Timeliness, Failure-Detectors, and Consensus Performance

Idit Keidar
Technion
idish@ee.technion.ac.il

Alexander Shraer
Technion
shralex@tx.technion.ac.il

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.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Performance, Reliability, Theory

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 [7, 5]. Therefore, a system design that does not rely on strict synchrony is often advocated [27, 18, 9]; algorithms that tolerate arbitrary periods of asynchrony are called *indulgent* [21].

As it is well-known that consensus is not solvable in asynchronous systems [19], 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)* [18], following which certain properties hold. These properties can be expressed in terms of eventual timeliness of communication links [18, 11][1], or using the abstraction of *oracle failure detectors* [9]. 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 [23, 1, 2, 3, 29], 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.

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) [18]. 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 GI-

[1]A timely link delivers messages with a bounded latency; the bound is either known or unknown a priori.

PODC'06, July 22-26, 2006, Denver, Colorado, USA.

RAF, 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) [20], 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 [20], and our rounds do not have to be communication-closed like Gafni's[2]. 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*, Ω, which eventually outputs the same correct *leader* process at all processes; many consensus algorithms were designed for this model, e.g., [27, 14, 22]. 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 [20]. In contrast, we show that GIRAF is *general* enough to faithfully capture *any* oracle-based asynchronous algorithm in the model of [8] (see [26]). 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)* [13]. 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 ϕ 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 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 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 [9, 8], without taking complexity into account. In Section 3, we define the notion of α-*reducibility*, where round $GSR + l$ of the emulated model (for any l) occurs at most in round $GSR + \alpha(l)$ in the original model, in the same run. This notion captures reductions that incur a function $\alpha()$ penalty in running time. We define a special case of this notion, namely k-*round reducibility*, which is simply α-reducibility with $\alpha(l) = l + k$, and captures reductions that incur a k-round penalty in running time. Gafni [20] 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 α-reducibility (and k-round reducibility) provides a convenient instrument to describe such relations.

[2]In communication-closed rounds, each message arrives in the round in which it is sent.

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. [13] have shown that in the *Eventual Synchrony (ES)* [18] 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*, $\diamond$LM (see Table 1, row 2). In this model, processes are equipped with a *leader oracle*, Ω [8]. 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$_v$, where the subscript v denotes that the incoming links of a process can change in each round. $\diamond LM$ 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 4, we present a leader-based consensus algorithm for $\diamond$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; Ω and timely communication with a majority suffice. Interestingly, we show in Section 5, that if we replace Ω with an equivalent failure detector (in the "classical" sense), namely, $\diamond S$ [9], then for $f < \frac{n}{2} - 1$ this entails a linear lower bound on running time from GSR (see Table 1, row 3).

We next consider whether timely communication with a majority can be used in lieu of an oracle. 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 (possibly different) 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 6, 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. We show in [25], that the $\diamond AFM$ model is extremely scalable, i.e., implementing it in a system with many processes is much easier than implementing many other indulgent models.

Finally, we examine whether one can weaken the model even further. Can we relax the assumption that *all* correct processes

[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.

Model	Model Properties	Upper Bound	Lower Bound
ES	all links $\Diamond$timely	GSR+2 [13]	GSR+2 [13]
$\Diamond$LM	Ω, the leader is $\Diamond n$-source every correct process $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$	GSR+2 **Algorithm 2**	GSR+2 [13]
$\Diamond$SR	$\Diamond S$, the unsuspected process is $\Diamond n$-source every process $(n-f-1)$-destination$_v$ $f < n/2 - 1$ and reliable links	GSR+$2n+2$ [31]	GSR+$n-1$ **Lemma 3**
$\Diamond$AFM	$\exists m \in N, f \leq m < n/2$ s.t. every correct process $\Diamond(m+1)$-source$_v$ and $\Diamond(n-m)$-destination$_v$	if $n = 2m+1$ and $GSR > 0$: GSR+4 Otherwise: GSR+5 **Algorithm 3**	GSR+2 [13]
$\Diamond$MFM(m) $m \in N^+$ $f \leq m < n/2$	every correct process $\Diamond(n-m)$-source m correct processes $\Diamond n$-source $(n-m)$ correct processes $\Diamond(n-m)$-accessible every correct process $\Diamond m$-accessible reliable links	Unbounded [2, 3, 29]	Unbounded **Lemma 4**

Table 1: Upper and lower bounds on consensus global decision times in various models ($t < n/2$).

have incoming timely links from a majority, and allow a minority of the processes to each have one fewer timely link? (In case $n = 3$, only one timely link is removed). In Section 7, 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, 29], 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, 29], which were used for solving consensus. We note, though, that in [29], 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 8 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, 14, 22, 6, 17, 15, 30], 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 [13, 16]. As noted above, [13] shows that $GSR+2$ is a tight bound for global decision in ES. It uses Gafni's RRFD [20] framework. In [16], Dutta et al. focus on actual time rather than rounds, again in ES; they present an algorithm that decides by $GST + 17\delta$, where δ 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 our 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 [27], may require a linear number of rounds after GSR [12]. Most other Ω-based protocols, e.g., [14, 22], 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. [12] 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.

Models that allow moving send and receive omission failures, e.g., [34, 33], resemble our definitions of variable j-sources and j-destinations. For example, Schmid and Weiss [34] show that consensus is solvable in a model allowing each process to experience up to f^r receive omission failures and f^s send omission failures if and only if $f^r + f^s < n$. This precisely corresponds to our AFM model. Nevertheless, these papers only deal with perpetual timely links, and not with eventual models like $\Diamond$AFM.

2. MODEL AND PROBLEM DEFINITION

2.1 Distributed computation model

We consider an asynchronous distributed system consisting of a set Π 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 [28]. An automaton's transitions are triggered by *actions*, which are classified as *input*, *output*, and *internal*. Action π of automaton A is *enabled* in state s if A has a transition of the form (s, π, 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}, π_i, s_i) is a transition of A. We only consider *fair* runs, where no action is enabled without occurring in an infinite suffix.

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

States:

$k_i \in N$, initially 0 */*round number*/*

$sent_i[\Pi] \in$ *Boolean array*, initially $\forall p_j \in \Pi : sent_i[j] =$ *true*

$FD_i \in$ *OracleRange*, initially arbitrary

$M_i[N][\Pi] \in$*Messages*$\cup\{\perp\}$, initially $\forall k \in N \forall p_j \in \Pi : M_i[k][j] = \perp$

Actions and Transitions:

input *receive*$(\langle m, k\rangle)_{i,j}$, $k \in N$
  Effect: $M_i[k][j] \leftarrow m$

input *end-of-round*$_i$
  Effect: $FD_i \leftarrow$ *oracle*$_i$ (k_i)
    if $(k_i = 0)$ **then** $M_i[1][i] \leftarrow$ *initialize* (FD_i)
    else $M_i[k_i + 1][i] \leftarrow$ *compute* (k_i, M_i, FD_i)
    $k_i \leftarrow k_i + 1$
    $\forall p_j \in \Pi : sent_i[j] \leftarrow$ *false*

output *send* $(\langle M_i[k_i][i], k_i\rangle)_{i,j}$
  Precondition: $sent_i[j] =$ *false*
  Effect: $sent_i[j] \leftarrow$ *true*

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 $t < n/2$ of the processes may fail by crashing. The failure of process p_i is modeled using the action *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 [8], and is queried using the *oracle*$_i$ function.

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: *initialize()*, and *compute()*. Both are passed the oracle output, and *compute()* also takes as parameters the set of messages received so far and the round number. These functions are non-blocking, i.e. they are not allowed to wait for any other event.

Each process's computation proceeds in *rounds*. The advancement of rounds is controlled by the environment via the *end-of-round* input action. The *end-of-round*$_i$ actions occur separately in each process p_i, and there are no restrictions on the relative rate at which they occur at different processes, i.e., rounds are not necessarily synchronized among processes. The *end-of-round* action first occurs in round 0, whereupon it queries the oracle and calls *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 *end-of-round* action occurs, at which point the oracle is queried and *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 *end-of-round*$_i$ (i.e., *end-of-round* invocations at process p_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 *end-of-round*$_i$ in round k without a *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 [8] (see [26]). 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 *end-of-round* actions; (ii) oracle outputs; and (iii) *send* and *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), satisfying the following property:

Definition (GSR(r)): GSR(r) is the first round k, s.t. in every round $k' \geq k$ no process fails, and all eventual properties hold in k'.

As was defined earlier, the first communication round is round one. By slight abuse of terminology, we say that GSR$(r) = 0$ in a run r if (i) there are no failures in r; (ii) the oracle properties (if defined) hold from round zero in r; and (iii) the communication properties hold from round one in r. (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. [18, 35], or using GPS clocks).

2.3 Environment properties

We define several environment properties in GIRAF, mostly in perpetual form. Prefixing a property with $\diamond$ means that it holds from GSR onward. 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: $\forall k \in N^+$ 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 in round k**:** 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, possibly different in every round. (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.)

leader: $\exists$ correct p_i s.t. for every round $k \in N$ and every $p_j \in \Pi$, *oracle*$_j(k) = i$. The range of the *oracle()* function is Π.

Ω **failure detector:** $\diamond$ *leader*.

Note that the reliable and timely link properties imply 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.

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* $\cup\{\perp\}$. In every run r, $prop_i$ is initialized to some value $v \in$ *Values*, and dec_i is initialized to $\perp$. 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. REDUCIBILITY

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 [9, 8] 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 $T_{M_1 \to M_2}$ 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 $T_{M_1 \to M_2}$ 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 α-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 (α**-reducibility).** *For* α : N $\to$ N*, we say that model* M_2 *is* α*-reducible to model* M_1*, denoted* $M_1 \geq_\alpha M_2$*, if there exists a reduction algorithm* $T_{M_1 \to M_2}$ *s.t. for every run r and every* $l \in$ N*, round* $GSR_{M_2}(r) + l$ *in model* M_2 *occurs at most in round* $GSR_{M_1}(r) + \alpha(l)$ *in model* M_1*.*

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.* $\alpha(l) = l + k$*.*

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 5 we use the notion of k-round reducibility to prove that $\diamond S$ is 0-round reducible to $\diamond n$-source.

4. OPTIMAL LEADER-BASED ALGORITHM IN $\diamond$LM

The $\diamond$LM model requires that each process have a majority of incoming timely links (from GSR onward), which can vary in each round, and an Ω oracle that selects a correct $\diamond n$-source as leader. Formally:

$\diamond LM$ *(Leader-Majority)*: $t < n/2$, Ω failure detector, the leader is a $\diamond n$-source, and every correct process is a $\diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$.

The $\diamond$LM model is strictly weaker than ES: it is easy to show that $ES \geq_0 \diamond LM$, i.e. to simulate $\diamond LM$ in ES with no penalty, since $\diamond LM$ requires less $\diamond$timely links than ES, and the Ω failure detector output in any given round k can be (for example) the lowest-id of a process whose round k message was received in round k. Starting from GSR_{ES} this process is assured to be correct and since in ES all correct processes receive the same set of messages in each round, all simulated oracles will believe in the same correct process starting at round GSR_{ES}, meaning that $GSR_{\diamond LM} = GSR_{ES}$.

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 [10, 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 [13].

Algorithm 2 works in GIRAF, and therefore implements only the *initialize()* and *compute()* functions. These function are passed *leader*$_i$, 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 [27] initiates a new "ballot", that

Algorithm 2 Optimal leader–based algorithm for $\Diamond LM$, code for process p_i.

1: **Additional state**
2: $est_i \in Values$, initially $prop_i$
3: ts_i, $maxTS_i$, $lastApproval_i \in N$, initially 0
4: $prevLD_i$, $newLD_i \in \Pi$
5: $msgType_i \in \{$PREPARE, COMMIT, DECIDE$\}$, initially PREPARE
6: **Message format**
7: $\langle msgType \in \{$PREPARE, COMMIT, DECIDE$\}$, $est \in Values$, $ts \in N$, $leader \in \Pi$, $lastApproval_i \in N \rangle$
8: **procedure** initialize($leader_i$)
9: $prevLD_i \leftarrow newLD_i \leftarrow leader_i$
10: return message $\langle msgType_i, est_i, ts_i, newLD_i, lastApproval_i \rangle$ */*round 1 message*/*
11: **procedure** compute(k_i, M[*][*], $leader_i$)
12: **if** $dec_i = \perp$ **then**
13: */*Update variables*/*
14: $prevLD_i \leftarrow newLD_i$; $newLD_i \leftarrow leader_i$
15: $maxTS_i \leftarrow \max\{ m.ts \mid m \in M[k_i][*] \}$
16: **if** $|\{ j \mid M[k_i][j] \neq \perp \}| > \lfloor n/2 \rfloor$ **then**
17: $lastApproval_i \leftarrow k_i$
18: */*Round Actions*/*
19: **if** $\exists m \in M[k_i][*]$ s.t. $m.msgType =$ DECIDE **then** */*decide-1*/*
20: $dec_i \leftarrow est_i \leftarrow m.est$; $msgType_i \leftarrow$ DECIDE
21: **else if** $(|\{ j \mid M[k_i][j].msgType =$ COMMIT $\}| > \lfloor n/2 \rfloor)$
and $(M[k_i][prevLD_i].msgType = M[k_i][i].msgType =$ COMMIT$)$ **then** */*decide-2*/*
22: $dec_i \leftarrow est_i$; $msgType_i \leftarrow$ DECIDE
23: **else if** $(|\{ j \mid M[k_i][j].leader = prevLD_i \}| > \lfloor n/2 \rfloor)$ */*commit-1*/*
and $(M[k_i][prevLD_i].lastApproval = k_i - 1 \wedge M[k_i][prevLD_i].leader = prevLD_i)$ */*commit-2*/*
and $(newLD_i = prevLD_i)$ **then** */*commit-3*/*
24: $est_i \leftarrow M[k_i][prevLD_i].est$; $ts_i \leftarrow k_i$; $msgType_i \leftarrow$ COMMIT;
25: **else**
26: $est_i \leftarrow$ any $est' \in \{ M[k_i][j].est \mid M[k_i][j].ts = maxTS_i \}$
27: $ts_i \leftarrow maxTS_i$; $msgType_i \leftarrow$ PREPARE
28: return message $\langle msgType_i, est_i, ts_i, newLD_i, lastApproval_i \rangle$ */*round $k_i + 1$ message*/*

is, aborts any pending attempts to decide on some value, whenever a higher timestamp is observed, potentially leading to linear running time after GSR [12]. 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.

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 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 the full version [26]. Our main lemma 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_i 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_i is the first process to decide). Therefore, in round k, p_i hears COMMIT from majority M, including itself and its round k *prevLD*, p_l, 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_l. 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_l. Therefore, any commitment in round k is made with the estimate of p_l, 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 $k'-1 \geq k$, 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 $ts \geq k-1$ 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 $\geq k-1$. 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 the full version [26]). 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 Ω 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.

5. 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)*: reliable links, $\diamond S$ failure detector, the unsuspected process is $\diamond n$-source and all correct processes are $(n-f-1)$-destinations$_v$, where $f < \frac{n}{2} - 1$.

In the full version [26], we give a formal proof of our results. Below we state the roadmap of the proof.

LEMMA 1. *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., $M_{\diamond n} \geq_0 M_{\diamond S}$.*

From Lemma 1, 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 $\diamond SR \backslash \diamond S$.

We prove the lower bound using the impossibility of consensus in the *mobile failure* model [32], 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 2. *For any $k \in N$, let r be a run in the mobile failure model. There exists a run r' in $\diamond SR \backslash \diamond S$ with GSR(r) $= k$ and $f = 0$ such that $r'(k+n-2) = r(k+n-2)$.*

We strengthen the lower bound by proving that it is impossible to reach global decision in less than n rounds from GSR in the $\diamond SR \backslash \diamond S$ model, even with an algorithm especially tailored for some specific GSR.

LEMMA 3. *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 $\diamond SR \backslash \diamond S$ model.*

Note that our proof (combined with Algorithm 2, which achieves global decision by GSR+2 in $\diamond LM$) immediately implies that $\diamond SR \not\geq_k \diamond LM$ for any $k < n-3$.

6. 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)*: $\exists m \in N$, $f \leq m < n/2$ such that every correct process is a $\diamond(n-m)$-destination$_v$ and a $\diamond(m+1)$-source$_v$. 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. In runs with GSR $= 0$, this means that consensus is achieved in 5 rounds. In runs with GSR > 0, global decision is reached in 6 rounds. 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 the full version [26] we prove that the optimized algorithm reaches global decision by round GSR+4 for $n = 2m+1$ (when GSR > 0), and by round GSR+5 for other values of m ($f \leq m < n/2$). 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

Algorithm 3 Majority–based algorithm for $\diamond AFM$ model. Code for process p_i. Optimization for $n = 2m + 1$ is marked in gray.

1: **Additional state**
2: est_i, $maxEST_i \in$ *Values*, initially $prop_i$
3: ts_i, $maxTS_i \in \mathbf{N}$, initially 0
4: $IgotCommit_i \in$ *Boolean*, initially *false*
5: $gotCommit_i \in 2^{\Pi}$, initially $\emptyset$
6: $msgType_i \in \{$PREPARE, PRE-COMMIT, COMMIT, DECIDE$\}$, initially PREPARE
7: **procedure** initialize()
8: return message $\langle msgType_i, est_i, ts_i, IgotCommit_i, gotCommit_i \rangle$ */*round 1 message*/*
9: **procedure** compute(k_i, M[*][*])
10: **if** $dec_i = \perp$ **then**
11: */*Update variables*/*
12: $maxTS_i \leftarrow$ **max**$\{$ *m.ts* $\mid m \in M[k_i][*] \}$
13: $maxEST_i \leftarrow$ **max**$\{$ *m.est* $\mid m \in M[k_i][*] \wedge m.ts = maxTS_i \}$
14: $IgotCommit_i \leftarrow \exists m \in M[k_i][*]$ s.t. $m.msgType =$ COMMIT
15: $gotCommit_i \leftarrow \{$ j $\mid M[k_i][j].IgotCommit \}$
16: */*Round Actions*/*
17: **if** $\exists m \in M[k_i][*]$ s.t. $m.msgType =$ DECIDE **then** */*decide-1*/*
18: $dec_i \leftarrow est_i \leftarrow m.est$; $msgType_i \leftarrow$ DECIDE
19: **else if** $|\{ j \mid M[k_i][j].msgType =$ COMMIT $\}| > \lfloor n/2 \rfloor \ \wedge\ M[k_i][i].msgType =$ COMMIT **then** */*decide-2*/*
20: $dec_i \leftarrow est_i$; $msgType_i \leftarrow$ DECIDE
21: **else if** $|\bigcup_{j \in \Pi} M[k_i][j].gotCommit| > \lfloor n/2 \rfloor$ **then** */*decide-3*/*
22: $dec_i \leftarrow est_i \leftarrow maxEST_i$; $msgType_i \leftarrow$ DECIDE
23: **else if** $|\{ j \mid M[k_i][j].est = maxEST_i \}| > \lfloor n/2 \rfloor$ **then** */*pre-commit*/*
24: **if** $\exists j$ s.t. $M[k_i][j].est = maxEST_i \wedge M[k_i][j].msgType =$ COMMIT or PRE-COMMIT **then** */*commit*/*
25: $est_i \leftarrow maxEST_i$; $ts_i \leftarrow k_i$; $msgType_i \leftarrow$ COMMIT;
26: **else**
27: $est_i \leftarrow maxEST_i$; $ts_i \leftarrow maxTS_i$; $msgType_i \leftarrow$ PRE-COMMIT;
28: **else**
29: $ts_i \leftarrow maxTS_i$; $est_i \leftarrow maxEST_i$; $msgType_i \leftarrow$ PREPARE
30: return message $\langle msgType_i, est_i, ts_i, IgotCommit_i, gotCommit_i \rangle$ */*round $k_i + 1$ message*/*

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 message type to PREPARE) and adopts the estimate $maxEST_i$ and timestamp $maxTS_i$ (line 29).

Correctness. A process may commit with different estimates in different rounds. However, we show (in the full version [26]) 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 $maxEST_i$, which has the maximal timestamp, p_i commits with a timestamp $\geq k$. Using an inductive argument, we get that $maxEST_i = x$. 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 the full version [26]). 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 =GSR+1, at which point all processes have the same $maxEST$. 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 =GSR+2. In both cases, every process has the same $maxEST$ 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 the full version [26], 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 the full version [26], 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 $maxEST$ (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 the full version [26].

7. IMPOSSIBILITY OF BOUNDED TIME GLOBAL DECISION IN ◇MFM

We define the ◇MFM family of models, for $m \in N^+$, $f \leq m < n/2$, as follows:

$\diamond MFM(m)$ *(Majority-From-Majority)*: 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 ◇AFM, where we have shown that constant-time decision is attainable. In the full version [26], we show that the time for global decision after GSR in all of these models is unbounded. Specifically, we prove the following lemma:

LEMMA 4. *For any $m \in N^+$ s.t. $f \leq m < n/2$, there exists no consensus algorithm that reaches global decision in bounded time from GSR in ◇MFM(m).*

Note that this implies that there is no k-round reduction from $\diamond LM$ or $\diamond AFM$ to $\diamond MFM(m)$ for any k.

Our proof builds three indistinguishable runs, using a partition argument, to derive a contradiction. Note that our notion of timely links is more abstract than the real-time-based definition used in [2, 3, 29], 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.

8. 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, 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 haver 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.
- 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 $\Diamond$AFM by making fewer processes $\Diamond(m+1)$-sources, and still achieve constant or bounded time consensus? and what would be the effect of weakening the assumption that the leader is a $\Diamond$n-source in $\Diamond$LM, on consensus performance?
- In this paper, we have focused on global decision. It can be interesting to investigate local consensus decision [17], 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, Neeraj Suri for many helpful discussions. We also thank the anonymous reviewers whose remarks helped us to greatly improve the paper.

9. REFERENCES

[1] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Stable leader election. In *DISC*, pages 108–122, 2001.

[2] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. On implementing omega with weak reliability and synchrony assumptions. In *PODC*, pages 306–314, 2003.

[3] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *PODC*, pages 328–337, 2004.

[4] O. Bakr. Performance evaluation of distributed algorithms over the Internet. Master's thesis, MIT, Feb. 03.

[5] O. Bakr and I. Keidar. Evaluating the running time of a communication round over the Internet. In *PODC*, pages 243–252, 2002.

[6] F. Brasileiro, F. Greve, A. Mostefaoui, and M. Raynal. Consensus in one communication step. In *6th Intl. Conference on Parallel Computing Technology*, pages 42–50, Sept. 2001.

[7] N. Cardwell, S. Savage, and T. Anderson. Modeling the performance of short tcp connections, 1998.

[8] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[9] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[10] B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. *J. Algorithms*, 51(1):15–37, 2004.

[11] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. In *IEEE TPDS*, pages 642–657, June 1999.

[12] P. Dutha, R. Guerraoui, and I. Keidar. The overhead of consensus failure recovery. Technical Report 200456, École Polytechnique Fédérale de Lausanne, 2004.

[13] P. Dutha, R. Guerraoui, and I. Keidar. The overhead of consensus failure recovery. Submitted for publication, 2005.

[14] P. Dutta and R. Guerraoui. Fast indulgent consensus with zero degradation. In *EDCC*, Oct. 2002.

[15] P. Dutta and R. Guerraoui. The inherent price of indulgence. In *PODC*, July 2002.

[16] P. Dutta, R. Guerraoui, and L. Lamport. How fast can eventual synchrony lead to consensus?. In *DSN*, pages 22–27, 2005.

[17] P. Dutta, R. Guerraoui, and B. Pochon. Tight lower bounds on early local decisions in uniform consensus. In *DISC*, pages 264–278, Oct 2003.

[18] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[19] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[20] E. Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *PODC*, pages 143–152, 1998.

[21] R. Guerraoui. Indulgent algorithms. In *19th ACM Symp. on Principles of Distributed Computing (PODC-19)*, pages 289–298, July 2000.

[22] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, 2004.

[23] R. Guerraoui and A. Schiper. "Γ-accurate" failure detectors. In *WDAG*, pages 269–286, 1996.

[24] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults - a tutorial. Technical Report MIT-LCS-TR-821, MIT, May 2001.

[25] I. Keidar and A. Shraer. How to choose a timing model? Technical Report CCIT 586, Department of Electrical Engineering, Technion, May 2006.

[26] I. Keidar and A. Shraer. Timeliness, failure-detectors, and consensus performance. Technical Report CCIT 576, Department of Electrical Engineering, Technion, Feb. 2006.

[27] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[28] N. Lynch and M. Tuttle. An introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989.

[29] D. Malkhi, F. Oprea, and L. Zhou. Omega meets paxos: Leader election and stability without eventual timely links. *DISC*, pages 199–213, sep 2005.

[30] J.-P. Martin and L. Alvisi. Fast byzantine consensus. In *DSN*, pages 402–411, 2005.

[31] A. Mostefaoui and M. Raynal. Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In *13th Intl. Symp. on Distributed Computing*, pages 49–63, Sept. 1999.

[32] N. Santoro and P. Widmayer. Time is not a healer. *6th Annual Symp. Theor. Aspects of Computer Science*, volume 349 of LNCS:304–313, feb 1989.

[33] U. Schmid and C. Fetzer. Randomized asynchronous consensus with imperfect communications. *SRDS*, 00:361, 2003.

[34] U. Schmid and B. Weiss. Impossibility results and lower bounds for consensus under link failures. Technical Report 183/1-129, Technische Universita"t Wien, Dept. of Automation, Apr. 2002.

[35] J. L. Welch and H. Attiya. *Distributed computing: fundamentals, simulations and advanced topics*. McGraw-Hill, Inc., Hightstown, NJ, USA, 1998.