed in the previous round. Since the first round in which this happens is \( k \), this rule may evaluate to true only starting from round \( k + 1 \). The last one is rule decide-3. According to Lemma 18, if this rule evaluates to true in round \( k' \), there must have been a majority of processes that received a COMMIT message in round \( k' - 2 \). Since the first round in which this could happen is \( k \), we get that \( k' \geq k + 2 \). So in any case, no decision is possible before round \( k + 1 \). We now prove that all decisions and commitments made in rounds \( k' \geq k - 1 \) are with estimate \( x \).

**Base Case.** \( k' = k - 1 \). As proven above, no process decides in round \( k' < k + 1 \). Assume by contradiction that some process commits on a value \( z \neq x \) in round \( k - 1 \). By Lemma 16, no process can commit or pre-commit on \( x \) in the same round. Therefore, in round \( k \), no process receives a COMMIT or PRE-COMMIT message with the estimate \( x \). Thus, no process commits \( x \) in round \( k \) (rule commit). This contradicts the definition of \( k \).

**Induction Hypothesis.** If any process commits or decides in any round \( k' \) such that \( k - 1 \leq k' \leq k' \), then it commits with estimate \( x \) or decides \( x \).

**Induction Step.** decision in round \( k' + 1 \). Suppose some process \( p \) decides in round \( k' + 1 \). If it decides using rule decide-1 or decide-2, then in that round either some other process sends a DECIDE message with decision value \( y \) or \( p \) sends a COMMIT message with estimate \( y \). In both cases, by the induction hypothesis, \( y = x \).

If it decides by rule decide-3, then according to Lemma 18, there must be a majority of processes that receive COMMIT messages two rounds earlier, in round \( k' - 1 \). Since the first round in which this can happen is \( k \), we have that \( k' - 1 \geq k \). According to the induction hypothesis, the commit messages received are with estimate \( x \). Therefore, in round \( k' - 1 \), some majority of processes \( M \) received a COMMIT message with the estimate \( x \). According to Lemma 19, if a process in \( M \) does not decide in round \( k' - 1 \), it will adopt \( x \) with timestamp \( \geq k' - 2 \). By the induction hypothesis, every process that decides in round \( k' - 1 \) or \( k' \), decides \( x \) and no process commits with a different value in round \( k' - 1 \) or \( k' \). Therefore, in round \( k' \), all estimates different from \( x \) are sent with a timestamp \( < k' - 2 \). No process can commit or pre-commit on an estimate other than \( x \) in round \( k' \) since \( x \) is the value processes in \( M \) send and every two majorities intersect (rule pre-commit must be false for any other value).

\( p \) receives a round \( k' + 1 \) message from at least one process \( p_i \) that receives a round \( k' \) message from some process in \( M \). Therefore, \( p_i \) receives a round \( k' \) message with the estimate \( x \) and timestamp \( \geq k' - 2 \). As was explained above, no other estimate can have a timestamp that high in round \( k' \), so if \( p_i \) prepares or pre-commits, it must be with estimate \( x \). If \( p_i \) commits, it is with the estimate \( x \) as well, according to the induction hypothesis. \( p_i \) does not decide, since otherwise \( p \) would decide by rule decide-1 and not decide-3. Therefore, \( p_i \) sends a round \( k' + 1 \) message with the estimate \( x \) and a timestamp \( \geq k' - 2 \). Since no process can commit on a value different than \( x \) in round \( k' \) or \( k' - 1 \), this timestamp is higher than the timestamp of any other estimate sent in round \( k' + 1 \). Therefore maxEST of \( p \) must be equal to \( x \). Therefore, \( p \) decides \( x \).

**Commit in round \( k' + 1 \).** Suppose by contradiction that some process \( p_j \) commits in round \( k' + 1 \) with estimate \( z \neq x \). Then \( p_j \) does not receive any DECIDE message in round \( k' + 1 \). Also note that according to rule pre-commit, \( p_j \) commits on an estimate that it receives with the highest timestamp: maxTS. Therefore,
some process sends a round \( k' + 1 \) message with timestamp \( \max TS \) and estimate \( z \). By Lemma 14, the highest timestamp that can be received in round \( k' + 1 \) is \( k' \), and therefore \( \max TS \leq k' \). Since \( p_j \) commits in round \( k' + 1 \), it receives round \( k' + 1 \) messages from a majority of process (rule \textit{pre-commit}) and hence, receives a round \( k' + 1 \) message from at least one process \( p_i \in S_x \). According to Lemma 19, \( p_i \) has at least timestamp \( k - 1 \) at the end of round \( k \). By Lemma 15, \( p_i \)'s timestamp is at least \( k - 1 \) and therefore \( \max TS \geq k - 1 \). Thus, we have \( k - 1 \leq \max TS \leq k' \). Since \( k - 1 > 0 \) (as shown above), and since \( p_j \) does not receive any \textit{DECIDE} messages in round \( k' + 1 \), by Lemma 17 there is a process that commits \( z \) in round \( \max TS \). By the induction hypothesis, every process that commits in round \( \max TS \) commits \( x \neq z \); a contradiction.

We now turn to prove the performance guarantees of Algorithm 3.

\textbf{Lemma 21.} If \( n = 2m + 1 \), in every run \( r \), if some process \( p \) commits in round \( GSR(r) \) with an estimate \( x \), then all processes decide by the end of round \( GSR(r) + 3 \).

\textit{Proof.} Suppose that some process \( p \) does not decide by the end of round \( GSR(r) + 3 \). This means that it evaluates rules \textit{decide-1}, \textit{decide-2} and \textit{decide-3} to \textit{false}. Therefore, \( p \) does not receive any \textit{DECIDE} message, and \(|\bigcup_{j \in M} M[k_i][j].\text{gotCommit} \mid \leq \lfloor n/2 \rfloor \). \( p_i \) receives a round \( GSR(r) + 3 \) message from a group \( M \) of \((n - m)\) processes. Thus, there are \((n - m)\) processes that together receive in round \( GSR(r) + 2 \) messages with \( I\text{gotCommit} = \textit{true} \) from at most \(|n/2| \) processes. Every process in \( M \) does not receive a \textit{DECIDE} message in round \( GSR(r) + 2 \), since otherwise their next round message would be \textit{DECIDE}. Since each process’s message reaches \((m + 1)\) from \( GSR(r) \) onward, it reaches at least one process from any group of \((n - m)\) processes. Therefore, the number of processes that send a message with \( I\text{gotCommit} = \textit{true} \) in round \( GSR(r) + 2 \) is at most \(|n/2| \). We get that in round \( GSR(r) + 1 \), at most \(|n/2| \) processes received a \textit{COMMIT} message. Every process whose message reaches a process in \( M \) does not decide in round \( GSR(r) + 1 \), since otherwise their next round message would be \textit{DECIDE} and processes in \( M \) do not receive any such messages. Since \( p \) sends a \textit{COMMIT} message, its message should reaches at least \( m + 1 \) processes (a majority when \( n = 2m + 1 \)). As explained above, we get a contradiction the assumption that \( p \) sends a \textit{COMMIT} message in round \( GSR(r) + 1 \). \hfill \Box

\textbf{Notations:} (relating to a specific run \( r \))

\begin{align*}
\text{absMaxTS}(k) &= \max\{m.ts \mid \text{message } m \text{ is sent in round } k\} \\
\text{absMaxEST}(k) &= \max\{m.estr \mid \text{message } m \text{ is sent in round } k \text{ s.t. } m.ts = \text{absMaxTS}(k)\}
\end{align*}

\textbf{Lemma 22.} In every run \( r \), if no correct process decides by the end of round \( k \geq GSR(r) \), then in round \( k \) at least \( m + 1 \) correct processes adopt the estimate \( \text{absMaxEST}(k) \) with timestamps equal to \( \text{absMaxTS}(k) \) or to \( k \) (in case it is adopted by committing).

\textit{Proof.} Let us observe round \( k \) messages.

Denote by \( p_{\text{max}} \) the (correct) process that sends \( \text{absMaxEST}(k) \) with the timestamp \( \text{absMaxTS}(k) \). By the assumptions of our model, the round \( k \) message \( \langle *, \text{absMaxEST}(k), \text{absMaxTS}(k), *, * \rangle \) will reach at least \( m + 1 \) correct processes. Denote the group of processes that actually get this message by \( A \). Note that since a process receives a subset of all messages sent in round \( k \), for any \( p_i \in A, \text{maxTS}_i = \text{absMaxTS}(k) \) and \( \text{maxEST}_i = \text{absMaxEST}(k) \). The conditions of the lemma assume that no correct process decides by the end of round \( k \). Therefore, each process \( p_i \in A \) must commit, pre-commit or just prepare for the next round. If \( p_i \) commits or pre-commits, rule \textit{pre-commit} must hold for it. This rule makes sure that the estimate \( p_i \) adopts is equal to \( \text{maxEST}_i \). Therefore, \( p_i \) will adopt \( \text{absMaxEST}(k) \). If \( p_i \) prepares, it will execute line 29 of the pseudo-code, adopting \( \text{absMaxEST}(k) \) as well. Therefore all the processes in \( A \) (at
Lemma 23. In every run \( r \), if no correct process decides by the end of round \( GSR(r) + 1 \), and no process commits in round \( GSR(r) \), all processes will have the same estimate by the end of round \( GSR(r) + 1 \).

Proof. By Lemma 22, at the end of round \( GSR(r) \), at least \( m + 1 \) of processes adopt the estimate \( absMaxEST(GSR(r)) \) with a timestamp equal to \( absMaxTS(GSR(r)) \). Notice that an estimate \( est' \neq absMaxEST(GSR(r)) \) can become \( absMaxEST(GSR(r) + 1) \) only by adopting a new timestamp (that was not sent in round \( GSR(r) \)). This can be done only if a process commits with \( est' \) in round \( GSR(r) \), and this is not possible by the assumptions of our lemma. We conclude that \( absMaxEST(GSR(r) + 1) = absMaxEST(GSR(r)) \) (\( \neq est' \)).

No process decides in round \( GSR(r) + 1 \), and each \( p_i \) process receives a round \( GSR(r) + 1 \) message from \( n - m \) processes, including one message of the form \( \langle *, absMaxEST(GSR(r) + 1), ts, *, * \rangle \) and \( ts \) is either equal to \( absMaxTS(GSR(r)) \) (\( ts \neq GSR(r) \) since we assume in this lemma that no process commits in round \( GSR(r) \)). Whether \( p_i \) commits, pre-commits or prepares, because of rule \( pre-commit \) and line 29, the estimate \( p_i \) adopts is equal to \( maxEST \). Therefore, \( p_i \) will adopt \( absMaxEST(GSR(r) + 1) \), and we get that all processes adopt the same estimate by the end of round \( GSR(r) + 1 \).

Lemma 24. In every run \( r \), if no correct process decides by the end of round \( GSR(r) + 2 \), and some process commits in round \( GSR(r) \), all processes will have the same estimate by the end of round \( GSR(r) + 2 \).

Proof. By Lemma 22, at the end of round \( GSR(r) + 1 \), at least \( m + 1 \) of processes adopt the estimate \( absMaxEST(GSR(r) + 1) \) with a timestamp equal to \( GSR(r) \). Notice that an estimate \( est' \neq absMaxEST(GSR(r) + 1) \) can become \( absMaxEST(GSR(r) + 2) \) only by adopting a new timestamp (that was not sent in round \( GSR(r) + 1 \)). This can be done only if a process commits with \( est' \) in round \( GSR(r) + 1 \), and this is not possible because some process will receive the COMMIT message sent in this round, and Lemma 19. We conclude that \( absMaxEST(GSR(r) + 2) = absMaxEST(GSR(r) + 1) \) (\( \neq est' \)).

No process decides in round \( GSR(r) + 2 \), and each \( p_i \) process receives a round \( GSR(r) + 2 \) message from \( n - m \) processes, including one message of the form \( \langle *, absMaxEST(GSR(r) + 2), ts, *, * \rangle \) and \( ts \) is either equal to \( GSR(r) \) or to \( GSR(r) + 1 \). Whether \( p_i \) commits, pre-commits or prepares, because of rule \( pre-commit \) and line 29, the estimate \( p_i \) adopts is equal to \( maxEST \). Therefore, \( p_i \) will adopt \( absMaxEST(GSR(r) + 1) \), and we get that all processes adopt the same estimate by the end of round \( GSR(r) + 2 \).

Lemma 25. If in a round \( k \geq GSR(r) \) all estimates being sent are the same, all correct processes decide by round \( k + 2 \).

Proof. Observe that in our model every correct process executes an infinite number of rounds, and in particular, executes round \( k + 2 \). Also, it is obvious that all estimates being sent remain the same in all rounds starting at \( k \). We prove the lemma by contradiction. Assume that some correct process \( p_j \) does not decide by round \( k + 2 \) in some run \( r \). Therefore, \( p_j \) couldn’t have received a COMMIT message from a majority of processes in round \( k + 2 \). Since, in our model, from \( GSR(r) \) onward, every correct process receives messages from a majority of correct processes (including itself), it must have received a round \( k + 2 \) message \( m \) s.t. \( m.msgType = t \) from some process \( p_i \) with type \( t \neq COMMIT \). \( t \neq \)DECIDE, since \( p_j \) didn’t decide in round \( k + 2 \). Therefore, \( t \) must be PREPARE or PRE-COMMIT. If \( t = PREPARE \), this can happen only if in round \( k + 1 \), process \( p_i \) received messages with different estimates, since otherwise (if all estimates it receives are the same), even if there were no proper conditions (according to the algorithm)
for $p_i$ to DECIDE or COMMIT, its PRE-COMMIT rule would definitely evaluate to true and its round $k + 2$ message would be (at least) PRE-COMMIT. Therefore, $t = \text{PREPARE}$ is a contradiction to our assumption that in round $k + 1$ all estimates being sent are the same. If $t = \text{PRE-COMMIT}$, this means that $p_i$ didn’t receive any DECIDE message in round $k + 1$, and that rule pre-commit evaluated to true for $p_i$, but rule commit did not. Therefore, $p_i$ received $k + 1$ round messages from a majority of processes with some estimate $\text{maxEST}_i$, but didn’t receive any of them with the type COMMIT or PRE-COMMIT. This means that at the end of round $k$, there were processes that didn’t PRE-COMMIT. Let’s observe one such process $p_c$ (who’s round $k + 1$ message reached $p_i$) at the end of round $k$. It couldn’t have decided since $p_i$ didn’t receive any DECIDE messages in round $k + 1$. Since all estimates sent are the same in round $k$, its rule pre-commit must evaluate to true at the end of round $k$, and it sends either a COMMIT or a PRE-COMMIT message in round $k + 1$, a contradiction to the fact that $p_i$ received no such messages (since starting with round $k$, all estimates are the same, $\text{est'}$ must be the estimate sent by $p_c$).

Lemma 26. If $n = 2m + 1$ then in every run $r$ all correct processes decide by round $\text{GSR}(r) + 4$.

Proof. If some process correct process decides by the end of round $\text{GSR}(r) + 1$, lets denote by $k$ the round in which this happens, or $\text{GSR}(r) - 1$ (the later round between the two). Since $k \geq \text{GSR}(r) - 1$, in round $k + 1$, it is assured that the decision message will reach $m + 1$ processes, and in round $k + 2$, it will reach all the process since each one receives a message from $n - m$ processes. Therefore, every process will decide by round $k + 2$. If $k = \text{GSR}(r) + 1$, $k + 2 = \text{GSR}(r) + 3$, and the lemma holds.

Suppose that no correct process decides by the end of round $\text{GSR}(r) + 1$. If some process commits in round $\text{GSR}(r)$, all processes will decide by round $\text{GSR}(r) + 3$, by Lemma 21. If no process commits in $\text{GSR}(r)$, by Lemma 23, all processes will adopt the same estimate by the end of round $\text{GSR}(r) + 1$, and send it in round $\text{GSR}(r) + 2$. By Lemma 25, all processes will decide by the end of round $\text{GSR}(r) + 4$. □

Lemma 27. In every run $r$ all correct processes decide by round $\text{GSR}(r) + 5$.

Proof. If some process correct process decides by the end of round $\text{GSR}(r) + 2$, lets denote by $k$ the round in which this happens, or $\text{GSR}(r) - 1$ (the later round between the two). Since $k \geq \text{GSR}(r) - 1$, in round $k + 1$, it is assured that the decision message will reach $m + 1$ processes, and in round $k + 2$, it will reach all the process since each one receives a message from $n - m$ processes. Therefore, every process will decide by round $k + 2$. If $k = \text{GSR}(r) + 2$, $k + 2 = \text{GSR}(r) + 4$, and the lemma holds.

Suppose that no correct process decides by the end of round $\text{GSR}(r) + 2$. If some process commits in round $\text{GSR}(r)$, by Lemma 24, all processes adopt the same estimate by the end of round $\text{GSR}(r) + 2$, and send it in round $\text{GSR}(r) + 3$. By Lemma 25, all processes will decide by the end of round $\text{GSR}(r) + 5$. If no process commits in $\text{GSR}(r)$, by Lemma 23, all processes will adopt the same estimate by the end of round $\text{GSR}(r) + 1$, and send it in round $\text{GSR}(r) + 2$. By Lemma 25, all processes will decide by the end of round $\text{GSR}(r) + 4$. □

Theorem 28. The algorithm solves consensus in our model with global decision by round $\text{GSR}(r) + 5$ (or $\text{GSR}(r) + 4$ in case $n = 2m + 1$).

Proof. From Lemma 26, every correct process decides by round $\text{GSR}(r) + 4$, if $n = 2m + 1$. From Lemma 27, every process decides by round $\text{GSR}(r) + 5$. Validity holds, since the decision can only be one of the initial estimates of the processes. Uniform agreement was proved in Lemma 20. □