Time Bounds for Decision Problems in the Presence of Timing Uncertainty and Failures[1]

Hagit Attiya and Taly Djerassi-Shintel
Computer Science Department, The Technion, Haifa 32000, Israel

This paper studies the time complexity of solving decision problems in a distributed message-passing system, when there is inexact information about time and process failures. A *semi-synchronous* model is assumed, in which the amount of (real) time between two consecutive steps of a nonfaulty process is at least c_1 and at most c_2; a message sent by a nonfaulty process is delivered within time at most d. Faulty processes do not always obey the timing requirements on their steps and on messages they send (*late timing* failures).

We present a new stretching technique for deriving lower bounds in the presence of late timing failures. The combination of the stretching technique with known results yields the following lower bounds for this model:

1. The worst-case running time of any comparison-based renaming algorithm for $p \leq n$ participants is $\Omega(\log p \frac{c_2}{c_1} d)$, when there are $p-1$ late timing failures. A similar result holds, under certain restrictions, also for renaming algorithms which are not comparison-based.

2. The worst-case running time of any consensus algorithm is $\Omega(\frac{n}{n-f} \frac{c_2}{c_1} d)$, when there are $f < n$ late timing failures.

3. The worst-case running time of any k-set consensus algorithm is $\Omega(\frac{n}{n-f} \frac{1}{k} \frac{c_2}{c_1} d)$, when there are $f < n$ late timing failures.

Key Words: semi-synchronous model, renaming, consensus, k-set consensus, late timing failures

1. INTRODUCTION

In the *semi-synchronous* model of distributed computing, the amount of time between two consecutive steps of any nonfaulty process is at least c_1 and at most c_2,

[1] A preliminary version of this paper appeared in proceedings of the *7th International Workshop on Distributed Algorithms 1993*, (A. Schiper, Ed.), pp. 204–218, Lecture Notes in Computer Science #725, Springer-Verlag. This work was supported by grant No. 92-0233 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, and the fund for the promotion of research in the Technion.

and the time for delivering a message sent by a nonfaulty process is at most d. (The constants c_1, c_2 and d are known to the processes.) This is an intermediate model between the synchronous and the asynchronous models; it captures distributed systems in which the amount of time required for processes to take steps, for clocks to advance, and for messages to be delivered, are only approximately known. Adverse situations, such as an overloaded operating system, congested message buffers, or hardware malfunctioning, can cause the time between consecutive steps of a process to be more than c_2 or the delay of its messages to be more than d. This behavior is modeled by *late timing* failures [8].

We study the time complexity of solving *renaming*, *consensus* and *k-set consensus*, in a semi-synchronous system with n processes, in the presence of late timing failures. (*Early timing* failures, where the steps of a faulty process can occur faster than c_1, are not addressed here.)

In the semi-synchronous model, late timing failures can be detected by a *timeout* mechanism [3]: If an expected message from some process is not received within d time units, then that process is known to have failed. By counting its own steps, a process can detect the failure of another process within time approximately $c_2 \cdot d/c_1 + d$ (see Section 4.1). The ratio $\frac{c_2}{c_1}$ is denoted C; it captures the *timing uncertainty* of the model.

In the *strong renaming* problem [1], p processes are required to terminate with distinct outputs ('names') in $\{1, \ldots, p\}$. In the synchronous model, the optimal algorithm for solving renaming in the presence of $p-1$ crash failures uses $\Theta(\log p)$ communication rounds [13]. (The lower bound is proved for *comparison-based* algorithms, which only test processes' id's for equality and order.)

In the *consensus* problem [16], processes are required to output the same value; if a nonfaulty process outputs v then v must be the input of some process. In the synchronous model, exactly $f+1$ communication rounds are required for reaching consensus in the presence of f crash failures, for any $f < n$ [10]. In the semi-synchronous model, there are algorithms for reaching consensus in the presence of f crash [3] or send omission [17] failures within time $O(fd + Cd)$; $fd + Cd$ is also a lower bound on the time for reaching consensus in the presence of crash, and hence, send omission failures [3]. In contrast, the running time of the best consensus algorithm tolerating Byzantine failures is $O(fCd)$ [17].

In the *k-set consensus* problem [5], processes are required to output at most k *different* values; if a nonfaulty process outputs v then v must be the input of some process. The consensus problem is a special case where $k = 1$. In the synchronous model, exactly $\frac{f}{k}+1$ communication rounds are required for solving k-set consensus in the presence of f crash failures, for any $f < n$ [5, 6]. For the semi-synchronous model, it was recently shown that $\lfloor \frac{f}{k} - 1 \rfloor d + Cd$ is a lower bound on the time to solve k-set consensus in the presence of f crash failures [14].

We present a new *stretching* technique which exploits late timing failures, and use it to prove new lower bounds for the semi-synchronous model:

1. An $\Omega(\log p\, Cd)$ lower bound on the worst-case running time of any comparison-based algorithm for renaming in the presence of $p-1$ late timing failures, for any $p < n$. (The proof uses ideas from [13].) The same result is proved for algorithms which are not comparison-based, assuming that p is much smaller than n.

2. An $\Omega(\frac{n}{n-f}Cd)$ lower bound on the worst-case running time of any consensus algorithm in the presence of f late timing failures. (This relies on a result of [3].)

3. An $\Omega(\frac{n}{n-f}\frac{1}{k}Cd)$ lower bound on the worst-case running time of any k-set consensus algorithm in the presence of f late timing failures. (This extends the previous lower bound, and relies on a result of [6].)

Our results show a problem—renaming—which requires $\Theta(\log p)$ communication rounds in the synchronous model, but its time complexity in the semi-synchronous model is $\Theta(\log p\ Cd)$, in the presence of $p-1$ late timing failures. Thus, any renaming problem that tolerates late timing failures must incur an overhead of $\Omega(Cd)$ per round. For the semi-synchronous model, there is a renaming algorithm that tolerates crash failures, with a worst-case running time of $O(\frac{\log n}{\log\log n}(c_2 \log n + Cd))$ [9]. Thus, there is no need to incur an overhead of Cd per round when only crash failures have to be tolerated.

When $f = n-1$, there is a significant gap (depending on n and C) between the time required to solve consensus in the presence of crash and send omission failures and the time required for solving consensus in the presence of late timing failures.

Ponzio mentions [18, Ch. 6], without proof, a lower bound of $\left(\lceil\frac{f}{n-f}\rceil + \lfloor\frac{f}{n-f}\rfloor\right)Cd$ on the worst-case running time of any consensus algorithm in the semi-synchronous model. This result requires $f \geq n/2$ authenticated Byzantine failures, which allow more faulty behavior than late timing failures; therefore, it does not imply the lower bound we prove for consensus. Bazzi [4] proves that any translation from the synchronous model to the semi-synchronous model must incur an overhead of $\left(\lceil\frac{f}{n-f}\rceil + \lfloor\frac{f}{n-f}\rfloor\right)Cd$ per round. His proof requires arbitrary timing failures, which allow more faulty behavior than late timing failures; therefore, it does not imply the lower bounds we prove. Moreover, Bazzi's result does not imply that algorithms for *specific* problems, e.g., renaming, must incur this overhead.

2. THE COMPUTATION MODEL

We use the model defined in [3], adapted to include late timing failures. A *system* consists of n processes $p_1, \ldots, p_n$, and their respective message buffers, $buff_1$, ..., $buff_n$. Each process p_i is modeled as a (possibly infinite) state machine with a local state set Q_i, including a distinguished *initial state*.

A *configuration* is a vector $s = ((q_1, b_1) \ldots, (q_n, b_n))$ where $state_i(s) = q_i$ is the local state of p_i and $buff_i(s) = b_i \subseteq \mathcal{M}$ is the content of p_i's buffer. In the *initial configuration* all processes are in their initial states and all buffers are empty.

Processes communicate by sending *messages* taken from some alphabet $\mathcal{M}$. We assume that messages sent from p_i to p_j contain a sequence number and that the sender's id is part of every message. The action $send(p_j, m)$ represents the sending of message m to process p_j.

Each process p_i follows a deterministic algorithm that governs its state transitions and the messages it sends. The algorithm consists of a transition function γ_i, for each process p_i; for every $q \in Q_i$ and $buff_i \subseteq \mathcal{M}$, $\gamma_i(q, buff_i)$ gives a state q' and a finite set of send actions, S.

The possible events in the system are either *computation events* of the form $comp(p_i, S)$, where p_i is a process and S is a set of send actions, or *delivery events* of the form $deliv(p_i, m)$, where p_i is a process and m is a message.

An *execution* is an infinite sequence of alternating configurations and events $\alpha = C_0, \pi_1, C_1, \ldots, \pi_r, C_r, \ldots$, satisfying the following conditions:

1. C_0 is the initial configuration.

2. If π_r is an event of process p_i, then $state_j(C_{r-1}) = state_j(C_r)$ and $buff_j(C_{r-1}) = buff_j(C_r)$ for every $j \neq i$. That is, states and buffers of processes other than p_i do not change.

3. If $\pi_r = comp(p_i, S)$, then $state_i(C_r)$ and S are obtained by applying γ_i to $state_i(C_{r-1})$ and $buff_i(C_{r-1})$; furthermore, $buff_i(C_r) = \emptyset$. That is, p_i, based on its local state and the contents of its buffer, performs the send actions in S, clears its buffer and possibly changes its local state, all in one atomic transition.

4. If $\pi_r = deliv(p_i, m)$, then $state_i(C_r) = state_i(C_{r-1})$ and $buff_i(C_r) = buff_i(C_{r-1}) \cdot \{m\}$. That is, the message m is appended to p_i's buffer.

5. For every delivery event $\pi_r = deliv(p_i, m)$ there is exactly one computation event $\pi_l = comp(p_j, S)$ where $l < r$ and $send(p_i, m) \in S$. That is, each delivery is matched to a unique earlier send.

Below, 'time' is always a nonnegative real number. A *timed event* is a pair (π, t), where π is an event and t is a time. A *timed execution* is an infinite sequence of alternating configurations and timed events $\alpha = C_0, (\pi_1, t_1), C_1, \ldots, (\pi_r, t_r), C_r, \ldots$, where $C_0, \pi_1, C_1, \ldots, \pi_r, C_r, \ldots$ is an execution and the times are nondecreasing and unbounded.

Fix real numbers c_1, c_2, and d, $0 < c_1 \leq c_2 < \infty$ and $0 < d < \infty$. A process p_i is *nonfaulty* in a timed execution α if the following conditions hold:

1. There is a computation event $comp(p_i, S)$ at time 0.

2. If the lth and rth timed events, $l < r$, are both computation events of p_i with no intervening computation events of p_i, then $c_1 \leq t_r - t_l \leq c_2$.

3. If a message m is sent by p_i to p_j at the lth timed event then there exists $r > l$ such that the rth timed event is the matching delivery $deliv(p_i, m)$, and $t_r - t_l \leq d$.

Algorithms are required to tolerate only *late timing* failures. A *faulty* process follows the algorithm, but the time difference between its consecutive computation steps can be larger than c_2, and messages it sends can be delayed more than d time units (or not delivered at all).

Two other failure types are briefly mentioned in this paper: In a *crash* failure, a faulty process stops executing computation events from a certain point in the execution and further on; only a subset of the messages sent in the last computation event of p_i is delivered. In a *send omission* failure, some of the messages sent by a faulty process are not delivered.

3. STRETCHING TIMED EXECUTIONS

This section describes a technique for using late timing failures to *stretch* timed executions and multiply their time by roughly C. Stretching is applied only to *fast, round-structured* timed executions[2] in which: processes take steps in lockstep as

[2] Originally [2, 9], stretching was applied to *sets* of processes; fast, round-structured executions correspond to *fast, half-uniform executions with singleton sets*, in the original terminology.

quickly as possible (at multiples of time c_1); the timed execution is partitioned into rounds, each taking approximately time d; and messages sent during a round are delivered at the very end of that round.

In more detail, let $\hat{d} = c_1\lceil \frac{d}{c_1} \rceil$; note that $C\hat{d} = c_2\lceil \frac{d}{c_1} \rceil$ and $\hat{d} \geq d$. A fast, round-structured timed execution satisfies the following conditions (Figure 1):

1. Each process either takes its first step at time 0 or does not take any step.

2. Each process either takes a step every c_1 time units or does not take any further steps.

3. A message sent at time t, $r\hat{d} \leq t < r\hat{d} + d$, for some integer r, is delivered at time $r\hat{d} + d$.

4. Messages which are delivered at the same time are delivered in the order they are sent, breaking ties using the id's of sending processes.

Let α be a fast, round-structured timed execution. For every integer $r \geq 1$, *round* r of α consists of all timed events (π, t) such that $(r-1)\hat{d} \leq t < r\hat{d}$.

Assume that process p_g is nonfaulty in α; we construct a timed execution α', which is a 'stretched' version of α, as follows (Figure 2):

1. Processes that take a step at time 0 in α, take a step at time 0 in α'.

2. Process p_g takes a step every c_2 time units.

3. A message sent by p_g at time $rC\hat{d} + \ell c_2$, $0 \leq \ell \leq \lceil \frac{d}{c_1} \rceil - 1$, is delivered at time $rC\hat{d} + d + \ell d/\hat{d}(c_2 - c_1) \leq rC\hat{d} + Cd$. Note that the delay of this message is $\leq d$.

4. For every integer $r \geq 0$, if process p_i, $i \neq g$, takes $\ell > 0$ steps in round r of α, then it takes ℓ steps at times $rC\hat{d}, rC\hat{d} + c_1, \ldots, rC\hat{d} + (\ell - 1)c_1 \leq rC\hat{d} + d$; p_i does not perform additional steps until time $(r+1)C\hat{d}$.

5. A message sent by process p_i, $i \neq g$, at time $rC\hat{d} + \ell c_1$, $0 \leq \ell \leq \lceil \frac{d}{c_1} \rceil - 1$, is delivered at time $rC\hat{d} + Cd$, if sent to process p_g, otherwise, it is delivered and at time $rC\hat{d} + d + \ell d/\hat{d}(c_2 - c_1) \leq rC\hat{d} + Cd$.

6. Messages which are delivered at the same time are delivered in the order they are sent, breaking ties using the id's of sending processes.

7. Messages that are not delivered in α are not delivered in α'.

Process p_g is nonfaulty in α' since it is nonfaulty in α and it obeys the timing requirements in α'. Other processes may experience late timing failures in α', e.g., their messages delayed more than d time units, even if they are nonfaulty in α.

An event performed in round r of α is performed at time t, $(r-1)C\hat{d} \leq t < rC\hat{d}$, in α'; these events form *round* r of α'. A round of α' takes time $c_2\lceil \frac{d}{c_1} \rceil = C\hat{d}$, while a round of α takes time $\hat{d}$; thus, α' "stretches" α.

The next lemma shows that processes do not distinguish between α and α'. Let D_r be the first configuration after all the events of rounds $1, \ldots, r$ in α. Similarly, D'_r is the first configuration after all the events of rounds $1, \ldots, r$ in α'. D_0 and D'_0 denote the initial configurations of α and α', respectively.

STRETCHING LEMMA. For every process p_i and every $r \geq 0$, $state_i(D_r) = state_i(D'_r)$ and $buff_i(D_r) = buff_i(D'_r)$.

Proof. The proof is by induction on r. The lemma holds for the base case, $r = 0$, since every process is in its initial state and its buffer is empty in the initial configurations of α and α'.

Assume that $state_i(D_{r-1}) = state_i(D'_{r-1})$ and $buff_i(D_{r-1}) = buff_i(D'_{r-1})$, for every process p_i. Thus, p_i performs the same transition in its first step of round r in α and α'. Hence, p_i is in the same state and its buffer is empty after its first step of round r in α and α'; moreover, p_i sends the same messages in its first step of round r in α and α'. No messages are delivered to p_i in its computation steps of round r and hence, p_i sends the same messages during round r in α and α'; this also implies that $state_i(D_r) = state_i(D'_r)$.

Clearly, the same set of messages is delivered to p_i between its last computation step in round r and its first computation step in round $r+1$, in α and in α'.

In α, all messages sent to p_i in the ℓth step of some process in round r are delivered at the same time $(r\hat{d} + d)$. This also happens in α': All messages sent to p_i at the ℓth step of some process in round r are delivered at same time ($rC\hat{d} + Cd$, if $i = g$, and $rC\hat{d} + d + \ell d/\hat{d}(c_2 - c_1)$, if $i \neq g$). Therefore, messages are delivered in the same order in α and in α' and hence, $buff_i(D_r) = buff_i(D'_r)$. ■

In the following lower bounds, we first construct a timed execution in which a nonfaulty process p_g does not output before time $r\hat{d}$, for some $r > 0$. The stretching lemma is then applied to show that there is a timed execution in which p_g does not output before time $rC\hat{d}$.

4. THE STRONG RENAMING PROBLEM

Assume that each process has a binary input register, indicating 'don't participate' or 'participate' and an output register. An algorithm *solves strong renaming in the presence of* $p-1$ *failures* if in every execution in which at most p processes participate and at most $p-1$ processes fail, nonfaulty participating processes output *distinct* values in the range $\{1, \ldots, p\}$.

4.1. Algorithm

There is a synchronous algorithm solving strong renaming [13]; it requires $\log_2 p + 2$ rounds and tolerates $p-1$ crash failures. The transformation of Neiger and Toueg [15] can be applied to make the algorithm tolerate the same number of send omission failures, by doubling the number of rounds.

The timeout technique of Attiya et al. [3, Sec. 4] is used to simulate a synchronous algorithm tolerating send omission failures by a semi-synchronous algorithm tolerating late timing failures. (Attiya et al. simulate a synchronous algorithm tolerating crash failures by a semi-synchronous algorithm tolerating crash failures.)

Processes broadcast *alive* messages, tagged with sequence numbers, in every computation event. Each process p_i collects a set of timed out processes, using a separate timeout task; p_i times out another process p_j if it counts $\lceil \frac{d+c_2}{c_1} \rceil$ steps without receiving the next *alive* message from p_j. The following properties hold:

1. If p_i times out p_j, then p_j is faulty.

2. If p_j sends a message to a nonfaulty process p_i at time t, then p_i either receives the message or times out p_j no later than time $t + c_2\lceil \frac{d+c_2}{c_1} \rceil + d + c_2$.

Let A be a synchronous algorithm tolerating send omission failures. A process p_i simulates A round by round: To simulate the first round of A, p_i sends the round-1 message of A, tagged with 1; p_i then waits to receive round-1 messages from all processes that were not timed out yet and then broadcasts the round-2 message of A, tagged with 2. Subsequent rounds are handled similarly.

Property 1 implies that no process sends a round-r message before receiving a round-$(r-1)$ message from all nonfaulty processes; thus, we accurately simulate A. A nonfaulty process p_i may detect process p_j as faulty and not wait for its round-r message, while another nonfaulty process p'_i may receive a round-l, $l > r$, message from p_j. The simulated algorithm, A, copes with this situation since it tolerates send omission failures.

Induction on the round number $r \geq 1$, using Property 2, shows that a process sends its round-r message no later than time $r(c_2\lceil\frac{d+c_2}{c_1}\rceil + d + c_2)$.

THEOREM 4.1. *There is a semi-synchronous renaming algorithm tolerating* $p-1$ *late timing failures, with worst-case running time* $2\lceil\log_2 p + 2\rceil(c_2\lceil\frac{d+c_2}{c_1}\rceil + d + c_2)$.

4.2. Lower Bound for Comparison-Based Algorithms

We first restrict our attention to *comparison-based* algorithms in which the only operations a process can perform on process id's is testing for equality and order; it cannot examine the structure of the id's in any more detail (cf. [11, 13]).

We assume that algorithms are in *full-information* form, that is, a process's local state consists of its id and the history of messages it received; a process broadcasts its current local state in every step. For a full-information algorithm, we can represent a process' local state as a nested list: The initial state of p_i is simply its id, p_i. Later states are the expressions $q_i = (q'_i, (m_1, \ldots, m_k))$ where q'_i is the previous state, and $m_1, \ldots, m_k$ are the messages delivered to p_i since its last step, in order of delivery.

DEFINITION 4.1. Two states q and q' are *order-equivalent* if
(1) they have the same structure as nested lists, and
(2) if two id's from q satisfy one of the order relations $<, =$ or $>$, then the corresponding id's in q' satisfy the same relation.

Intuitively, if two processes are in order-equivalent states then a comparison-based algorithm instructs them to perform the same action. In a full-information algorithm, the only interesting transition is whether a process outputs a value in a step and which value.

DEFINITION 4.2. An algorithm A is *comparison-based* if every pair of processes, p_i and p_j, which are in order-equivalent states, make the same transition. In particular, p_i outputs v if and only if p_j outputs v.

To prove the lower bound, fix a comparison-based renaming algorithm, A. We first construct α, a fast, round-structured timed execution of A in which some nonfaulty process outputs after time $\lfloor\log_3 p\rfloor\hat{d}$. This inductive construction is an

adaptation of the 'sandwich' failure pattern of Herlihy and Tuttle [13]; it guarantees that there are $\frac{p}{3^r}$ nonfaulty processes which are in order-equivalent states after round r of α, for every r, $1 \leq r \leq \lfloor \log_3 p \rfloor$.

Let $l_r = 3^{\lfloor \log_3 p \rfloor - r}$, $0 \leq r \leq \lfloor \log_3 p \rfloor$; note that $l_{r-1} = 3l_r$. Define a sequence $b_0, b_1, \ldots$ as follows: $b_0 = 0$ and $b_r = b_{r-1} + l_r$ for every r, $1 \leq r \leq \lfloor \log_3 p \rfloor$.

We construct a fast, round-structured timed execution α with $\lfloor \log_3 p \rfloor$ rounds such that for every r, $0 \leq r \leq \lfloor \log_3 p \rfloor$, processes $p_{b_r+1}, \ldots, p_{b_r+l_r}$ are nonfaulty and in order-equivalent states in D_r. Recall that D_o is the initial configuration of α and D_r is the first configuration after all events of rounds $1, \ldots, r$ in α.

Processes $p_1, \ldots, p_{3^{\lfloor \log_3 p \rfloor}}$ participate in α; all non-participating processes fail at the beginning of α, before they take a step or send a message. Clearly, processes $p_{b_0+1}, \ldots, p_{b_0+l_0}$ are nonfaulty and in order-equivalent states in D_0.

Assume we have constructed α up to round $r-1$. Processes $p_{b_{r-1}+1}, \ldots, p_{b_{r-1}+l_{r-1}}$ (that is, processes $p_{b_{r-1}+1}, \ldots, p_{b_{r-1}+3l_r}$) are nonfaulty and in order-equivalent states in D_{r-1}, $1 \leq r \leq \lfloor \log_3 p \rfloor$.

In round r, process $p_{b_{r-1}+l_r+j}$, for every j, $1 \leq j \leq l_r$, receives messages only from processes $p_{b_{r-1}+j}, \ldots, p_{b_{r-1}+2l_r+j}$; all other messages are not delivered (see Figure 3). Clearly, processes $p_{b_{r-1}+1}, \ldots, p_{b_{r-1}+l_r}$ (with lowest id's) and processes $p_{b_{r-1}+2l_r+1}, \ldots, p_{b_{r-1}+3l_r}$ (with highest id's) are faulty. From round $r+1$ on, all messages from these processes are not delivered.

By the construction, processes $p_{b_{r-1}+l_r+1}, \ldots, p_{b_{r-1}+2l_r}$, that is, $p_{b_r+1}, \ldots, p_{b_r+l_r}$, are nonfaulty before round $r+1$. A nonfaulty process receives messages from exactly $2l_r$ processes, and its rank among them is exactly $l_r + 1$; this is the key to proving the next lemma:

LEMMA 4.1. *For every r, $0 \leq r \leq \lfloor \log_3 p \rfloor$, processes $p_{b_r+1}, \ldots, p_{b_r+l_r}$ are in order-equivalent states in D_r,*

Proof. The proof is by induction on r. In the base case, we consider the initial configuration, D_0. The state of a process in D_0 is its id; hence, $p_{b_0+1}, \ldots, p_{b_0+l_0}$ are in order-equivalent states.

For the induction step, assume that the lemma holds for D_{r-1}, $0 \leq r < \lfloor \log_3 p \rfloor$, and consider D_r. Let M_i be the sequence of messages sent by process p_i during round r, in the order they were sent. By the induction hypothesis, M_i and M_j are order-equivalent, for every pair of processes p_i and p_j, $b_{r-1} + 1 \leq i, j \leq b_{r-1} + 3l_r$.

Consider two nonfaulty processes p_i and p_j, in states q_i and q_j, respectively, in D_{r-1}; by the induction hypothesis, q_i and q_j are order-equivalent. The first computation step of round r is performed after the delivery events of all messages sent by other processes in round $r-1$. Therefore, after their first step of round r in α, the state of p_i is $(q_i, (M_{b_{r-1}+i}, \ldots, M_{b_{r-1}+2l_r+i}))$ and the state of p_j is $(q_j, (M_{b_{r-1}+j}, \ldots, M_{b_{r-1}+2l_r+j}))$. These states are order-equivalent, by the induction hypothesis. No message is delivered during round r, and hence p_i and p_j are in order-equivalent states after every step of round r. ■

Assume some nonfaulty process outputs at time $t < \lfloor \log_3 n \rfloor \hat{d}$, that is, in some round $r < \lfloor \log_3 p \rfloor$. By the construction, $\frac{p}{3^r} > 1$ processes are nonfaulty in round r of α (since $r < \lfloor \log_3 p \rfloor$). By Lemma 4.1, the nonfaulty processes are in order-

equivalent states during round r of α and hence, they output the same name. This contradicts the uniqueness condition for renaming, and proves the next lemma:

LEMMA 4.2. *In α, no nonfaulty process outputs strictly before time $\lfloor \log_3 p \rfloor \hat{d}$.*

By the construction, some process p_g is nonfaulty in α. By the stretching lemma, there is a timed execution α' in which p_g does not output before time $\lfloor \log_3 p \rfloor C\hat{d}$ in α', proving the next lower bound:

THEOREM 4.2. *The worst-case running time of any comparison-based algorithm solving renaming, for p participating processes in the presence of late timing failures, is at least $\lfloor \log_3 p \rfloor C\hat{d}$.*

4.3. Lower Bound for General Algorithms

Consider some set of processes, P; an algorithm A is *comparison-based for P* if A is comparison-based in all fast, round-structured timed executions in which only processes from P participate. We use Ramsey's theorem to pick the participating processes from a very large set of processes; the proof closely follows [11].

LEMMA 4.3. *Fix some integer p, and assume A is a renaming algorithm for p participating processes whose worst-case running time is strictly less than $\lfloor \log_3 p \rfloor C\hat{d}$. Then there is an integer $N = N(p)$ and a subset P of $[1..N]$, $|P| = p$, such that A is comparison-based for P.*

Proof. Let X and Y be two p-subsets of process id's; that is, sets containing exactly p process id's. Let $X = \{x_1, \ldots, x_p\}$ and $Y = \{y_1, \ldots, y_p\}$ be their representations in increasing order of id's. Two states s and s' are *output-equivalent* if the output of process x_i with state s is the same as of process y_i with state s', for every i, $1 \leq i \leq p$; that is, x_i outputs if and only if y_i outputs and if so, they output the same value. (Since A is in full-information form, this is the only meaningful decision a process can make.)

Given a nested list, s, containing only id's from X, its *Y-substitute*, s', is created by replacing each id x_i with the id y_i; the *X-substitute* of a nested list containing only id's from Y is created similarly. Note that s and s' are order-equivalent.

Fix an algorithm A, and partition all p-subsets of integers into equivalence classes as follows. Let X and Y be two p-subsets; X and Y are *output-equivalent* if every state s that occurs in some fast, round-structured timed execution of A where only processes from X participate is output-equivalent to its Y-substitute, denoted s', and vice versa. Note that s' occurs in a fast, round-structured timed execution of A where only processes from Y participate.

In A, processes output in time strictly less than $\lfloor \log_3 p \rfloor C\hat{d}$ and hence, A has only a finite number of combinations of outputs in fast, round-structured timed executions. Therefore, output-equivalence partitions the p-subsets into a finite number of equivalence classes. We use the following version of Ramsey's theorem [12]:

RAMSEY'S THEOREM. *For an integer p, partition all p-subsets of integers into a finite number of equivalence classes. There is an integer $N = N(p)$ and a $2p$-subset of $[1..N]$, B, such that all p-subsets of B are in the same equivalence class.*

Picking $N = N(p)$ as in Ramsey's theorem, there is a $2p$-subset of $[1..N]$, P', such that all p-subsets of P' are in the same equivalence class, i.e., they are all output-equivalent. Let P be the set containing the p smallest elements of P'; we claim that A is comparison-based for P.

Consider two order-equivalent states q_1 and q_2 occuring in some fast, round-structured timed execution of A, in which only processes from P participate.

Let X_1 and X_2 be the subsets of processes from P that appear in q_1 and q_2, respectively. Since q_1 and q_2 are order-equivalent and in particular, structurally equivalent, $|X_1| = |X_2| = m$; since only processes from P participate in the execution, $m \leq p$. Clearly, q_2 is the X_2-substitute of q_1.

Extend X_1 and X_2 into p-subsets of P', denoted X_1' and X_2' (respectively), by adding the $p - m$ largest elements of P'. Note that q_2 is also the X_2'-substitute of q_1. Since P' was picked by an application of Ramsey's theorem, and X_1' and X_2' are p-subsets of P', it follows that X_1' and X_2' are output-equivalent. Thus, the decisions in q_1 and q_2 are equal, since q_2 is the X_2'-substitute of q_1. Therefore, A is comparison-based in all fast, round-structured timed executions in which only processes from P participate. ■

Together with the lower bound construction of Section 4.2 and the stretching technique of Section 3, this implies:

THEOREM 4.3. *For any p, there is an integer $N = N(p)$, such that the worst-case running time of any algorithm solving renaming, for p participating processes out of N processes in the presence of late timing failures, is at least $\lfloor \log_3 p \rfloor C\hat{d}$.*

5. THE CONSENSUS AND k-SET CONSENSUS PROBLEMS

Assume each process p_i has an input register $v_i \in \{0, 1\}$ and an output register. An algorithm *solves consensus in the presence of f failures* if in every execution in which at most f processes fail, all nonfaulty processes output the same value $v \in \{v_1, \ldots, v_n\}$.

The transformation of Section 4.1 can be applied to any synchronous consensus algorithm tolerating crash failures within $f+1$ rounds, e.g., [10]. This yields a semi-synchronous consensus algorithm tolerating f late timing failures whose worst-case running time is $O(f(c_2\lceil \frac{d+c_2}{c_1} \rceil + d + c_2))$, for any $f < n$.[3]

To prove the lower bound, first consider the special case of an m-process algorithm solving consensus in the presence of $m-1$ late timing failures. By Lemma 6.2 of [3], the algorithm has a fast, round-structured timed execution in which some process p_g is nonfaulty and does not output before time $m\hat{d}$. The stretching lemma implies:

[3] There is an algorithm solving consensus in the semi-synchronous model in the presence of Byzantine failures with $O(fCd)$ worst-case running time [17]; however it requires that $n > 3f$.

LEMMA 5.1. *For any $m \geq 2$, if an algorithm solves consensus for m processes in the presence of $m-1$ late timing failures, then it has a timed execution in which some nonfaulty process does not output strictly before time $mC\hat{d}$.*

Assume that A solves consensus for n processes, in the presence of $f < n$ late timing failures. Partition the processes into disjoint sets, each of size $n-f$, denoted $S_1, \ldots, S_{\lfloor \frac{n}{n-f} \rfloor}$. Consider the following algorithm A' for $n' = \lfloor \frac{n}{n-f} \rfloor$ processes: Process p_i performs the actions of algorithm A for all $n-f$ processes in the set S_i (in parallel); processes in S_i have the same initial value v_i (p_i's input value) and the messages between them are not delayed. (If $n-f$ does not divide n, then the remaining processes fail before they take a step or send a message.)

If some process is nonfaulty in an execution of A', then A' simulates an execution of A in which at least $n-f$ processes are nonfaulty. Thus, if A solves consensus for n processes and $f < n$ late timing failures within time $\lfloor \frac{n}{n-f} \rfloor C\hat{d}$ then A' solves consensus for n' processes and $n'-1$ late timing failures processes within time $\lfloor \frac{n}{n-f} \rfloor C\hat{d} = n'C\hat{d}$. This contradicts Lemma 5.1 and implies the lower bound:

THEOREM 5.1. *The worst-case running time of any algorithm that solves consensus is at least $\lfloor \frac{n}{n-f} \rfloor C\hat{d}$.*

The k-set consensus problem is an extension of the consensus problem in which processes have to output at most k different values [5]. (Clearly, the problem is interesting only if there are more than k different inputs.) The transformation of Section 4.1 can be applied to the k-set consensus algorithm which requires $\lfloor \frac{f}{k} \rfloor + 1$ rounds [5, 6]; this yields a semi-synchronous algorithm whose worst-case running time is $O(\frac{f}{k}(c_2\lceil \frac{d+c_2}{c_1} \rceil + d + c_2))$, for any $f < n$.

As for consensus, we first consider an m-process algorithm solving k-set consensus in the presence of $m-1$ late timing failures. Lemma 21 of [14] implies that this algorithm has a fast, round-structured timed execution in which some nonfaulty process does not output before time $(\lfloor \frac{m-1}{k} \rfloor + 1)\hat{d}$. The stretching lemma implies:

LEMMA 5.2. *For any $m \geq 2$, if an algorithm solves k-set consensus for m processes in the presence of $m-1$ late timing failures, then it has a timed execution in which some nonfaulty process does not output before time $(\lfloor \frac{m-1}{k} \rfloor + 1)C\hat{d}$.*

Assume that A solves k-set consensus for n processes, in the presence of $f < n$ timing failures. As for consensus, partition the processes into disjoint sets $S_1, \ldots, S_{\lfloor \frac{n}{n-f} \rfloor}$, each of size $n-f$, and consider an algorithm A' for $n' = \lfloor \frac{n}{n-f} \rfloor$ processes in which p_i simulates the set S_i.

If some process is nonfaulty in the execution of A', then it simulates an execution of A in which at least $n-f$ processes are nonfaulty. Thus, if A solves k-set consensus for n processes and $f < n$ late timing failures within time $(\lfloor (\lfloor \frac{n}{n-f} \rfloor - 1)\frac{1}{k} \rfloor + 1)C\hat{d}$ then A' solves k-set consensus for n' processes and $n'-1$ late timing failures processes within time $(\lfloor (\lfloor \frac{n}{n-f} \rfloor - 1)\frac{1}{k} \rfloor + 1)C\hat{d} = (\lfloor \frac{n'-1}{k} \rfloor + 1)C\hat{d}$. This contradicts Lemma 5.2 and implies the lower bound:

THEOREM 5.2. *The worst-case running time of any algorithm that solves k-set consensus is at least* $(\lfloor(\lfloor\frac{n}{n-f}\rfloor - 1)\frac{1}{k}\rfloor + 1)C\hat{d}$.

ACKNOWLEDGMENT

We thank Rinat Rappoport and Jennifer Welch for valuable feedback on the results, and Gil Neiger and the referees for very helpful comments on the paper.

REFERENCES

1. Attiya, H., Bar-Noy, A., Dolev, D., Peleg, D., and Reischuk, R. Renaming in an asynchronous environment. *J. ACM 37*, 3 (1990), 524–548.
2. Attiya, H., and Djerassi-Shintel, T. Time Bounds for Decision Problems in the Presence of Timing Uncertainties and Failures. In *proceedings of the 7th International Workshop on Distributed Algorithms*, 1993, 204–218.
3. Attiya, H., Dwork, C., Lynch N. A., and Stockmeyer, L. J. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM 41*, 1 (1994), 122–152.
4. Bazzi, R. Automatically increasing fault tolerance in distributed systems, Ph.D. Thesis, College of Computing, Georgia Institute of Technology, Technical Report GIT-CC-94-62, December 1994.
5. Chaudhuri, S. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation 103*, 1 (1993), 132–158.
6. Chaudhuri, S., Herlihy, M., Lynch, N. A., and Tuttle, M. R. A tight lower bound for k-set agreement. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, 1993, 206–215.
7. Chaudhuri, S., Herlihy, M., and Tuttle, M. R. Wait-free implementations in message-passing systems. *Theoretical Computer Science 220*, 1 (1999), 211–245.
8. Cristian, F., Aghili, H., Strong, H. R., and Dolev, D. Atomic broadcast: from simple message diffusion to Byzantine agreement. *Information and Control 118*, 1 (1995), 158–179.
9. Djerassi-Shintel, T. Lower bounds for decision problems in semi-synchronous systems. M.Sc. Thesis (in Hebrew), Department of Computer Science, The Technion, October 1993.
10. Dolev, D., and Strong, H. R. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput. 12*, 4 (1983), 656–666.
11. Fredrickson, G. N., and Lynch, N. A. Electing a leader in a synchronous ring. *J. ACM 34*, 1 (1987), 89–115.
12. Graham, R. L, Rothschild, B. L., and Spencer, J. H. *Ramsey Theory*, John Wiley & Sons, New York, 1980.
13. Herlihy, M. P., and Tuttle, M. R. Wait-Free Computation in Message-Passing Systems. In *Proceedings 9th ACM Symposium on Principles of Distributed Computing*, 1990, 347–362. See also [7].
14. Herlihy, M. P., Rajsbaum, S., and Tuttle, M. R. Unifying synchronous and asynchronous message-passing models. In *Proceedings 17th ACM Symposium on Principles of Distributed Computing*, 1998, 133–142.
15. Neiger, G., and Toueg, S. Automatically increasing the fault-tolerance of distributed algorithms. *J. Algorithms 11*, 2 (1990), 374–419.
16. Pease, M., Shostak, R., and Lamport, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (1980), 228–234.
17. Ponzio, S. Consensus in the presence of timing uncertainty: Omission and Byzantine failures. In *Proceedings 10th ACM Symposium on Principles of Distributed Computing*, 1991, 125–138.
18. Ponzio, S. The real-time cost of timing uncertainty: Consensus and failure detection. S.M. Thesis, Report MIT/LCS/TR-518, Laboratory for Computer Science, MIT, October 1991.

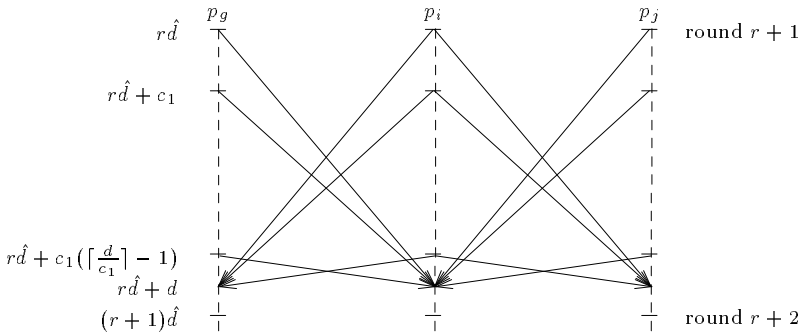

FIG. 1. One round of a fast, round-structured timed execution.

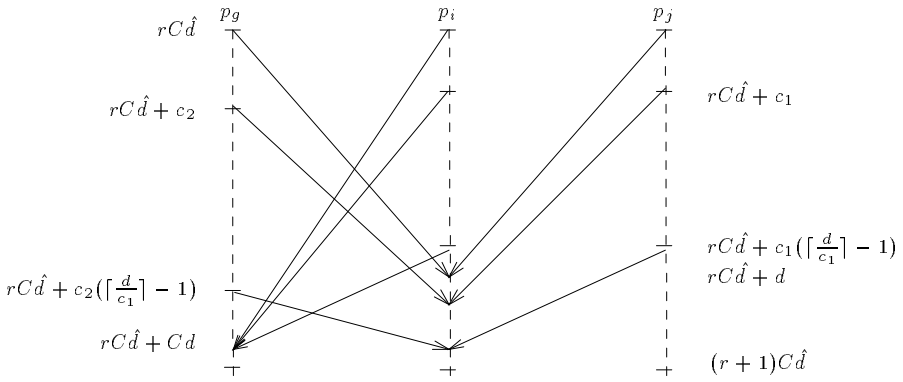

FIG. 2. Stretching a fast, round-structured timed execution.

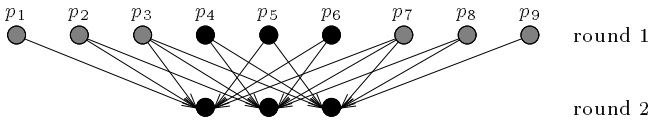

FIG. 3. The first round of the sandwich failure pattern for nine processes; nonfaulty processes are black.