How to Choose a Timing Model?

Idit Keidar and Alexander Shraer

Abstract

When employing a consensus algorithm for state machine replication, should one optimize for the case that all communication links are usually timely, or for fewer timely links? Does optimizing a protocol for better message complexity hamper the time complexity? In this paper, we investigate these types of questions using mathematical analysis as well as experiments over PlanetLab (WAN) and a LAN. We present a new and efficient leader-based consensus protocol that has $O(n)$ stable-state message complexity (in a system with n processes) and requires only $O(n)$ links to be timely at stable times. We compare this protocol with several previously suggested protocols. Our results show that a protocol that requires fewer timely links can achieve better performance, even if it sends fewer messages.

***Index Terms*— synchrony assumptions, eventual synchrony, failure detectors, consensus algorithms, FT Middleware.**

I. Introduction

Consensus is an important building block for achieving fault-tolerance using the state-machine paradigm [20]. It is therefore not surprising that the literature is abundant with fault-tolerant protocols for solving this problem. But how does a system designer choose, among the multitude of available protocols, the right one for her system? This decision depends on a number of factors, e.g., time and message complexity, resilience to failures (process crashes, message loss, etc.), and robustness to unpredictable timing delays.

In this paper we focus on the latter, namely the assumptions the protocol makes about timeliness. These are captured in a *timing model*. We study the impact of the choice of a timing model on the performance in terms of time and message complexity. It is important to note that although the physical system is often given, the system designer has freedom in choosing the timing model representing this system. For example, one seldom comes across a system where the network latency can exceed an hour. This suggests that in principle, even the most unpredictable systems can be modeled as synchronous, with an upper bound of an hour on message latency. Although a round-based synchronous protocol works correctly in this system, it can take an hour to execute a single communication round, and hence may not be the optimal choice. Indeed, measurements show that timely delivery of 100% of the messages is feasible neither in WANs nor under high load in LANs[8], [5], [3]. Instead, systems choose timeouts by which messages *usually* arrive (e.g., 90% or 99% of the time); note that by knowing the typical latency distribution in the system, a designer can fine-tune the timeout to achieve a desired percentage of timely arrivals. One can then employ protocols that ensure safety even when messages arrive late [8], [21], [14]. Such protocols are called indulgent [17].

While indulgent protocols ensure safety regardless of timeliness, they do make some timeliness assumptions in order to ensure progress. Periods during which these assumptions hold are called *stable*. For example, it is possible to require *Eventual Synchrony (ES)* [14], [8], where messages among all pairs of processes are timely in stable periods. Alternatively, one can use weaker majority-based or leader-based models, where only part of the links are required to be timely in stable periods. This defines a tradeoff: whereas weaker models may require more communication rounds for decision, they may also have better *coverage*, that is, their timeliness requirements will be satisfied more often. A second consideration is message complexity: protocols that send more messages per round may require fewer rounds. Thus, there may also be a tradeoff between the time and message complexities.

In order to provide insights into such tradeoffs, this paper (1) defines a new timing model, (2) introduces a novel time and message efficient algorithm, and (3) presents an evaluation of different consensus algorithms using probabilistic analysis, as well as concrete measurements in a LAN and in WAN over PlanetLab [4]. We next elaborate on each one of these contributions.

We define a new model (Section III), *eventually weak leader-majority* $\Diamond WLM$. It includes a leader oracle, and only requires that in stable periods, there be timely links from a designated leader process to other processes and from a majority of processes to the leader. Nothing is required before

Department of Electrical Engineering, Technion, Haifa, Israel. {idish@ee, shralex@tx}.technion.ac.il

A preliminary version of this paper appears in the 37th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)

stabilization. The leader oracle can be implemented with linear (in n, the number of processes) per-round stable state message complexity [22], [24].

We then present a new efficient algorithm for $\Diamond WLM$ (Section IV), which has linear stable state message complexity, and decides within 5 rounds from stabilization. If the leader stabilizes earlier than the communication, our algorithm decides in 4 rounds. Although $\Diamond WLM$ was not previously defined, its conditions allow some existing algorithms [21], [9] to make progress. However, these algorithms may take $O(n)$ rounds after stabilization [11] when run in $\Diamond WLM$, in runs where the leader is not initially known.

Section V performs probabilistic analysis of the behavior of consensus in different indulgent models, comparing our new algorithm with three previously known algorithms. All the compared algorithms always take a constant number of rounds from stabilization, however unlike our new algorithm, which has linear stable-state message complexity, the other three algorithms have quadratic message complexity. Our analysis studies the coverage of different models as well as the number of rounds needed to reach stabilization and then decision in each model. Although it makes simplifying assumptions, this analysis gives a good starting point to understand such behaviors in real systems. Note that we study the performance of consensus without taking into account the cost of leader election. This is justified since election protocols often ensure leader stability [24], [1], [15], i.e., the leader is seldom re-elected. Thus, the same leader may persist for numerous instances of consensus (possibly thousands). For simplicity, our analysis looks at runs where the processes' rounds are synchronized.

In a real implementation, a round synchronization mechanism must be employed. One way to do so is using synchronized clocks (e.g., GPS clocks) when present. Section VI discusses possible ways to achieve round synchronization, including the solution we implemented and deployed in PlanetLab.

We then compare the performance of our new algorithm in LAN and WAN with that of three previously known algorithms (Section VII). We compare our measurements with the probabilistic analysis and explain discrepancies that arise. We give insights to the effect of good leader election on leader-based consensus protocols. We show that our message efficient protocol, although requiring more stable communication rounds than several previously known protocols, incurs practically no cost in terms of actual running time, due to its easier to satisfy weak timeliness requirements: it achieves comparable (and sometimes superior) performance to that of the best $O(n^2)$ (message complexity) protocol, provided that adequate timeouts are set. Section VIII concludes the paper.

II. Related work

Model and Algorithm. In an earlier paper [19], we introduced a round-based framework, GIRAF, for describing timing models and indulgent protocols that exploit them. We have studied the number of rounds required for consensus in stable periods in several timing models. Nevertheless, [19] studies neither how long it takes to reach stability in practical network settings, nor the round durations in these models. The current paper provides analysis and measurements of the actual time it takes to reach consensus while assuming the different models in a LAN and a WAN (PlanetLab). Moreover, [19] focuses on time complexity, and ignores message complexity, which is no less important. Our new protocol has $O(n)$ stable state message complexity, unlike the algorithms in [19], which all have quadratic message complexity.

The $\Diamond WLM$ model satisfies the progress requirements of the well-known Paxos protocol [21], and recent improvements, such as [9]. But as noted in [11], although these algorithms ensure constant time decision in Eventual Synchrony (ES), they may take a linear number of communication rounds after stabilization to decide in weaker models like $\Diamond WLM$. Most other previously suggested leader-based protocols, e.g., [10], [18], require the leader to receive timely messages from a majority in each round, including during unstable periods, and hence do not work in $\Diamond WLM$.

Malkhi et al. [24] have presented a somewhat weaker timing model intended for use with Paxos, where, as in $\Diamond WLM$, some process has bidirectional timely links with a majority, but unlike $\Diamond WLM$, this process does not have outgoing timely links to the rest of the processes. Although their model allows Paxos to make progress so that some of the processes decide, it does not allow *all* the processes to reach consensus decision in a timely manner [19]. Here, we measure time until *global decision*, i.e., until all processes decide, and therefore strengthen the model accordingly. Probabilistic analysis of the coverage of other models was performed in a different context by Schmid [25].

Evaluation. The time to reach consensus *after* stabilization in ES has been studied in [13]; here, we also measure the time it takes to reach stabilization, and consider additional models. Other papers evaluated related algorithms in practical settings. Cristian and Fetzer [8] studied stable periods, but only for a model similar to ES, over a LAN. The insight that a leader-based algorithm can work better than ES appears in previous measurements on WANs [3], [2] and simulations [26]. However these studies treated different questions than we do, e.g., did not measure the time required to get a sufficiently long stable period that allows for consensus decision. Unlike most of the previous evaluations, our evaluation includes mathematical analysis as well as measurements in both LAN and WAN, thus identifying general trends that do not depend on a specific setting.

III. Model and Problem Definitions

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 state-machines, called I/O automata [23].

Algorithm 1 Generic algorithm for process p_i.

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 $\forall k \in N \forall p_j \in \Pi : M_i[k][j] = \perp$
$D_i \in 2^{\Pi}$, initially $\emptyset$
Actions and Transitions:
input *receive*$(\langle m, k\rangle)_{i,j}$, $k \in N$
Effect: $M_i[k][j] \leftarrow m$
output *send* $(\langle M_i[k_i][i], k_i\rangle)_{i,j}$
Precondition: $j \in D_i \wedge sent_i[j] = false$
Effect: $sent_i[j] \leftarrow true$
input *end-of-round*$_i$
Effect: $FD_i \leftarrow oracle_i\,(k_i)$
if $(k_i = 0)$ **then** $\langle M_i[1][i], D_i\rangle \leftarrow initialize\,(FD_i)$
else $\langle M_i[k_i+1][i], D_i\rangle \leftarrow compute\,(k_i,\ M_i,\ FD_i)$
$k_i \leftarrow k_i + 1$
$\forall p_j \in \Pi : sent_i[j] \leftarrow false$

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. Communication links do not create, duplicate, or alter messages. Messages may be lost by links or take unbounded latency. Timing models defined below restrict such losses and late arrivals. Less than $n/2$ processes may fail by crashing. A process that does not fail is *correct*.

Algorithms and models are defined using the GIRAF framework [19], which we extend here to allow for arbitrary communication patterns. For space limitations, we only overview GIRAF; for formal treatment see [19]. In GIRAF, all algorithms are instantiations of Algorithm 1, a generic round-based algorithm. Process p_i is equipped with a *failure detector oracle*, which can have an arbitrary output range [6], and is queried using the *oracle*$_i$ function. To implement a specific algorithm, one implements 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.

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. However, specific environment properties defined below do require some synchronization between processes, 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., see Section VI).

When the *end-of-round* action first occurs, it queries the oracle and calls *initialize()*, which returns the message for sending in round 1 and a set, D_i, of the destinations of this message. Subsequently, in each round, a process sends a message to processes in D_i (although allowed, self messages are not necessary since a message is always stored in the incoming buffer of the sender) 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, and a new set D_i of target processes. Note that although in Algorithm 1 all received messages are stored, in a real implementation many of these messages can be garbage-collected, depending on the needs of the specific application.

Environments are specified using *round-based properties*. We consider only *eventual* properties. Namely, the system may be asynchronous for an arbitrary period of time, but eventually there is a round GSR (*Global Stabilization Round*), starting from which no process fails and all properties hold in each round. GSR is the *first* round that satisfies this requirement.

We now define some round-based properties. The link from p_s to p_d is *timely in round* k, if the following holds: if (i) *end-of-round*$_s$ occurs in round k, (ii) $d \in D_s$ in round k, and (iii) p_d is correct, then p_d receives the round k message of p_s in round k. A process p is a $\Diamond j$*-source*$_v$ if in every round $k \geq$ GSR, there are j processes to which it has timely outgoing links. Correctness is not required from the recipients, and p's link with itself counts towards the count of j. The subscript "v" indicates that the set of j timely links is allowed to change in each round (i.e., the failures are mobile). Similarly, a correct process p is a $\Diamond j$*-destination*$_v$ if in every round $k \geq$ GSR, it has j timely incoming links from correct processes. An Ω failure detector outputs a process so that there is some correct p_i s.t. for every round $k \geq$ GSR and every correct p_j, $oracle_j(k) = i$.

We study the following four timing models:

- ES *(Eventual Synchrony)*[14] in every round $k \geq$ GSR, all links between correct processes are timely.
- $\Diamond LM$ *(Leader-Majority)*[19]: Ω failure detector, the leader is a $\Diamond n$-source, and every correct process is a $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$.
- (New) $\Diamond WLM$ *(Weak-Leader-Majority)*: Ω failure detector, the leader is a $\Diamond n$-source and a $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$.
- $\Diamond AFM$ *(All-From-Majority)*[19] (simplified): every correct process is a $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$, and a $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-source$_v$.

Consensus. A consensus problem is defined for a given value domain, *Values*. We assume that *Values* is a totally ordered set (our algorithm makes use of this order). Every process p_i has a read-only variable $prop_i \in$ *Values*, initialized to some value $v \in$ *Values*, and a write-once variable $dec_i \in$ *Values*$\cup\{\perp\}$ initialized to $\perp$. We say that p_i *decides* $d \in$*Values* in round k if p_i writes d to dec_i when $k_i = k$.

A consensus algorithm must ensure: (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 algorithm A achieves *global decision* at round k if every process that decides decides by round k and at least one process decides at round k. It was shown in [16] that any indulgent algorithm that solves consensus also solves uniform consensus, a variant of consensus in which no two processes (whether correct or faulty) are allowed to decide differently. Therefore, for the rest of this paper, we implicitly refer to uniform consensus whenever consensus is mentioned.

IV. Time and Message Efficient Algorithm

Algorithm 2 is a consensus algorithm for ◊WLM, which has a linear stable state message complexity and reaches global decision within 5 rounds of GSR.

As in many indulgent algorithms, including Paxos, processes commit with increasing timestamps (called "ballots" in [21]), and decide on a value committed by majority. In Paxos, the leader always attempts to discover the highest timestamp in the system before committing on a new one. Although this occurs promptly in ES, in $\Diamond WLM$, even after stabilization, the leader can continue to hear increasing timestamps for $O(n)$ rounds. Each time it receives a timestamps higher than the one it has, the decision attempt is aborted, leading to a linear worst case decision time after GSR [11]. Our algorithm avoids such scenarios. Nevertheless, we still need the leader to start a new decision attempt with a fresh timestamp higher than those previously possessed by processes. But unlike Paxos, our algorithm does not assume that the leader knows all the timestamps of correct processes. Instead, the new timestamp is chosen to be the round number, which is monotonically increasing. This must be done with care, so as to ensure that the leader does not miss timestamps of real decisions.

Key idea to preserving consistency is to trust the leader, even if it competes against a higher timestamp, provided that it indicates that at least a majority believes it to be the leader. The latter is conveyed using the *majApproved* message field, which attests to the fact that the leader's timestamps reflect "fresh" information from a majority, and therefore any timestamp it does not know of could not have led to decision.

A second challenge our algorithm addresses is avoiding "wasted" rounds when the system stabilizes in the middle of a decision attempt. This poses a problem, as we strive to reduce the number of rounds needed for reaching a consensus, so that the system is not required to have long periods of stability. The solution we employ is to pipeline proposals. Namely, the leader tries in each round to make progress towards a decision, based on its current state and the messages it gets in the current round, regardless of the unknown status of previous attempts to make progress.

We now describe the algorithm in detail. Algorithm 2 works in the framework of Algorithm 1 described in Section III, and therefore implements the *initialize()* and *compute()* functions. These function are passed $leader_i$, the leader trusted by p_i's Ω oracle in the current round. Process p_i maintains the following local variables: an estimate of the decision value, est_i; the timestamp of the estimated value, ts_i; the maximal timestamp received in the current round, $maxTS_i$; the maximal estimate received with timestamp $maxTS_i$ in the current round, $maxEST_i$ (recall that *Values* is a totally ordered set); the leader provided by the oracle at the end of the previous round, *prevLD*$_i$, and in the current round, *newLD*$_i$; a *Boolean* flag, *majApproved*$_i$, which is used to indicate whether p_i received a message in the current round from a majority of processes that indicated p_i as their leader; and the message type, *msgType*$_i$, which is used as follows: If p_i sees a possibility of decision in the next few rounds, 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 (lines 15-18), and 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 (rule *decide-1*, line 20), and sets its message type (for the round $k_i + 1$ message) to DECIDE.
- If p_i receives a COMMIT message from a majority, including itself (rule *decide-2*), and receives a message from itself with the *majApproved* = *true* (rule *decide-3*), it decides on its own estimate and sets its message type to DECIDE (line 23). Rule *decide-3* ensures that no other process commits or decides in the same round with a different value, since the *commit* rule checks $majApproved$ of the leader, and two processes cannot claim to be $majApproved$ in the same round, since it is not possible that different processes were trusted to be leaders by a majority in the same round (round $k_i - 1$). Rule *decide-2* ensures that a majority of processes have the latest information about the decided value. Since commits in further rounds require the leader to hear from a majority (the $majApproved$ indicator required by rule *commit*), the leader must hear from at least one process that has this information, and this will ensure that it does not promote a value that contradicts agreement.
- Let *prevLD*$_i$ be the leader indicated in p_i's round k_i message. If p_i receives a round k_i message from *prevLD*$_i$ with the *majApproved* indicator as $true$, 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 25). We say that p_i *commits in round* k_i with estimate est'. The $majApproved$ indicator ensures that commits of the same round are on the same value, since any such commit is on an estimate received from a leader that was trusted by a majority in the previous round (k_i-1), and majorities intersect.

Algorithm 2 leader–based algorithm, code for process p_i.

1: **Additional state**
2: $est_i \in Values$, initially $prop_i$; ts_i, $maxTS_i \in N$, initially 0; $majApproved_i \in Boolean$, initially *false*
3: $prevLD_i$, $newLD_i \in \Pi$; $msgType_i \in \{$PREPARE, COMMIT, DECIDE$\}$, initially PREPARE
4: **Message format**
5: $\langle msgType \in \{$PREPARE, COMMIT, DECIDE$\}$, $est \in Values$, $ts \in N$, $leader \in \Pi$, $majApproved_i \in Boolean\rangle$
6: **procedure** Destinations($leader_i$)
7: **if** ($leader_i = p_i$) **then** return Π.
8: **else** return $\{leader_i\}$
9: **procedure** initialize($leader_i$)
10: $prevLD_i \leftarrow newLD_i \leftarrow leader_i$
11: return $\langle\langle msgType_i, est_i, ts_i, newLD_i, majApproved_i\rangle$, Destinations($leader_i$)$\rangle$
12: **procedure** compute(k_i, M[*][*], $leader_i$)
13: **if** $dec_i = \perp$ **then**
14: */*Update variables*/*
15: $prevLD_i \leftarrow newLD_i$; $newLD_i \leftarrow leader_i$
16: $maxTS_i \leftarrow \max\{\ m.ts \mid m \in M[k_i][*]\ \}$
17: $maxEST_i \leftarrow \max\{\ m.est \mid m \in M[k_i][*] \wedge m.ts = maxTS_i\ \}$
18: $majApproved_i \leftarrow (|\{\ j \mid M[k_i][j].leader = p_i\ \}| \ > \lfloor n/2 \rfloor)$
19: */*Round Actions*/*
20: **if** $\exists m \in M[k_i][*]$ s.t. $m.msgType =$ DECIDE **then** */*decide-1*/*
21: $dec_i \leftarrow est_i \leftarrow m.est$; $msgType_i \leftarrow$ DECIDE
22: **else if** $((|\{\ j \mid M[k_i][j].msgType =$ COMMIT $\}| \ > \lfloor n/2 \rfloor) \wedge M[k_i][i].msgType =$ COMMIT$)$ */*decide-2*/* **and** $(M[k_i][i].majApproved)$ **then** */*decide-3*/*
23: $dec_i \leftarrow est_i$; $msgType_i \leftarrow$ DECIDE;
24: **else if** $(M[k_i][prevLD_i].majApproved)$ **then** */*commit*/*
25: $est_i \leftarrow M[k_i][prevLD_i].est$; $ts_i \leftarrow k_i$; $msgType_i \leftarrow$ COMMIT;
26: **else** $ts_i \leftarrow maxTS_i$; $est_i \leftarrow maxEST_i$; $msgType_i \leftarrow$ PREPARE
27: return $\langle\langle msgType_i, est_i, ts_i, newLD_i, majApproved_i\rangle$, Destinations($leader_i$)$\rangle$

- Otherwise, p_i prepares (sets his message type to PREPARE) and adopts the estimate $maxEST_i$ and timestamp $maxTS_i$ (line 26).

Finally, p_i returns the message for the next round and a subset of processes to which this message is intended. This group is calculated using procedure *Destinations()* as follows: if p_i believes that it is the leader of the current round, then *Destinations()* returns the set of all processes, and otherwise, the procedure returns the trusted leader. Thus, starting from the first round in which all processes indicate the same leader in their messages (at most one round after GSR), every process sends a message to this leader, and the leader sends a message to every other process. The stable state message complexity is therefore linear in n.

We prove the correctness of Algorithm 2 in Appendix I, and show that it reaches global decision by round GSR+4, i.e., in 5 rounds starting at GSR. If the eventual requirements of the Ω leader are satisfied starting from round GSR-1 (instead of starting from round GSR as the model requires), then all correct processes decide by round GSR+3, i.e., in 4 rounds (if GSR$=1$ this means that querying the oracle before the first communication round returns the correct Ω leader at all processes). We make this distinction in order to analyze the performance of the algorithm in the common case, when leader re-election is rare.

V. Probabilistic Comparison

We study four models and the fastest known algorithm in each model – 3 rounds for ES ([12]), 3 for $\diamond LM$ ([19]), 4 with stable leader for $\diamond WLM$ (Section IV), and 5 for $\diamond AFM$ ([19]).

In this section we model link failure probabilities as Independent and Identically Distributed (IID) Bernoulli random variables. By "link failure" we mean that the link fails to deliver a message in the same round in which it is sent. We assume that processes proceed in synchronized rounds, although this is not required for correctness, and focus on runs with no process failures, which are common in practice. Additionally, we do not take the cost of leader election into account, since we assume a stable leader, i.e., a leader that is seldom re-elected (e.g., [24], [1]). Such a leader can persist throughout numerous instances of consensus.

We denote the probability that a message arrives on time by p. For simplicity, we do not treat a process' link with itself differently than other links. Our metric in this section is number of rounds until global decision. The length of each round is the time needed to satisfy p, and it is the same for all algorithms we deal with, while the number of rounds depends on the algorithm. In Section VII-B we investigate the effect of changing the explicit time length of each round on the overall decision time in each model.

A. Mathematical Analysis

All communication in some single round k can be represented as an n by n matrix A, where the rows are the destination process indices, the columns are the source process indices, and each entry $A_{i,j}$ is 0 if a message sent by p_j to p_i does not arrive in round k, and 1 if it does reach p_i in round

k. p is the probability of any entry $A_{i,j}$ to be 1. Note that our protocol for $\Diamond WLM$ may not send messages on some links. If a message is not sent, we denote the corresponding entry in A by $\perp$. We define random variables for decision time in different models subscripted by the model name, e.g., D_{ES} is the total number of rounds until decision (including the time until stabilization) in ES. We denote by P_M (e.g., P_{AFM}) the coverage of model M, i.e., the probability that a communication round satisfies the requirements of M. Below we assume a fixed n, and show the asymptotic analysis in Appendix III.

Analysis of ES. Recall that ES requires all entries in the matrix A to be 1. The probability for this is:

$$P_{ES} = p^{n^2} \tag{1}$$

An optimal ES consensus algorithm reaches a global decision in 3 rounds from stabilization, thus we need the assumptions of ES to be satisfied for 3 consecutive rounds starting at some round $k \geq 1$. The probability of this to happen at any given round k is $(P_{ES})^3$. Thus:

$$E(D_{ES}) = \frac{1}{(P_{ES})^3} + 2 \tag{2}$$

Analysis of $\Diamond$LM. Let p_k be the stable leader. For $\Diamond LM$, it is required that A has a majority of ones in every row. Additionally, $\Diamond LM$ requires that $\forall 1 \leq j \leq n\ A_{j,k} = 1$. Denote the event that there is a majority of ones in row A_j by M and the event that $A_{j,k} = 1$ by L. We have n independent rows, and thus:

$$P_{\Diamond LM} = (Pr(L \cap M))^n = (Pr(L) \cdot Pr(M|L))^n \tag{3}$$

Note that $Pr(L) = p$. Given that $A_{j,k} = 1$, the probability that more than $\frac{n}{2} - 1$ of the remaining $n-1$ entries of row j are 1 is:

$$Pr(M|L) = \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-1-i} \tag{4}$$

Global decision is achieved in 3 rounds from stabilization in $\Diamond LM$, meaning that this condition on A has to be satisfied for 3 rounds, and thus:

$$E(D_{\Diamond LM}) = \frac{1}{(P_{\Diamond LM})^3} + 2 \tag{5}$$

Analysis of $\Diamond$WLM. Let p_k be the stable leader. $\Diamond WLM$ requires that A has a majority of ones in row A_k. We denote this event by M. Additionally, it requires that $\forall 1 \leq j \leq n\ A_{j,k} = 1$. We denote this event by L'.

$$P_{\Diamond WLM} = Pr(L' \cap M) = Pr(L') \cdot Pr(M|L') \tag{6}$$

Note that $Pr(L') = p^n$, and $Pr(M|L') = Pr(M|L)$ (defined in Equation 4) since row A_k is independent of other rows. These conditions only examine the row and column corresponding to the leader, p_k. Since p_k is stable, all processes agree on its identity, and thus, the leader sends messages to all other processes, while every other process sends a message to the leader. Hence, the entries of A are not $\perp$.

We first analyze the algorithm of Section IV, which takes 4 rounds starting from GSR, under the stable leader assumption. We get:

$$E(D_{\Diamond WLM}) = \frac{1}{(P_{\Diamond WLM})^4} + 3 \tag{7}$$

For comparison, we also examine an alternative solution: running the optimal algorithm for $\Diamond LM$ over a simulation of $\Diamond LM$ in $\Diamond WLM$ (shown in Appendix II). We show that this simulation reaches global decision in 7 rounds. Therefore:

$$E(D_{Simulated\ \Diamond WLM}) = \frac{1}{(P_{\Diamond WLM})^7} + 6 \tag{8}$$

Analysis of $\Diamond$AFM. This model requires A to have a majority of ones in each row and column. Consider a given row k of A. We first analyze the probability that the row includes a majority of ones. To this end, let X_j be the random variable representing the cell $A_{k,j}$. According to our assumption, $X_1, X_2, ..., X_n$ are independent and identically distributed Bernoulli random variables with probability of success p. Let $X = \sum_{i=1}^{n} X_i$. The probability that any given row in A has a majority of 1's is:

$$Pr(X > \frac{n}{2}) = \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

For n (independent) rows we need to raise this expression to the power of n. Now assume that every row has a majority of 1 entries. The probability of an entry to be 1 is still at least p. We therefore can make an identical calculation for the columns, raising the expression again to the power of 2.

$$P_{\Diamond AFM} \geq (Pr(X > \frac{n}{2}))^{2n} \tag{9}$$

Since the algorithm for $\Diamond AFM$ achieves global decision in 5 rounds from GSR, this needs to hold for 5 consecutive rounds, and therefore we additionally raise the expression to the power of 5. We get:

$$E(D_{\Diamond AFM}) = \frac{1}{(P_{\Diamond AFM})^5} + 4 \tag{10}$$

B. Numerical results

We plot the upper bounds on expected decision times given in Equations 2, 5, 7, 8 and 10 for specific values of p. We focus on the case that $n = 8$, similarly to the group sizes used in other performance studies of consensus-based systems [8], [2], [9], which used 4-9 nodes.

In Figure 1(a) we see that even with a very high probability of timely message delivery, performance in ES deteriorates drastically as p decreases, while $\Diamond AFM$, $\Diamond LM$ and the direct algorithm for $\Diamond WLM$ maintain excellent performance. The direct algorithm for $\Diamond WLM$ does not incur practically any penalty for its improvement of message complexity from quadratic in n to linear. We can also see that $\Diamond LM$ and our algorithm for $\Diamond WLM$ outperform $\Diamond AFM$ in this high range of p. Finally, the simulated algorithm for $\Diamond WLM$ ($\Diamond LM$ algorithm running over the simulation from Appendix II) is

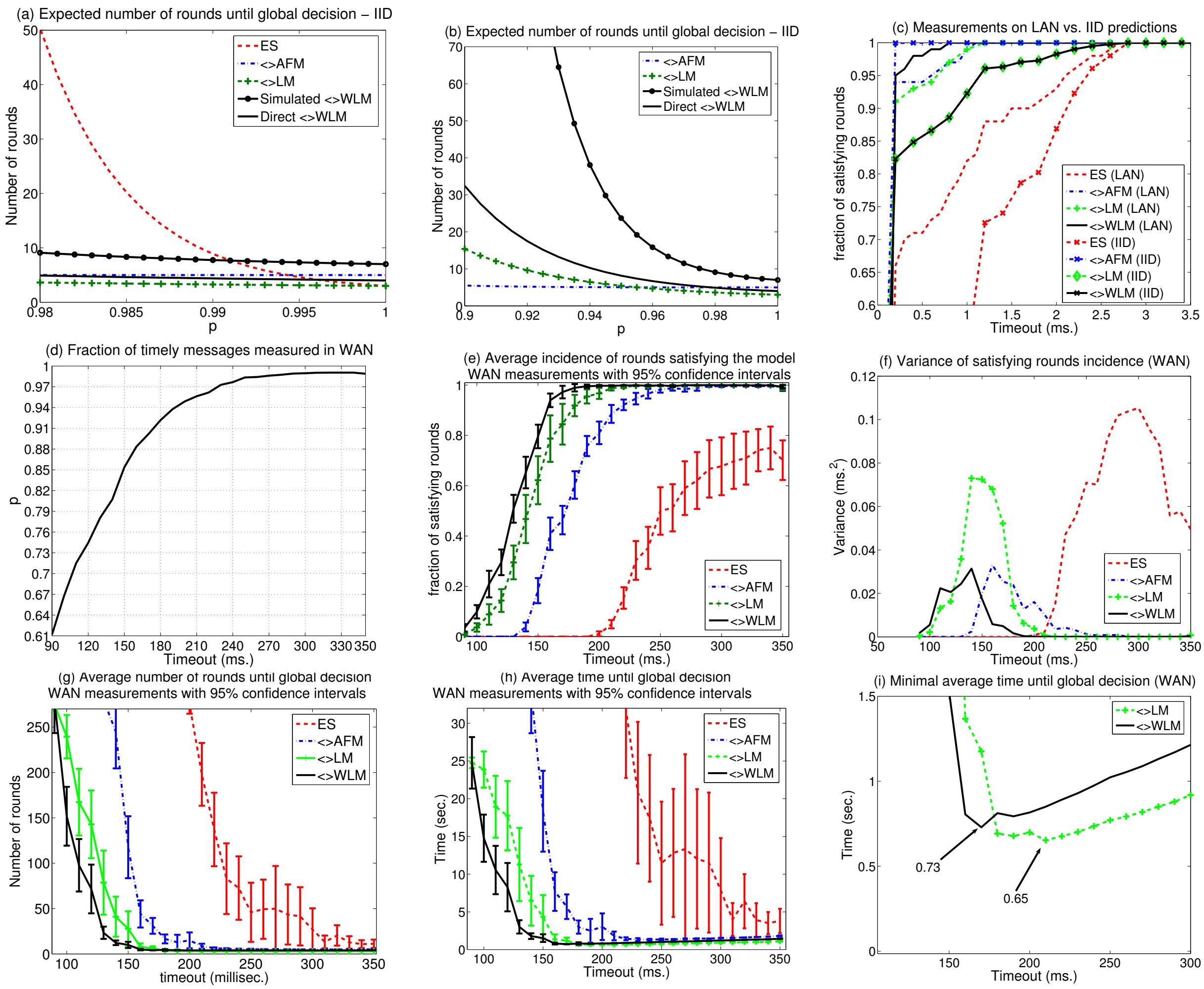

Fig. 1. Comparison between ES, $\Diamond AFM$, $\Diamond LM$ and $\Diamond WLM$.

worse than the direct one, as it is much harder to maintain the needed timeliness conditions for 7 rounds than for 4 rounds.

Figure 1(b) examines smaller success probabilities, starting from from 0.9. Here ES is not shown, since it steeply deteriorates as we decrease p (e.g., ES requires 349 rounds for $p = 0.97$). The intuition of why ES performs so poorly, is that it is practically impossible to get 3 matrices not containing a single zero entry, if the probability for a zero is non-negligible. Our direct algorithm for $\Diamond WLM$ greatly outperforms the simulated algorithm (e.g., for $p = 0.92$ our algorithm requires 18 rounds, while the simulation-based requires 114 rounds). $\Diamond AFM$ is better than $\Diamond LM$ and $\Diamond WLM$ when p is low, but from $p = 0.96$, $\Diamond LM$ becomes better, and starting from $p = 0.97$, the direct algorithm for $\Diamond WLM$ becomes better. Thus, $\Diamond AFM$ is better for lower p values, e.g., for $p = 0.85$ $\Diamond AFM$ is expected to take 10 rounds and $\Diamond LM$ - 69 rounds. Comparing the algorithms for $\Diamond LM$ and $\Diamond WLM$, we see that even though $\Diamond WLM$ requires fewer timely links, $\Diamond LM$ is slightly better, since the dominant factor in the performance of both is the requirement that the leader is a $\Diamond n$-source, and satisfying it for 4 rounds instead of 3 is harder.

VI. Implementing Round Synchronization

In GIRAF (Algorithm 1), there are no restrictions on the relative rate at which rounds occur at different processes, i.e., rounds are not necessarily synchronized among processes. However, certain model-specific environment properties do require some synchronization of rounds, e.g., in $\Diamond WLM$, starting from some round onward, the leader is required to receive a message from some majority of processes (that might be different in each round) during the same round in which they are sent. Therefore, an implementation of an environment that guarantees such properties needs to employ some sort of round or clock synchronization mechanism. One way to do so is using synchronized clocks (e.g., GPS clocks or a clock synchronization protocol like NTP) when present. This is often sufficient in a LAN, where machines usually have synchronized clocks. In WANs, it is preferable to implement solutions that do not require synchronized clocks.

One way to achieve this is to explicitly wait for sufficiently many messages in each round before moving to the next round. For example, the leader in an algorithm that assumes $\Diamond WLM$ might wait until it receives messages from a majority of processes in a round before proceeding to the next round. However, this approach has several drawbacks. First, the algorithm has no knowledge of when the stabilization occurs, thus the round-termination conditions (e.g., sufficiently many messages) must be applied from the outset, even before stabilization. A temporary partition can then cause a process to fall behind, and others to be an arbitrary number of rounds ahead of it. When stabilization eventually occurs, it may take this process unbounded time to catch up. This can be solved by making the late processes "skip" rounds, as done for example in [11]. Second, it is impossible to specify round-termination conditions corresponding to properties such as j-source, which require j processes to receive some outgoing message. Thus, such an implementation cannot support the full range of GIRAF predicates.

We next propose a timeout-based implementation, which does not suffer from the above drawbacks. Our solution is simple to implement, and, as we see in the next section, also achieves excellent synchronization. An important part of our implementation is a "network learning" process, which measures the average latency between every pair of nodes in the system using pings. At each node n_i, this process maintains an array L_i, such that $L_i[j]$ is the average latency between node n_i and node n_j as measured at n_i. This information is used for two purposes: to set the parameters and fine-tune the round synchronization mechanism, as we describe below, and to elect one well-connected process as the leader, as discussed in Section VII-A. The system can be made adaptive and self healing by periodically running this process, re-calculating the arrays L_i and re-electing the leader, however this was not required in our setting as our experiments were short and it sufficed to run the network-learning process only once.

The round synchronization process on a node n_i gets the *timeout* as a parameter and runs two threads. In each local round k_i, the task of the first thread is to receive and record messages, inserting them into a message buffer according to the round to which the message belongs (this information is included in the message). Upon receipt of a message belonging to a future round $k_j > k_i$ from a node n_j, this thread records the message and notifies the second thread.

The second thread starts each round k_i by sending messages to its peers, and then waits for the remainder of the round as specified by the *timeout* parameter. At the end of each round it calls *compute()*. In case of a notification from the first thread about a receipt of round-k_j message from node n_j, this thread stops waiting, i.e., the round is ended immediately, and *compute()* is called. It then starts round k_j, and the duration of this round is set to *timeout* $-L_i[j]$.

This algorithm allows a slow node to join its peers already in round k_j, utilizing round-k_j message it received, and takes into account the expected latency of this message to approximate the remaining time for round k_j in order to start round $k_j + 1$ together with the peers. We found that this algorithm achieves very fast synchronization, and whenever the synchronization is lost, it is immediately regained.

VII. Measurements

In this section we compare ES, $\Diamond AFM$, $\Diamond LM$ and $\Diamond WLM$ using experiments in two different practical settings - a LAN and a WAN (using PlanetLab). Additionally, we investigate whether the predictions made assuming the IID model in Section V were accurate. Like our analysis, the experiments involve 8 nodes.

A. LAN

Our experiment includes 8 nodes running simultaneously on a 100Mbit/sec LAN. Each process sent 100 UDP messages to all others. In a LAN, machines often have synchronized clocks, and there is no need for a synchronization algorithm. We therefore do not focus on round synchronization over LAN, and only measure message latencies and their impact on satisfying the conditions for consensus in different models.

The purpose of this experiment is to compare P_M, i.e., the coverage of model M (the probability of a communication round to satisfy model M) according to IID-based predictions to its coverage according to measurements in a LAN, for various timeouts. A message is considered to arrive in a communication round if its latency is less than the timeout. The IID-predicted values are calculated by taking the fraction of all messages that arrived in all communication rounds of the experiment as an estimate for p (the probability of a message to arrive on time in the IID analysis) and then using Equations 1, 3, 6 and 9 from Section V-A. We found that the measured p values were high already for very short timeouts. For example, whereas for a timeout of $0.1ms$ we measured $p = 0.7$, for a timeout of $0.2ms$ it was already $p = 0.976$.

Figure 1(c) shows measured and predicted P_{ES}, $P_{\Diamond AFM}$, $P_{\Diamond LM}$ and $P_{\Diamond WLM}$. We see that even in a LAN, the ES model has low coverage, which matches the IID-based predictions. Although still worse than the other models, ES is better in practice than what was predicted. The reason is that the messages that are late in a run tend to concentrate, rather than to spread among all rounds of the run uniformly as in IID. Thus, in practice, there are fewer rounds that suffer from message loss, and P_{ES} is higher.

On the other hand, $\Diamond AFM$ is worse in reality than was predicted, since it is sensitive to a poor performance of any single node. While in IID all nodes are the same, in our experiment, one node was occasionally slow. $\Diamond AFM$ requires this node, like any other, to receive a message from a majority of processes, and its message had to reach a majority of processes (these two requirements can be satisfied by the same set of links). Since this node is slow, there is a higher chance of messages to be late on its links than on other links (unlike in IID), making it harder to satisfy $\Diamond AFM$. As $\Diamond LM$ requires each process to receive a message from a majority, it suffers from the same problem as $\Diamond AFM$. $\Diamond LM$ additionally requires that the messages of the leader reach all processes, which explains why there are more rounds satisfying $\Diamond AFM$ than $\Diamond LM$.

According to IID-based prediction, at a high rate of message arrival (p values), $P_{\Diamond LM}$ and $P_{\Diamond WLM}$ are almost identical as can be seen from Figure 1(c), and both are worse than $\Diamond AFM$. In practice, for leader-based algorithms, choosing a good leader helps. As implementing a leader election algorithm is beyond the scope of this paper, we designated one process to act as a leader in all runs. We chose this process as follows: before running our experiments, we measured the round-trip times of all links using pings, and then chose a well-connected node to be the leader. Given this leader, both $\Diamond WLM$ and $\Diamond LM$ behaved much better than IID analysis predicted, and we see that $\Diamond WLM$ performs much better than all other models. When we run $\Diamond LM$ and $\Diamond WLM$ with a less optimal leader, whose links have average timeliness, we saw that much bigger timeouts are needed for reasonable performance, and in particular, bigger timeouts than for $\Diamond AFM$. For example, while $\Diamond AFM$ reaches $P_{\Diamond AFM} = 0.97$ at a timeout of $0.9ms$, with an average leader $\Diamond WLM$ and $\Diamond LM$ reach the same incidence only at a timeout of $1.6ms$. With a good leader $\Diamond WLM$ reaches this point at $0.35ms$ and $\Diamond LM$ at $0.8ms$.

Note that we did not experiment under high load conditions, as our focus is not on high throughput applications. Under a high load, different protocols may prove advantageous.

B. WAN

We implemented round synchronization (Section VI) and deployed it in PlanetLab, using 8 nodes located in Switzerland, Japan, California USA, Georgia USA, China, Poland, United Kingdom, and Sweden. The participating processes on these nodes are started up non-synchronously, and then synchronized and continue running for an overall of 300 communication rounds per experiment. We consider only rounds that occur after the system stabilizes for the first time (with respect to the model) to eliminate startup effects. The experiment was repeated with different timeouts, 33 times (runs) for each timeout. The PlanetLab node located in United Kingdom was chosen to serve as the leader for the leader-based protocols, since it was found to be well connected using the same method as was done for our LAN experiment (Section VII-A). We measure the time and number of rounds until the appropriate conditions for global decision are satisfied for each model, starting at 15 random points of each run, and the average of these represent the run. We also measure the fraction of rounds in each run that satisfy the timeliness requirements of the different models.

Figure 1(d) shows how timeouts translate to fraction of delivered messages (p in Section V) as measured in our experiment. We have chosen to work with timeouts which assure that up to 99% messages are delivered on time, since it is known that in WANs, the maximal latency can be orders of magnitude longer than the usual latency [5], [3], and thus assuring 100% is unrealistic.

Figure 1(e) shows the measured coverage P_{ES}, $P_{\Diamond AFM}$, $P_{\Diamond LM}$ and $P_{\Diamond WLM}$, averaged over the repetitions of the experiment for each timeout, as well as the 95% confidence interval for the average. Figure 1(f) shows the varience of the values used to calculate the average points in Figure 1(e). We see that $\Diamond WLM$ has much better coverage than the other models. This is because $\Diamond WLM$ only requires timeliness from the incoming and outgoing links of the leader. We also observe that $\Diamond LM$ and $\Diamond WLM$ have much better coverage than $\Diamond AFM$ and ES. For example, for a timeout of $160ms$ we get $P_{ES} = 0$, $P_{\Diamond AFM} = 0.4$ while $P_{\Diamond LM} = 0.79$ and $P_{\Diamond WLM} = 0.94$.

We see that ES rounds are really rare, especially with short timeouts (for example when the timeout is less than 200ms, $P_{ES} = 0$), which matches the IID-based prediction of Section V (on average, a timeout of $200ms$ corresponds to $p = 0.95$ used in IID analysis, i.e., 95% of messages arrive on time). We observe that while the confidence intervals of $P_{\Diamond AFM}$, $P_{\Diamond LM}$, and $P_{\Diamond WLM}$ are small and diminish as we increase the timeout, the confidence intervals for ES grow. Given a fixed number of measurements, the interval length follows from the variance. ES has high variance even for large timeouts, due to message loss. While in some runs, over 80% of rounds satisfy ES with a timeout of $350ms$, in others only 30% do. For short timeouts the variance of ES is low and its confidence intervals are short since the incidence of ES rounds is consistently low.

Figure 1(f) shows that for longer timeouts, the high incidence of $\Diamond AFM$, $\Diamond LM$ and $\Diamond WLM$ rounds varies only slightly (unlike ES). However, for short timeouts $\Diamond LM$ has high variance. This is caused by its sensitivity to bad performance by any single node, as was also observed in LAN. Specifically, for a timeout of $160ms$, while in some runs 95% of all rounds satisfy the conditions of $\Diamond LM$, in other runs little more than 15% do. This happened because in some runs with this timeout, PlanetLab node located in Poland was slow to receive messages, although most of the messages it sent arrived on time. While in IID all links are the same, we saw that in reality this is not true. This affects $\Diamond LM$ which requires every node to receive a message from a majority. On the other hand, $P_{\Diamond AFM}$ is consistently low (around 0.4, rarely above 0.5) for this timeout, hence the low variance. For larger timeouts, usually all nodes manage to receive a message from a majority, and we see that the incidence of $\Diamond AFM$ and $\Diamond LM$ is high, while the confidence intervals become shorter and the variance goes to 0.

Figure 1(g) and Figure 1(h) show the average (over all runs) number of rounds and time (resp.) that were needed to reach global decision in each model. We observe that for low timeouts the algorithm of Section IV achieves consensus much faster than the algorithms assuming any of the other models ([12], [19]). For timeouts starting with approximately 180ms and higher, its performance is comparable to $\Diamond LM$, whereas $\Diamond AFM$ takes more rounds and time than both for timeouts less than 230ms. As before, the choice of the leader gave $\Diamond LM$ and $\Diamond WLM$ an advantage over $\Diamond AFM$ and thus the difference from IID-based prediction in Figure 1(b) (according to Figure 1(d), a timeout of $160ms$ corresponds, on average, to $p = 0.88$). We note that the advantage of the leader-based protocols stems from our assumption of leader-stability. If the leader is highly unstable, the time it takes these protocols to decide can be expected to be much higher (depending on the specific leader election protocol, which is beyond the scope of this paper), and hence $\Diamond AFM$ would be preferred.

In general, we see that a longer timeout (a higher p in the IID analysis), reduces the number of rounds for decision. On the other hand, it is obvious that a higher p, or a longer timeout, make each individual round longer. We wish to explore this tradeoff and determine the optimal timeout. Of course, the specific optimum would be different for a different system setting, but the principle remains. Figure 1(i) zooms-in on the appropriate part of Figure 1(h), and demonstrates this tradeoff for $\Diamond LM$ and $\Diamond WLM$. For timeouts less than $170ms$ (on average, this corresponds to $p = 0.90$ for IID), while $\Diamond WLM$'s required number of rounds is increasing (as the timeout decreases), the length of each round is decreasing. For timeouts more than $170ms$ (as the timeout increases) the number of required rounds decreases, but the cost of each round increases. For example, if we set our timeout to $180ms$, although the number of rounds will be very small (4.5 rounds on average according to Figure 1(g)), the actual time until decision will be $800ms$, which is about the same as the average time we would get if we shorten the timeout to $160ms$ although the required number of rounds would be higher. This shows that setting conservative timeouts (improving p) will not necessarily improve performance. As we see from this graph, it might actually make it worse.

From Figure 1(i), we conclude that in our setting, choosing the timeout to be $170ms$ is optimal for the $\Diamond WLM$ algorithm and the timeout $210ms$ is optimal for $\Diamond LM$. These timeouts correspond to $p = 0.90$ and $p = 0.96$, e.g., setting the timeout to $170ms$ causes 90% of messages on average to arrive on time in our setting. Note that we present a methodology rather than a specific timeout: a system administrator can perform measurements and choose the timeout for a specific system, according to such criteria.

Finally, when compared with their optimal timeouts, we see that $\Diamond WLM$ is expected to take $730ms$, which is only $80ms$ more than what $\Diamond LM$ is expected to take at its best setting. We conclude that it is clearly well worth using $\Diamond WLM$, while gaining the reduction of stable state message complexity from quadratic to linear.

VIII. Conclusions

We presented a timing model that requires timeliness on $O(n)$ links in stable periods and allows unbounded periods of asynchrony. We introduced a consensus algorithm for this model, which has linear per-round stable state message complexity, and achieves global decision in a constant small number of rounds from stabilization. Since all previously known algorithms that can operate in this model require linear number of rounds, we compared our algorithm to algorithms that require stronger models, all of which also have quadratic message complexity.

Even though our algorithm might take more rounds to decide compared to the others, we have shown that its easier to satisfy weak stability requirements allow it to achieve comparable or even superior global consensus decision time (with very low variance), despite the fact that it sends much fewer messages in each round. Thus, optimizing for message complexity and requiring fewer timely links might actually improve decision time. Our analysis includes measurements in a LAN and a WAN, as well as mathematical analysis, and thus is valid in a broad variety of systems.

Appendix I
Correctness of Algorithm 2

Lemma 1. *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 28) 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 round k' message it received in round k' (line 29) and again, by induction hypothesis, the claim is true.

Lemma 2. *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 28) or the maximum timestamp (of a round k message) received in round k as its new timestamp (line 29). Since p_i receives its own message in round k, the latter is not lower than its current timestamp. In case it commits, since according to Lemma 1, its old timestamp cannot exceed $k - 1$, by adopting k it can only increase.

Lemma 3. *If in round k, a process p_i commits on estimate est_i, then no process commits in round k with a different* *estimate, or decides in round k with a different estimate using rules* decide-2,3.

Proof: Observe a process p_i that commits in round k. Then p_i evaluates rule *commit* to *true* and commits or decides on the estimate that it receives from its leader, $prevLD_i$ (line 28). By rule *commit*, $M[k][prevLD_i].majApproved = true$, meaning that there is a majority of processes that send a round $k-1$ message with $prevLD_i$ as their leader. Let us denote this majority by M_1.

Suppose that a process p_j commits in round k with estimate est_j. By the same reason as above, there is a majority of processes that send a round $k-1$ message with $prevLD_j$ as their leader. Let us denote this majority by M_2. Since M_1 and M_2 intersect, as two majorities, $prevLD_i = prevLD_j$. Since p_j commits on the estimate est_j sent by $prevLD_j$, we get that $est_j = est_i$.

If a process p_j decides using rules *decide-2,3* in round k with estimate est_j, then by rule *decide-3*, $M[k][j].majApproved = true$, meaning that p_j was believed to be the leader in the previous round $k-1$ by a majority of processes. Let us denote this majority by M_2. Since M_1 and M_2 intersect, as two majorities, $prevLD_i = j$. Since p_j decides on its own estimate est_j, we get that $est_j = est_i$.

Lemma 4. *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 4 does not hold for DECIDE messages, since a process decides adopting only the estimate and not the associated timestamp from another DECIDE message.

Lemma 5 (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 rules *decide-2,3*. Therefore, p_i receives a majority of COMMIT messages in round k, including from itself, and it decides on x - the estimate of one of the COMMIT messages (the one from itself). From Lemma 3, all COMMIT messages include the same estimate - x. 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 3, 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. 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 (rule *decide-1*) or p sends a COMMIT message with estimate y (rule *decide-2*). In both cases, by the induction hypothesis, $y = x$.

Suppose by contradiction that some process p_j commits in round $k'+1$ with estimate $z \neq x$. First, since p_i decides by rules *decide-2,3* in round k, by Lemma 3 we have that $k'+1 \neq k$. By induction hypothesis $k' \geq k-1$ and we now get that $k' > k-1$. Since $k' > k-1 \geq 1$ we also get that $k' > 1$. Since p_j commits, it hasn't received any DECIDE message in round $k'+1$. Since rule *commit* evaluated to true for p_j, a message $m = \langle type\ (\neq DECIDE),\ z,\ ts_z,\ *,\ true\rangle$ was received by p_j in round $k'+1$ from the leader ld. Notice that ts_z might be different than $maxTS_i$ of round $k'+1$.

Observe the $majApproved = true$ field of the message m. 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 2, since $k' > k-1$, p_a's timestamp is at least $k-1$.

If $type$ =COMMIT, this means that $ts_z = k'$ (line 28). As was explained, $k' > 1$, and by Lemma 4 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' (line 29). Because it received a message from p_a and because, according to Lemma 1, 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 4) 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.

Auxiliary Notation: we define $k_{leader} \geq GSR$ to be the first round starting from which all correct processes indicate in their messages the same correct process as their leader.

Lemma 6. *Starting from round k_{leader}, (a) the correct Ω leader receives a message from a majority of processes. (b) every correct process receives the message of the correct Ω leader.*

Proof: By the definition of $\Diamond WLM$, starting at round GSR the leader receives a message from a majority of processes, and every correct process receives a message from the leader. This is provided that these messages are actually sent by the processes (this follows from the definition of timely link). Since $k_{leader} \geq GSR$, it is left to prove that processes will send these messages.

In each round of Algorithm 2, every process sends a message to its leader, and the leader sends a message to all processes. It follows from the definition of k_{leader}, that in the computation of round $k_{leader}-1$, every correct process gets the identity of the same correct leader from its oracle. Therefore, every correct process sends a message to this unique leader at round k_{leader}. By the guarantees of $\Diamond WLM$, the leader receives a message from majority. This proves (a).

(b) is correct, since the leader also trust itself starting from the computation of round $k_{leader}-1$, and will therefore send a message to every process in round k_{leader}. By the the guarantees of $\Diamond WLM$ this message of the leader will be delivered to every correct process.

Lemma 7. *In every round $k \geq k_{leader}+1$, the Ω leader sends $majApproved = true$ in its round k message, and every correct process p that does not decide before round k, either commits or decides in round k.*

Proof: In our model, every correct process executes an infinite number of rounds, and in particular, executes round k. If p decides by rule *decide-1* or rules *decide-2,3* we are done. Otherwise, in order to prove the lemma, we need to show that rule *commit* is satisfied.

Starting from round k_{leader} all processes indicate the same correct process $leader$ in their messages. Since, by Lemma 6, $leader$ receives a message from a majority of processes in round k_{leader} onward, and these processes indicate it as leader in round $k-1 \geq k_{leader}$, it will send $majApproved = true$ in its round k message. Since, again by Lemma 6, starting from round k_{leader} every process receives a message from the correct Ω leader, p receives a message from $leader$ in round k (p has $prevLD = leader$ in round k), and evaluates rule *commit* to $true$.

Lemma 8. *All correct processes decide by round $k_{leader}+3$.*

Proof: Observe that in our model every correct process executes an infinite number of rounds, and in particular, executes round $k_{leader}+3$. We prove the lemma by contradiction. Assume that some correct process p_j does not decide by round $k_{leader}+3$. Then it did not receive any DECIDE messages in round $k_{leader}+3$, and in particular, since by Lemma 6 it receives a message from its leader, the leader did not decide in the previous round, namely round $k_{leader}+2$. This means that in round $k_{leader}+2$, the leader evaluated at least one of *decide-2* or *decide-3* to $false$. But according to Lemma 6 rule *decide-3* must evaluate to true for the leader. So the problem was with rule *decide-2*. Since by Lemma 6 the leader received a message from a majority of processes in round $k_{leader}+2$, one of the messages must have been with type $\neq$ COMMIT. According to Lemma 7, all non-commit messages must be DECIDE messages. But then the leader should decide in round $k_{leader}+2$ by rule *decide-1* - a contradiction.

Lemma 9. *(a) all correct processes decide by round $GSR+4$; and (b) if the eventual requirements of the Ω leader are satisfied from round $GSR-1$ (instead of from GSR), then all correct processes decide by round $GSR+3$.*

Proof: (a) According to the definition of $\Diamond WLM$, starting from round GSR all (correct) processes get the same leader indication from their Ω oracle (and this indication does not change in further rounds). Therefore, starting from round $GSR+1$ all processes indicate the same correct Ω leader in their messages, and we get that $k_{leader} = GSR+1$. From Lemma 8 every correct process decides by round $k_{leader} + 3 = GSR+4$.

(b) if the eventual requirements of the Ω leader are satisfied from round $GSR-1$ (instead of from GSR), then all correct processes indicate the same correct leader process in their messages starting from round GSR onward, we get that $k_{leader} = GSR$, from Lemma 8, all correct processes decide by round $k_{leader}+3 = GSR+3$.

Theorem 10. *(a) the algorithm solves consensus by round $GSR+4$; and (b) if the eventual requirements of the Ω leader are satisfied starting from round $GSR-1$ (instead of starting from GSR as required by the model), then all correct processes decide by round $GSR+3$.*

Proof: From Lemma 9, every correct process decides by round $GSR+4$, or $GSR+3$ if the condition of (b) is satisfied. Validity holds, since the decision can only be one of the initial estimates of the processes. Uniform agreement is proven in Lemma 5.

APPENDIX II
A SIMULATION OF $\Diamond LM$ IN $\Diamond WLM$

As was explained in [19], 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*$_T$*()*

Algorithm 3 simulation of $\Diamond LM$ in $\Diamond WLM$, at process p_i.

Additional state
  $M_i^{fixed}[N][\Pi] \in$ *Messages* $\cup \{\bot\}$,
    initially $\forall k \in N \forall p_j \in \Pi : M_i^{fixed}[k][j] = \bot$

procedure initialize$_{\Diamond WLM}$(*leader*$_i$)
  return $\langle$initialize$_{\Diamond LM}$(*leader*$_i$), $\Pi\rangle$

procedure compute$_{\Diamond WLM}$(k_i, M[*][*], *leader*$_i$)
  if (k_i is odd) **then**
    return $\langle\{$ $M[k_i][*]$, $\Pi\rangle$
      /*k_i *is even*/
  forall $j \in N$
    if ($\exists l \in N$, s.t. $M[k][l][j] \neq \bot$) **then**
      $M_i^{fixed}[k/2][j] = M[k][l][j]$
  return $\langle$compute$_{\Diamond LM}$($k_i/2$, M_i^{fixed}, *leader*$_i$), $\Pi\rangle$

and *compute*$_T$*()*, and invokes *initialize*$_A$*()* and *compute*$_A$*()* in model M_1 (while satisfying the properties of M_2).

Algorithm 3 presents a simulation, of the $\Diamond LM$ model introduced in [19], in the $\Diamond WLM$ model presented in this paper. Therefore, we show an implementation of *initialize*$_{\Diamond WLM}$*()* and *compute*$_{\Diamond WLM}$*()* functions that work in $\Diamond WLM$ model. We denote by *initialize*$_{\Diamond LM}$*()* and *compute*$_{\Diamond LM}$*()* the functions of an algorithm designed for $\Diamond LM$.

In odd rounds k_i, every process p_i just forwards the messages it collected in round k_i as an array. The j^{th} entry of the array is $\neq \bot$ only if p_i received a message from p_j in the current round. In even rounds k_i, each message $M_i[k_i][l]$ that p_i receives from p_l is in fact an array, as explained above. In order to find out what message p_j sent in the previous round, p_i looks for this message in one of the arrays it received. Thus, if there is a process p_l that sent p_j's message (has the j^{th} entry of the array it sent $\neq \bot$), p_i saves this message in a local message buffer, M_i^{fixed}, in the entry $M_i^{fixed}[k/2][j]$. It then calls $compute_{\Diamond LM}$ with this local message buffer, and local round number $k/2$. This function is called every other round, hence the $k/2$. Thus, we simulate one round of $\Diamond LM$ in every two rounds of $\Diamond WLM$.

Lemma 11. $GSR_{\Diamond LM} \leq GSR_{\Diamond WLM} + 2$

Proof: Recall that all eventual properties of $\Diamond WLM$ are satisfied starting from round $GSR_{\Diamond WLM}$, and that both $\Diamond WLM$ and $\Diamond LM$ do not have any perpetual properties.

By definition of Ω, there exists a correct process p_l that is indicated as leader by all oracles of correct processes starting from round $GSR_{\Diamond WLM}$. Since p_l is passed to $compute_{\Diamond LM}$ (and $initialize_{\Diamond LM}()$), the leader indication that these functions see will be constantly p_l starting from the first round $k \geq GSR_{\Diamond WLM}$ in which any of these functions are called. Notice that $k \leq GSR_{\Diamond WLM} + 1$ since if $GSR_{\Diamond WLM}$ is even, $compute_{\Diamond LM}$ will be called in $GSR_{\Diamond WLM}$, and $k = GSR_{\Diamond WLM}$. If $GSR_{\Diamond WLM}$ is odd, then $compute_{\Diamond LM}$ will be called in the next round, i.e. $k = GSR_{\Diamond WLM} + 1$.

In $\Diamond WLM$, starting from round $GSR_{\Diamond WLM}$, the leader p_l is assured to receive a message from a majority of processes. By the simulation code, in every odd round, the leader forwards all received messages to every other process. If $GSR_{\Diamond WLM}$ is odd, in round $GSR_{\Diamond WLM} + 1$ every process will hear from the leader and the $compute_{\Diamond LM}$ function will be called, where it will see messages from a majority sent in the previous round and received in this one from the leader. Similarly, every further invocation of $compute_{\Diamond LM}$ will see majority of messages from every correct process that were actually passed through the leader in $\Diamond WLM$. If $GSR_{\Diamond WLM}$ is even, $compute_{\Diamond LM}$ will still be called, but it is not assured to see messages from a majority, since the leader forwards what it saw in previous round, which was before round $GSR_{\Diamond WLM}$, and therefore the guarantees of the model were not assured to hold in that round. The next $compute_{\Diamond LM}$ is in round $GSR_{\Diamond WLM} + 2$, and only there it is assured to see a message from a majority sent in previous round and forwarded by the leader in this one. Thus, in the worst case, the timeliness guarantees of $\Diamond LM$ will hold starting at round $GSR_{\Diamond WLM} + 2$, and as explained above, by this round the failure detector guarantees hold as well. Thus, $GSR_{\Diamond LM} \leq GSR_{\Diamond WLM} + 2$.

Recall the α-reducibility notion defined in [19]: Model M_2 is α-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 and every $l \in N$, round $GSR_{M_2}(r) + l$ of model M_2 occurs at most in round $GSR_{M_1}(r) + \alpha(l)$ of model M_1.

Lemma 12. *$\Diamond WLM \geq_\alpha \Diamond LM$, where $\alpha(l) = 2l + 2$.*

Proof: By Lemma 11, round $GSR_{\Diamond LM}$ occurs at most at round $GSR_{\Diamond WLM} + 2$. From that round, $compute_{\Diamond LM}()$ is called in every even execution of $compute_{\Diamond WLM}()$. Thus, round $GSR_{\Diamond LM} + 1$ of model $\Diamond LM$ occurs at most at round $GSR_{\Diamond WLM} + 4$ of model $\Diamond WLM$, round $GSR_{\Diamond LM} + 2$ of model $\Diamond LM$ at most at round $GSR_{\Diamond WLM} + 6$ of model $\Diamond WLM$, etc. In general, round $GSR_{\Diamond LM} + l$ of model $\Diamond LM$ occurs at most in round $GSR_{\Diamond WLM} + 2l + 2$ of model $\Diamond WLM$. We get $\alpha(l) = 2l + 2$, and $\Diamond WLM \geq_\alpha \Diamond LM$.

An optimal consensus algorithm for $\Diamond LM$ was presented in [19]. This algorithm reaches global decision by round $GSR_{\Diamond LM} + 2$, i.e. in 3 $\Diamond LM$ rounds. By Lemma 12, there exists a simulation algorithm of $\Diamond LM$ in $\Diamond WLM$ (Algorithm 3), s.t. round $GSR_{\Diamond LM} + 2$ occurs at most at round $GSR_{\Diamond LM} + 2 * 2 + 2 = GSR_{\Diamond LM} + 6$, i.e., global decision is reached in 7 rounds of $\Diamond WLM$.

In Section IV and Appendix I we analyzed the performance of the direct algorithm for $\Diamond WLM$, Algorithm 3, in the common case when the leader is stable and the properties of the oracle are satisfied in round $GSR - 1$, i.e., one round earlier. If we use the simulation-based algorithm for $\Diamond WLM$ presented in this section, no improvement in performance will be achieved, and the algorithm will still take at most 7 rounds, since the worst case is when the timeliness (and not the oracle) properties are satisfied only starting at round $GSR{+}2$ (see proof of Lemma 11). Thus by making the oracle properties hold a round earlier we do not eliminate the worst case discussed in this Lemma.

Note that the requirements of $\Diamond WLM$ are satisfied in $\Diamond LM$, and therefore a simulation of $\Diamond WLM$ in $\Diamond LM$ is trivial. Both models are therefore equivalent by the "classical" notion of CHT [6]. Nevertheless, $\Diamond LM$ inherently requires a $\Omega(n^2)$ message complexity (since each process receives a message from a majority), whereas $\Diamond WLM$ requires only linear message complexity as we have shown in this paper. We therefore think that the "classical" notion of model reducibility and equivalence could be refined to take message complexity into account, similarly to the notion of k-round reducibility [19] that took time (round) complexity of the reduction into account.

Appendix III
Asymptotic Behavior of E(D)

a) ES. : For any fixed $p < 1$, $\lim_{n\to\infty} E(D_{ES}) = \infty$, since $\lim_{n\to\infty} p^{3n^2} = 0$.

b) LM. : For any fixed $p < 1$, its clear that $\lim_{n\to\infty} E(D_{\Diamond LM}) = \infty$, since $\lim_{n\to\infty} p^{3n} = 0$, and $Pr(M|L) \le 1$.

c) WLM. : Similarly to $\Diamond LM$, for any fixed $p < 1$, both the expression in Equation (7) and the one in Equation (8) go to ∞, however Equation (8) grows faster, since the exponent of p is bigger.

d) AFM. : In the following lemma we show that, asymptotically, $E(D_{\Diamond AFM})$ approaches the constant value of 5 rounds, as n, the number of processes, goes to infinity.

Lemma 13. *For a fixed $p > \frac{1}{2}$, $\lim_{n\to\infty} E(D_{\Diamond AFM})$= 5*

Proof: To bound the probability that A has a majority of 1's in a row, we use a Chernoff bound [7]: Let $X_1, X_2, ..., X_n$ and X be as defined above, and denote $\mu = E(X) = np$. By the Chernoff bound, for any $0 < \epsilon < 1$:

$$P(X \le (1-\epsilon)\mu) < e^{-\mu\epsilon^2/2}$$

We would like to bound the probability $P(X \le \frac{n}{2})$ and therefore take $\epsilon = (1 - \frac{1}{2p})$. Thus, for $p > 1/2$, we get:

$$P(X \le \frac{n}{2}) \le e^{-(1-\frac{1}{2p})^2 np/2}$$
$$P(X > \frac{n}{2}) > 1 - e^{-(1-\frac{1}{2p})^2 np/2}$$

This is a bound on the probability that any given row in A has a majority of 1's. For n (independent) rows, we get that the probability exceeds $(1-e^{-(1-\frac{1}{2p})^2 np/2})^n$. As was already explained, if we take p as the lower bound for the probability that given a majority of ones in each row, any given entry in A is 1, we have to raise this expression to the power of 2. Additionally, this needs to hold for 5 consecutive rounds:

$$E(D_{\Diamond AFM}) \le \frac{1}{(1-e^{-(1-\frac{1}{2p})^2 np/2})^{10n}} + 4$$

For a fixed $p < 1$, the first expression in the sum above approaches 1 as $n \to \infty$, and therefore $E(D_{\Diamond AFM}) \to 5$.

Acknowledgments

We thank Hagit Attiya, Paul Ezhilchelvan and Liran Katzir for helpful discussions and valuable comments.

References

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

[2] T. Anker, D. Dolev, G. Greenman, and I. Shnayderman. Evaluating total order algorithms in WAN. In *Int. Workshop on Large-Scale Group Communication*, 2003.

[3] O. Bakr and I. Keidar. Evaluating the running time of a communication round over the Internet. In *PODC*, 2002.

[4] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services, 2004.

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

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

[7] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Ann. Math. Statist.*, 23:493–507, 1952.

[8] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. In *IEEE TPDS*, June 1999.

[9] D. Dobre, M. Majuntke, and N. Suri. CoReFP: Contention-Resistant Fast Paxos for WANs. Technical report, TU Darmstadt, Germany, 2006.

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

[11] P. Dutta, R. Guerraoui, and I. Keidar. The overhead of consensus failure recovery. Technical Report 200456, EPFL, 2004.

[12] P. Dutta, R. Guerraoui, and I. Keidar. The Overhead of Indulgent Failure Recovery. *Distributed Computing*, 2006.

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

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

[15] A. Fernandez, E. Jimenez, and M. Raynal. Eventual leader election with weak assumptions on initial knowledge, communication reliability, and synchrony. In *DSN*, 2006.

[16] R. Guerraoui. Revisiting the relationship between non blocking atomic commitment and consensus problems. In *WDAG-9*, number 791 in LNCS, pages 87–100. Springer-Verlag, Sept. 1995.

[17] R. Guerraoui. Indulgent algorithms. In *PODC*, 2000.

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

[19] I. Keidar and A. Shraer. Timeliness, failure-detectors, and consensus performance. In *PODC*, 2006.

[20] L. Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2, 1978.

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

[22] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *SRDS*, pages 52–59, 2000.

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

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

[25] U. Schmid. Failure model coverage under transient link failures. Research Report 2/2004, Technische Universität Wien, 2004.

[26] P. Urban, I. Shnayderman, and A. Schiper. Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms. *DSN*, 2003.