# A Message-Passing and Adaptive Implementation of the Randomized Test-and-Set Object 

Emmanuelle Anceaume*, François Castella ${ }^{\dagger}$, Achour Mostéfaoui ${ }^{\ddagger}$ and Bruno Sericola ${ }^{\S}$<br>*IRISA / CNRS Rennes (France), emmanuelle.anceaume@irisa.fr<br>${ }^{\dagger}$ IRMAR / Université de Rennes 1 (France), francois.castella@univ-rennes1.fr<br>${ }^{\ddagger}$ LINA / Université de Nantes (France), achour.mostefaoui@univ-nantes.fr<br>${ }^{\S}$ INRIA Rennes - Bretagne Atlantique (France), bruno.sericola@inria.fr


#### Abstract

This paper presents a solution to the well-known Test\&Set operation in an asynchronous system prone to process crashes. Test\&Set is a synchronization operation that, when invoked by a set of processes, returns yes to a unique process and returns no to all the others. Recently many advances in implementing Test\&Set objects have been achieved, however all of them uniquely target the shared memory model. In this paper we propose an implementation of a Test\&Set object for a message passing distributed system. This implementation can be invoked by any number $p \leq n$ of processes where $n$ is the total number of processes in the system. It has an expected step complexity in $O(\log p)$ and an expected message complexity in $O(n p)$. The proposed Test\&Set object is built atop a new basic building block, called selector, that allows to select a winning group among two groups of processes.


Index Terms-Test\&Set, agreement problem, asynchronous message-passing system, crash failures, randomized algorithm, synchronization.

## I. Introduction

The Test\&Set problem is a classical synchronization service in shared-memory centralized systems classically provided by a unique hardware atomic instruction. It allows to solve competition problems. When invoked by a set of processes, it returns yes to a unique process (the winner) and returns no to all the others (the losers). According to the hierarchy of agreement problems based on consensus numbers given by Herlihy in [12], consensus ${ }^{1}$ is harder to solve than Test\&Set, that is a solution to Test\&Set does not allow to solve consensus, but Test\&Set is still too hard to be solved in a pure asynchronous system [5]. Indeed, Test\&Set has a consensus number equal to two, just like renaming and queues for example, whereas the consensus number of the consensus problem is infinite [12]. For instance, a simple way to solve the Test\&Set problem is to use a multivalued consensus service, such that each process proposes its identity to the consensus and decides a value. If the decided identity is its own identity it returns yes otherwise, it returns no.
a) Contributions: In this paper we propose a randomized and distributed message-passing implementation of the Test\&Set operation. Regarding randomization, Herlihy [12] has shown that Test\&Set does not have a deterministic implementation as soon as one crash may occur, and thus in order

[^0]to implement it in an asynchronous system, it is necessary to add synchrony assumptions or to use randomization. We focus on the latter option. For randomized solutions, the relations between random decisions and the scheduling of processes (i.e., read/write operations in the shared memory model, and send/receive operations in the message passing model) are taken into account through the definition of the adversary. In this paper we consider the oblivious adversary model, that is the model in which the adversary makes all its scheduling decisions at the beginning of the execution independently of the random values tossed by the processes in the course of the execution. In contrast, the adaptive adversary model supposes that the adversary makes its decisions based on the full history of the events. The adaptive adversary is stronger than the oblivious one and has led to space expensive known implementations [3] or non adaptive ones [1] in the shared memory model. Regarding communication, all the efficient solutions have, so far, been implemented with shared registered [1]-[3], [10] arguing that one can automatically transform shared-memory based algorithms to messagepassing ones [4]. Of course, this automatic transformation does not necessarily keep the efficiency property of the sharedmemory based algorithms. Table I proposes a summary of the important results of the Test\&Set object implemented in the shared memory model. The first column gives the step complexity. For shared memory systems, it represents the maximum number of steps (read/write) needed by any process in expectation to complete its execution. The step complexity, for message-passing systems, is the size of the longest causal sequence of messages that is needed for a process to complete its execution. A shared register can be implemented in a message-passing system [4]. A read/write operation on a shared register needs $\Omega(n)$ messages and a constant number of communication steps. The step complexity is thus the same for both systems. In Table I, the space complexity refers to the number of shared multi-writer/multireader atomic registers used in the implementation of the Test\&Set object. It is interesting to notice that all the best solutions [1]-[3], [10] implemented in the shared memory model require at least a number of atomic registers linear in the total number of processes $n$ in the system. Actually Giakkoupis and Woelfel [10] have proven that at least $\log n$ atomic registers are needed for any randomized register-based Test\&Set implementation. The message complexity column
represents the number of messages needed in expectation to complete the execution of all none crashed processes. Note that it is possible to give the message complexity for the sharedmemory solutions assuming the shared registers are implemented over a message-passing system [4]). To obtain this message complexity, we need the total number of read/write operations executed in expectation (the column "total step complexity") knowing that each read/write operation needs $O(n)$ messages.

In this work, and in contrast to the aforementioned solutions, we propose a message-passing implementation of the Test\&Set operation. Our implementation can be invoked by any number $p \leq n$ of processes. It has an expected step complexity in $O(\log p)$ and an expected message complexity in $O(n p)$ against an oblivious adversary (see the last line of the table). These complexities assume the scheduling of the worst adversary taken from the oblivious family. Having a step complexity that depends on $p$ and not on the number of processes $n$ of the system makes our solution adaptive. This makes our solution interesting from both the theoretical aspect but also from the practical one, as the cost of the implementation depends uniquely on the number of processes that concurrently invoke it. The implementation we propose of the Test\&Set object goes through a series of calls to a basic building block that we call in the following selector. A selector is a distributed service, invoked by a set of processes, that allows to select a winning group among at most two competing ones. We propose a message-passing implementation of the selector in presence of an oblivious adversary. The step complexity of the selector implementation is constant. A variant of the GroupElect object proposed by Woelfel and Giakouppis [10] would provide a shared memory implementation of the selector object in presence of an oblivious adversary.
b) Road map: In the remaining of the paper, Section II presents the underlying model and specifies the Test\&Set problem. Section III presents the selector object, proposes a randomized implementation of this object whose correctness is demonstrated, and derives its message complexity. Section IV presents a randomized implementation of the Test\&Set object, demonstrates the correctness of this implementation, and presents both its message and step complexity. Finally Section V concludes.

## II. Computation Model and Problem Definition

## A. Computation Model

We consider an asynchronous system consisting of a set $\Pi$ of $n$ processes, namely, $\Pi=\left\{p_{1}, p_{2}, \ldots, p_{n}\right\}$. A process can fail prematurely by crashing. A process behaves according to its specification until it (possibly) crashes. After it has crashed a process executes no step. A process that never crashes is said to be correct; otherwise it is faulty. Let $t$ denote the maximum number of processes that may crash. We assume that a majority of processes is correct, namely $t<n / 2$. We focus on a message-passing solution, that is processes communicate and synchronize by sending and receiving messages through reliable but not necessarily FIFO channels. As the system is asynchronous, there are no assumption regarding the relative
speed of processes nor the message transfer delays. The communication system offers two types of communication primitives. A point-to-point communication primitive send, and a broadcast primitive bcast that allows a process to send a same message to all the processes. This operation is not atomic, it can be implemented as a multi-send statement; if the sender of a message is faulty some processes can receive it and others not. Finally, we consider the oblivious adversary model, that is the model in which the adversary makes all its scheduling decisions at the beginning of the execution independently of the random values tossed by the processes in the course of the execution.

## B. The Randomized Test\&Set Problem

Test\&Set is usually a hardware operation offered by the processor. In the case of distributed computing, the Test\&Set problem is a coordination problem where a set of processes invoke Test\&Set and return a binary value yes or no such that exactly one returns yes (the winner) and all the others return no (the losers). From an operational point of view, the Test\&Set operation is attached to distributed objects. Let $o$ be a Test\&Set object that can be accessed through the method Test\&Set, which can be invoked by any process $p_{i}$ using o.Test\&Set(). An invocation returns a binary result yes or no. A protocol that solves the randomized Test\&Set problem must satisfy the following four properties:

- TS-Validity: A process, invoking the o.Test\&Set primitive, that returns a value must return either yes or no.
- TS-Obligation: If no process crashes then, exactly one process returns yes.
- TS-Agreement: At most one process returns yes and in this case, all the other returning processes return no.
- TS-Termination: An invocation by a correct process of the o.Test\&Set primitive terminates with probability 1.
Moreover, the different calls to the Test\&Set operations need to be linearizable. It has been proved in [11] that any object that satisfies the properties cited above can be used together to implement a linearizable Test\&Set object. Consequently, we will not worry about linearizability.


## III. A New Construction: the Selector Object

The key technical idea of our work relies on the selector, a new distributed structure that is used as a building block for the implementation of the randomized Test\&Set object. As will be shown in the sequel, the message complexity of the selector object invoked by $p$ processes requires $O(n p)$ messages, and has a constant step complexity, i.e., in average the number of round executed by each competing process is 2 . The following section presents this new construction.

## A. Specification of the Selector Object

The selector object proposes a unique access primitive play () , which is invoked with a Boolean parameter $g(g=0$ or $g=1$ ). Each of the two binary values represents a group, i.e., group 0 or 1 . A process randomly chooses its group 0 or 1 each time it invokes primitive play(). This basic object is

| Test\&Set Protocol | Step <br> complexity | Total step <br> complexity | Message <br> complexity | Space <br> complexity | Adversary | Adaptive |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | step | space |  |  |  |  |  |
| Afek et al. 1992 [1] | $O(\log n)$ | $O(n \log n)$ | $O\left(n^{2} \log n\right)$ | $O(n)$ registers | adaptive | no | no |
| Alistarh et al. 2010 [3] | $O(\log p)$ | $O\left(p \log ^{2} p\right)$ | $O\left(n p \log ^{2} p\right)$ | $\Theta\left(n^{3}\right)$ registers | adaptive | yes | no |
| Giakkoupis and Woelfel 2012 [10] | $O(\log p)$ | $O\left(p \log ^{2} p\right)$ | $O\left(n p \log ^{2} p\right)$ | $O(n)$ registers | adaptive | yes | no |
| Alistarh and Aspnes 2011 [2] | $O(\log \log n)$ | $O(n)$ | $O\left(n^{2}\right)$ | $\Theta\left(n^{3}\right)$ registers | oblivious | no | no |
| Giakkoupis and Woelfel 2012 [10] | $O\left(\log { }^{*} p\right)$ | $O\left(p \log ^{*} p\right)$ | $O\left(n p \log ^{*} p\right)$ | $O(n)$ registers | oblivious | yes | no |
| This paper | $O(\log p)$ | $O(p)$ | $O(n p) \operatorname{messages}^{-}$ | oblivious | yes | - |  |

Table I
Complexities of Test\&SEt objects in both shared memory and message-passing models

| Output | Meaning |
| :---: | :---: |
| (yes, yes) | $p_{i}$ 's group wins and $p_{i}$ is the winner in the group |
| (yes, no) | $p_{i}$ 's group wins and there is no winner in the group |
| ( $\mathrm{no}, \mathrm{no}$ ) | either the group of $p_{i}$ looses or there is a winner in $p_{i}$ 's group but the winner is not $p_{i}$ |
| (no,yes) | Impossible, $p_{i}$ cannot be a winner if its group does not win |

## Table II

The possible outputs of a selector object invoked by a set of $p \geq 1$ PROCESSES, $p_{i}$ BELONGS TO.
in charge of selecting the winning group $g^{\prime}$, and the winning process within this group, if any. Consequently the primitive play() returns two Boolean values to each invoking process. The first one says if the group of the invoking process is the winning group, and the second one indicates whether the invoking process is also the winner in the group. Table II shows the four possible responses process $p_{i}$ can receive upon invocation of primitive play.

More formally, let $s$ be a selector object, invoked by any process $p_{i}$ using $s . p l a y(g)$ with $g$ equal to 0 or 1 . A protocol that implements such an object must satisfy the following five properties.

- S-Validity: If a process invokes the $s$.play primitive and returns then, it returns either (yes,yes), (yes, no) or (no, no).
- S-Obligation-solo: If a correct process invokes s.play alone (solo execution) then, it returns (yes, yes).
- S-Obligation: If no process crashes then, at least one process returns (yes,-).
- S-Agreement: At most one process returns (yes, yes), and in this case, all the other returning processes return (no,no).
- S-Exclusion: If an invocation of $s$.play with parameter $g$ returns (yes,-) then, no invocation with parameter $\neg g$ can return (yes, - ).
- S-Termination: An invocation of s.play by a correct process terminates with probability 1.


## B. A Message-Passing Implementation of the Selector Object

The implementation of the selector object falls under the impossibility result of many agreement problems in the context of asynchronous distributed systems prone to process failures [9]. We thus consider an asynchronous message-passing
distributed system augmented with a random oracle to circumvent the impossibility result. Specifically, processes have access to a function common_coin(), which provides to all the invoking processes the same value ( 0 or 1 ) with probability $1 / 2$. Each process invokes function common_coin () at the beginning of each round of the protocol, and thus all the processes get the same value. Our solution is an adaptation of Ben-Or consensus algorithm [6] and can be seen as a variant of the commit/abort mechanism introduced in the messagepassing model in [13] coupled with a random generator to circumvent the impossibility result. The same approach has also been followed in the Test\&Set algorithm by Tromp and Vitany [15] in the context of shared memory systems, except that their algorithm can be invoked by at most two processes. Woelfel and Giakouppis [10] propose the GroupElect object in the context of shared-memory. Similarly to our solution, the GroupElect object supposes an oblivious adversary and uses random numbers, however, and in contrast to our solution, a process can never know whether it is the only winner of the election or not.

Operationally, the selector is attached to distributed objects. Let us consider a selector $s$. As previously described, selector $s$ can be concurrently invoked by $p$ processes, $1 \leq p \leq n$, however all the $n$ processes of the system have to participate. Indeed, as there is no shared memory and processes may fail by crashing, the participation of all processes is required to serve as arbiters and as collective memory [5].

The algorithm is round based. It is presented in Figure 1. It goes through a series of rounds each one composed of two communication phases. The algorithm is divided into two parts. The first one is executed by the invoking processes (that is the processes that have invoked method play on object $s$ ), while the second part (called Relay Task in Figure 1) is executed by all processes including the invoking ones. This is done for generality since the messages sent by a process to itself are directly delivered to it. The Relay Task serves as a relay to the messages sent by the competing processes and implements some kind of collective memory [5]. We will respectively call these two groups of processes the invoking processes and the relaying processes.

The goal of the method play is to determine the winning value among the proposed ones and the winner of the competition, if any. A process $p_{i}$ wins the competition if either $p_{i}$ is the only process that invoked the primitive play or $p_{i}$ has invoked the primitive play with the winning boolean value
$g^{\prime}$ and $p_{i}$ has no evidence that another process did the same. If none of both conditions hold, then all the processes that have proposed the winning value will compete in a new execution of primitive play, while all the other ones stop the competition. This is achieved as follows. Each invoking process $p_{i}$ handles a variable $g_{-} e s t_{i}$ representing its estimation of the winning group ( 0 or 1 ). Variable $g_{-} e s t_{i}$ is initially set to the value $g_{i}$ that $p_{i}$ proposed when it invoked the method play. Then, this estimate will evolve according to what $p_{i}$ will learn during the protocol. Similarly each invoking process $p_{i}$ manages a variable $i d \_^{\text {est }}{ }_{i}$ representing its estimation of the possible winning process inside the winning group (initially id_est $_{i}$ is set to $p_{i}$ ).

At the beginning of each round, each invoking process $p_{i}$ tosses a common coin $c$. During the first phase of the current round, $p_{i}$ broadcasts its estimates $g_{-} e s t_{i}$ and $i d \_e s t i_{i}$ in a PHASE message to all processes, and waits for their echo (Line 5 in Figure 1). As several processes may play during a same round, some relaying processes may first receive the PHASE message from some invoking process $p_{i}$ and thus will only echo $p_{i}$ estimate while other relaying processes may first receive a PHASE message from another invoking process $p_{j}$ possibly endorsing the group $\neg g$ and will echo $p_{j}$ estimate. Each relaying process manages two variables $g[r, x], i d[r, x]$ for each of the two phases $x$ of each round $r$. They are used to store the estimates ( $g$ and $i d$ ) received in the PHASE message received from an invoking process. Thereafter, these same estimates are echoed to all invoking processes from which a PHASE message was received for the same phase of the same round.

Each invoking process $p_{i}$ collects in set $G_{i}$ the echoed values received from a majority of processes including itself. Upon receipt of a majority of echoes, if $G_{i}$ contains a single value then $p_{i}$ keeps this value in $g_{-} a u x_{i}$ otherwise, it knows that there is contention between two groups of processes each one championing for the two possible groups ( 0 and 1 ). Process $p_{i}$ sets variable $g_{-} a u x_{i}$ to a value $\perp$ reflecting such a contention. Process $p_{i}$ applies the same argument for the echoed identifiers.

To summarize, the first phase ensures that for any pair of invoking processes $p_{i}$ and $p_{j}$, if $g_{-} a u x_{i}$ and $g_{-} a u x_{j}$ are both different from $\perp$ then they necessarily contain the same value $g$ and if $i d \_a u x_{i}$ and $i d \_a u x_{j}$ are both different from $\perp$ then they necessarily contain the same process identity $i d$.

During the second phase of the round, $p_{i}$ broadcasts both $g_{-} a u x_{i}$ and $i d \_a u x_{i}$ and collects in $G_{i}$ and $I d_{i}$ the echoes from a majority of processes. By construction of the first phase, if $G_{i}$ contains a value $g$ and possibly $\perp$ then $p_{i}$ is sure that any other invoking process $p_{j}$ will receive either $g$ or $\perp$ values. Moreover, if $G_{i}$ contains a unique value, $p_{i}$ is certain that any other invoking process $p_{j}$ will receive at least this value (two majorities always intersect). In particular, if $G_{i}$ contains only the $\perp$ value, $p_{i}$ knows that no winning values has been exhibited during the round, thus $p_{i}$ triggers a new round by setting its estimate to the random value $c$ picked at the beginning of the round, and $i d \_^{2} e s t_{i}$ to $\perp$ (Line 15). Now, if $G_{i}$ only contains a non bottom value $g$ then $g$ is the winning value of the round. Process $p_{i}$ must then determine whether

```
Function s.play \(\left(g_{i}\right)\)
    (1) \(r_{i} \leftarrow 0 ;\) g_est \(_{i} \leftarrow g_{i} ;\) id_est \(_{i} \leftarrow p_{i}\);
    (2) while true do // Sequence of rounds //
    (3) \(\quad r_{i} \leftarrow r_{i}+1 ; c \leftarrow\) common_coin();
        —— Phase 1 of round \(r_{i}\)
    bcast \(\operatorname{PHASE}\left(r_{i}, 1, g_{-} e s t_{i}, i d \_e s t_{i}\right)\);
(5) wait until (PHASE \(\left(r_{i}, 1, g_{-} e s t, i d_{-} e s t\right)\) messages were received
    from a majority of processes);
    (6) let \(G_{i}\) be the set of \(g_{-}\)est values received at Line 5;
    (7) let \(I d_{i}\) be the set of id_est values received at Line 5;
    (8) if \(\left(G_{i}=\{g\}\right.\) then \(g_{-} a u x_{i} \leftarrow g\) else \(g_{-} a u x_{i} \leftarrow \perp\);
    (9) \(\quad\) if \(\left(I d_{i}=\{i d\}\right.\) then \(i d \_a u x_{i} \leftarrow i d\) else \(i d \_a u x_{i} \leftarrow \perp\);
    —Phase 2 of round \(r_{i}\)
    \(b c a s t \operatorname{PHASE}\left(r_{i}, 2, g_{-} a u x_{i}, i d_{-} a u x_{i}\right)\);
(11) wait until (PHASE \(\left(r_{i}, 2, g_{-} a u x, i d_{-} a u x\right)\) messages were received
    from a majority of processes);
(12) let \(G_{i}\) be the set of \(g_{-} a u x\) values received at Line 11;
(13) let \(I d_{i}\) be the set of \(\bar{d}_{-} a u x\) values received at Line 11;
(14) case
(15) \(\quad\left(G_{i}=\{\perp\}\right): g_{-} e s t_{i} \leftarrow c\); id_est \(_{i} \leftarrow \perp\);
(16) \(\quad\left(G_{i}=\{g\}\right) \wedge\left(I d_{i}=\{i d\}\right)\) : if \(\left(i d=p_{i}\right)\) then
(17) return (yes, yes);
(18) \(\left(G_{i}=\{g\}\right) \wedge\left(I d_{i}=\{\perp\}\right):\) if \(\left(g=g_{i}\right)\) then
    19) return (yes, no)
    (20) else return (no, no);
(21) \(\left(G_{i}=\{g\}\right) \wedge\left(I d_{i}=\{i d, \perp\}\right):\) if \(\left(i d=p_{i}\right)\) then
(22) \(\quad\left(G_{i}=\{g\}\right) \wedge\left(I d_{i}=\{i d, \perp\}\right):\) if \(\left(i d=p_{i}\right)\) then
\(g_{-}\)est \({ }_{i} \leftarrow g ;\) id_est \(_{i} \leftarrow \perp ;\)
(23) else return (no, no);
(24) \(\left(G_{i}=\{g, \perp\}\right) \wedge\left(I d_{i}=\{i d, \perp\}\right)\) : if \(\left(i d=p_{i}\right)\) then
(25) \(\quad g_{-}\)est \(i \leftarrow g\); id_est \({ }_{i} \leftarrow \perp\);
(26) else return (no, no);
(27) \(\left(G_{i}=\{g, \perp\}\right) \wedge\left(I d_{i}=\{\perp\}\right):\) g_est \(_{i} \leftarrow \perp\); id_est \({ }_{i} \leftarrow \perp\);
(28) endcase;
(29)endwhile
Task s.relay // Launched by any process \(\overline{p_{i}, 1 \leq i \leq n . p_{i} \text { maintains }}\)
        four variables \(g[r, 1], g[r, 2], i d[r, 1]\) and \(i d[r, 2]\) per round \(r\),
        Initially initialized to \(\perp / /\)
(30)upon receipt of \(\operatorname{PHASE}(r, x, g, i d)\) message from \(p_{j}\)
(31) if \((g[r, x]=\perp\) and \(i d[r, x]=\perp)\) then
(32) \(g[r, x] \leftarrow g ; i d[r, x] \leftarrow i d\);
(33) send \(\operatorname{PHASE}(r, x, g[r, x], i d[r, x])\) to \(p_{j}\);
```

Figure 1. A randomized protocol implementing the selector object run by process $p_{i}(t<n / 2)$
the echoed values it has received reflect a contention among the potential winners or not. Such a contention exists if $I d_{i}$ contains at least the bottom value. If $p_{i}$ does not observe such a contention and if $p_{i}$ is actually the winner of the competition (i.e., $I d_{i}=\left\{p_{i}\right\}$ ) then it successfully leaves the competition by returning (yes, yes), see Line 17. Meanwhile, for any other process $p_{j}$, if $p_{j}$ suspects that $p_{i}$ may have won the competition (Lines 21 and 24) then $p_{j}$ abandons the competition by returning (no, no) (Lines 23 and 26). Now, if $p_{i}$ observes a contention among the potential winners but there is no hint of the potential winner, i.e., $I d_{i}=\{\perp\}$ (Line 18), then if $p_{i}$ is among the processes that initially proposed $g$ it starts a new competition by returning (yes, no), see Line 19. It abandons the competition otherwise, see Line 20 . If, on the other hand, $p_{i}$ knows that a majority of processes have seen its estimate in the first Phase ( $i d=p_{i}$ at Line 24) but not necessarily in the second Phase $\left(\perp \in I d_{i}\right)$, then $p_{i}$ triggers a new round by specifying that there is a winning group value, but there is no hint on the potential winner. This will allow all the processes involved in this new round to return (yes, no). The last possible scenario occurs when $p_{i}$ sees a contention on the group value (i.e., $\perp \in G_{i}$ ) but one group $g$ has nevertheless
been seen by a majority of processes (Line 24 and 27). If $p_{i}$ knows that a majority of processes have seen its estimate in the first Phase ( $i d=p_{i}$ at Line 24) but not necessarily in the second Phase, then it triggers a new round by specifying that there is a winning group value, but there is no hint on the potential winner. This will allow all the processes involved in the new round to return (yes, no). On the other hand, in Line 27, there is no hint on the potential winning process thus $p_{i}$ triggers a new round with $g_{-} e s t_{i}$ and $i_{-} e s t_{i}$ both equal to $\perp$. Finally, it is easy to see that if there is a unique invoking $p_{i}$ during some round $r, p_{i}$ will return (yes, yes) at line 17 of round $r$ as $p_{i}$ can only received echoes from its own value.

## C. Correctness of the Selector Implementation

In this section, we show that the randomized implementation of the selector object presented in Figure 1 is correct, that is guarantees the properties given in Section III-A. We start by showing the following lemmata prior to proving the properties of the Selector implementation. Let $s$ be a selector object.

Lemma 1 (Non-blocking): No correct process blocks forever in a round.

Proof: Let us first note that no relaying process can block forever at line 31 and will respond to any message sent by any invoking process. By assumption, there is a majority of correct processes. Thus any invoking process that broadcasts a message at lines 4 or 10 will receive at least a majority of associated echo (i.e., PHASE messages). Consequently, no invoking process can remain blocked forever at lines 5 or 11.

Lemma 2: If all the invoking processes start a round $r$ with the same estimate $g$ then, all the invoking processes that do not crash return either in round $r$ or in round $r+1$.

Proof: Let $g$ be the estimate proposed by all invoking processes at the beginning of round $r$. The invoking processes will broadcast the same value $g$ at line 4, and thus will get only value $g$ in their buffer $G$. Consequently, each invoking process $p_{i}$ executes line 8 by affecting $g_{-} a u x_{i}$ to $g$ and will receive (a majority of) PHASE messages with $g_{-} a u x=g$. Thus each of the three cases at Lines 16, 18, and 21 need to be considered. Let us examine the two former ones: $p_{i}$ returns (yes, yes) if it is the only invoking process seen by a majority of processes and, returns (yes, no) if the contention between the invoking processes has been detected (that is not all the relaying processes have received the same estimations). Now consider the case at Line 21. If $p_{i}$ is not the invoking process seen by a majority of relays during the first phase of the current round, then $p_{i}$ returns (no, no), otherwise $p_{i}$ triggers round $r+1$ with $g_{-}$est ${ }_{i}=g$ and $i d \_e s t=\perp$, and will exclusively execute Line 18. Consequently, $p_{i}$ will return (yes, no) in Phase 2 (of round $r+1$ ).

Lemma 3 (S-Validity): A process, invoking the s.play primitive, that returns a value, must return (yes,yes), (yes, no) or (no,no).

Proof: Straightforward from Lines 16, 18, 23, and 26.
Lemma 4 (S-Obligation-solo): If a process invokes the s.play primitive alone and does not crash then, it returns (yes,yes).

Proof: If an invoking process $p_{i}$ executes alone a given round then necessarily the echoes it will receive at lines 5 and 11 contain a single value $g$ and its identifier $p_{i}$. Consequently, $G_{i}$ will always contains a single non- $\perp$ value leading process $p_{i}$ to decide (yes,yes).

Lemma 5 (S-Agreement): At most one process returns (yes, yes), and in this case, all the other returning processes return (no, no).

Proof: Let $p_{i}$ be the first process that returns (yes, yes) at round $r$. By construction of the algorithm, this can only occur at Line 17 , that is $G_{i}=\{g\}$ and $I d_{i}=\{i d\}$, with $i d=p_{i}$. Thus, by the majority argument, for any other process $p_{k}$, variables $G_{k}$ and $I d_{k}$ must respectively contain at least $g$ and $i d$ at round $r$. Consequently, process $p_{k}$ necessarily executes one of the two cases at Lines 21 and 24, and in both cases $p_{k}$ returns (no, no) at round $r$.

Lemma 6 (S-Exclusion): If an invocation of $s$.play with parameter $g$ returns (yes,-) then, no invocation with parameter $\neg g$ can return (yes,-).

Proof: Let $r$ be the smallest round at which some invoking process $p_{i}$ returns (yes, -). By construction it can only happens at Line 17 or 19. If $p_{i}$ returns (yes, yes) (Line 17) then by Lemma 5 all the other processes, that return a value, return (no, no). Now, suppose that $p_{i}$ returns (yes, no) at Line 19 , and $g_{i}=g$. By construction, this means that for any other processes $p_{k}, G_{k}$ must contain at least $g$ at round $r$ (two majority always intersect). Thus, if $p_{k}$ returns (yes, no) then necessarily $g_{k}=g$ and thus the lemma holds. Now, if $p_{k}$ does not abandon the execution, then $p_{k}$ triggers round $r+1$ with $g_{-} e s t_{k}=g$. This applies for all processes executing round $r+1$. By Lemma 2, for all these processes that return a value, they return, at round $r+1$, (yes, - ) only if $g_{k}=g$.

Lemma 7 (S-Obligation): If no process crashes then, at least one process returns (yes,-).

Proof: By contradiction. Suppose that no correct process returns (yes,-). By Lemma 1, no correct process blocks forever in a round. By Lemma 4, at least two processes must invoke play otherwise the solo execution returns (yes, yes). Let $p_{i}$ and $p_{k}$ be any two of these processes such that $p_{i}$ invokes play $(g)$ and $p_{k}$ invokes play $\left(g^{\prime}\right)$ with $g, g^{\prime} \in\{0,1\}$. Two cases need to be considered.

1) Suppose first that for all the invoking processes, we have $g=g^{\prime}$. Thus all the processes execute either Line 16, Line 18, or Line 21 in Phase 2 of the protocol. By assumption, all the processes have initially proposed the same value $g$. Thus, none of them can return ( $\mathrm{no}, \mathrm{no}$ ) at line 20. Now, not all of them can return (no, no) at line 23 because the process whose id matches id should trigger round 2, in which case it would execute Line 19 and return (yes, no). In all the other cases, processes return (yes, no) or (yes,yes) in round 1, which also contradicts the assumption of the lemma.
2) Suppose now that $g=\neg g^{\prime}$.
a) If some process $p_{k}$ returns ( $\mathrm{no}, \mathrm{no}$ ) at Line 20, then its initial value $g_{k}=g^{\prime}$ is not the potential winning value $g$ and none of the other processes can execute line 15 (by the majority argument, $g$ must belong to all the $G_{j} \mathrm{~s}$ ). Thus all the processes
that have invoked play with $g^{\prime}$ return (no, no). Now, all the processes that have invoked play with $g$ (by construction there must be at least one such process) cannot all return (no, no) at Lines 23 or 26 because the process whose id matches id must necessarily have executed any of the cases (except the one at Line 15 by assumption of the case). Thus such a process must have either returned (yes, -) in this round or triggered round 2 with $g_{-} e s t_{i}=g$ and $i d \_e s t_{i}=\perp$, and thus shall return (yes, no) in round 2 . This contradicts the assumption of the lemma.
b) If some process $p_{k}$ returns ( $\mathrm{n} \circ$, no) at Line 23, then a similar argument as above applies.
c) If some process $p_{k}$ returns ( $\mathrm{no}, \mathrm{no}$ ) at Lines 26, then a similar argument as the above one also applies, although the case at Line 15 may also hold. Thus, as previously, all the processes that have invoked play with $g$ and that trigger round 2 do it with either $g_{-} e s t=g$ and $i d \_e s t=\perp$, or $g \_e s t=\neg g$ and $i d \_e s t=\perp$ if $c=\neg g$. Suppose that at least two processes $p_{i}$ and $p_{j}$ execute round 2 and one does it with $g_{-}$est $=g$ and $i d \_e s t=\perp$ and the other one does it with $g_{-} e s t=\neg g$ and id_est $=\perp$ (the case where a single process executes round 2 trivially returns (yes, no)). By construction, only cases at Line 15, 18 and 27 may hold. If all these processes execute Line 15 then they all trigger a new round with the same estimate during which they will return (yes, no). Suppose now, that at Lines 18 and 27, the potential winning value is $\neg g$ (this may happen because some process has triggered round 2 with $\neg g$ ). Note however that all the processes involved in round 2 have initially invoked play with $g$. Thus all the processes that run Line 18 will return (no, no), and those that execute Line 27 will trigger a new round with g_est $=\perp$, and will finally return in Line 18 with (yes, no). All the scenario contradict the assumption of the Lemma, which completes its proof.

Lemma 8 (S-Termination): An invocation of s.play by a correct process terminates with probability 1.

Proof: Suppose by contradiction that some correct process $p_{i}$ does not terminate. It must be the case that either $p_{i}$ blocks forever in an execution or $p_{i}$ never stops from triggering new rounds. By Lemma 1, $p_{i}$ cannot block forever. Now, by Lemma 2, if all the competing processes in a given execution start a round $r$ with the same estimate $g$, all the invoking processes that do not crash return either in round $r$ or in round $r+1$. By the proof of Lemma 5, if some process wins the competition at round $r$, that is returns (yes, yes), then all the other processes stop the execution, by returning ( $\mathrm{no}, \mathrm{no}$ ), at round $r$. By Lemma 7 if all processes are correct then at least one returns (yes,-). Thus it must be the case that $p_{i}$ and possibly some other processes end Phase 2 of some round
$r$ by either executing Lines $15,21,24$ or 27 . Once again, by Lemma 2, if Line 15 is executed and $c=g$ then all processes return either in round $r$ or in round $r+1$. Thus let $p^{\prime}$ be the number of processes that trigger round $r+1$, such that some of them propose $g$ and the other ones propose $\neg g$. In this last case, there is a probability $p=1 / 2$ that the value kept by the process that executes Line 3 is equal to $g$. So, there is a probability $p_{\ell} \geq 1-1 / 2^{\ell}$ that all none crashed processes have the same estimate after at most $\ell$ rounds. As $\lim _{\ell \rightarrow \infty} p_{\ell}=1$, it follows that, with probability 1 , both invoking processes will start a round with the same estimate. Then, according to Lemma 2, they will return.

## D. Complexity Analysis of the Selector Object Implementation

Theorem 1 (Message complexity of play ()): The total number of messages exchanged by the randomized implementation of the selector object when concurrently invoked by $p$ processes is $O(n p)$.

Proof: As explained above, if there is a unique process that invokes the selector, it will return within a unique round. The number of messages needed is at most $2(n+1)(n+1$ messages for each phase). If $p$ processes invoke the selector, they will go through a constant number of rounds as it is the case for randomized consensus [7], [8]. During a phase, each invoking process broadcasts a message and each process responds once to each of the invoking processes. Thus, during one phase a maximum of $p(n+1)$ messages are exchanged. As there are two phases per round and the total number of rounds is constant, the message complexity is $O(n p)$.

## IV. A Message-Passing Adaptive Implementation of the Randomized Test\&Set Object

We now present the implementation of the Test\&Set object. Recall that it can be invoked by any number $p \leq n$ of competing processes. As aforementioned, implementation of the Test\&Set object relies on instances of the selector object as illustrated in Figure 2(a). Correctness of the implementation is presented in Section IV-B, and its complexity is derived in Section IV-C.

## A. The Randomized Test\&Set Algorithm

Pseudo-code of the Test\&Set algorithm, given in Figure 2(b), can be seen as a process elimination by dichotomy. At the first step, each of the $p$ competing processes $p_{i}$ flips a local coin (we suppose that each process uses an unbiased coin) and invokes the first instance of the selector object with this coin as parameter. This parameter represents $p_{i}$ group for this instance of the selector object. The selector object selects the winning group (set of processes) allowed to continue the competition, and eliminates all the processes of the other group (if any). Specifically, any process $p_{i}$ that exits with (yes, no) from the current instance of the selector object triggers a new step of the Test\&Set algorithm by invoking the next instance of the selector object by flipping again a local coin. On the other hand, any process $p_{i}$ that exits from the current instance of the selector object with (no, no) also exits from the Test\&Set

(a) Randomized Test\&Set object using selector objects as building blocks

```
Function o.Test\&Set()
(1) step \(_{i}=1\);
(2) repeat forever
3) \(\quad\left(b_{1}, b_{2}\right) \leftarrow \operatorname{selector}\left[\right.\) step \(\left._{i}\right]\).play \((\mathbf{r n d}(0,1))\);
(4) if \(\left(b_{1}=\right.\) yes \(\wedge b_{2}=\) yes \() \vee\left(b_{1}=\right.\) no \()\) then
(5) \(\quad \operatorname{return}\left(b_{1}=\right.\) yes \(\wedge b_{2}=\) yes \()\)
(6) endif;
(7) \(\quad\) step \(_{i} \leftarrow\) step \(_{i}+1\);
(8) endrepeat;
```

(b) Randomized Test\&Set Object Algorithm

Figure 2. Randomized Test\&Set Object
invocation with no. The last step of the Test\&Set algorithm occurs when one of the remaining competing processes $p_{i}$ exits from the selector object invocation with (yes,yes). This winning process exits with yes from the Test\&Set invocation. As it will be proven in the sequel, any invocation by a process that does not crash terminates with probability 1 . As said above, our algorithm does not need to know how many processes access the Test\&Set object. Remaining of the paper will clarify all these points.

## B. Correctness of the Test\&Set Implementation

This section proves the correction of the algorithm of Figure 2(b) by proving the four properties of a Test\&Set object, namely the TS-Validity, TS-Obligation, TS-Agreement and TS-Termination properties and then shows the complexity both in terms of steps and messages of the algorithm.

The TS-Validity property is a direct consequence of line 4 of the algorithm, while the TS-Obligation property is a consequence of the S-Obligation of the selector underlying object. Indeed, if none of the processes that execute o.Test\&Set () crash, then necessarily they execute line 3 of the algorithm. By the S-Obligation property of the selector, at least some process will exit with (yes, -). If only (yes, no) is returned, then all these processes will execute a new instance of the selector until possibly exactly one process execute a solo execution in which case it will return (yes,yes). To prove the TSAgreement property, let us consider the first process that exits with yes at line 5 at some step step ${ }_{i}$. Necessarily this process invoked selector $\left[\right.$ step $\left._{i}\right]$.play () and this invocation returned (yes, yes). Consequently, by the S-Agreement property of a selector, all the other processes that invoke the selector will exit with (no,no) and consequently, these processes will return no at line 5 at the same step $s t e p_{i}$. Property TSTermination is more tricky to prove. By the S-Validity of a selector, returned values are pairs of boolean, consequently, the Test\&Set algorithm is properly executed (no type errors). Moreover, by the S-Termination property of a selector, a correct process terminates the call of line 3 with probability

1. Saying this, we conclude that if the Test\&Set algorithm does not terminate, it will execute an infinity of times the repeat loop. Section IV-C proves that this loop terminates after no more than $2 \log _{2}(p)$ invocations of the selector object in expectation for large values of the contention $p$ of the Test\&Set execution (see Theorem 2). Moreover, the average number of selector invocations done by any of the competing processes during a Test\&Set execution is constant ( 2 invocations per process as shown in Corollary 1).

## C. Complexity Analysis of the Test\&Set Implementation

We now analyze the complexity of our implementation with respect to both the number of execution steps and the number of exchanged messages.

To carry out the step complexity analysis, we consider the worst-case execution, namely that the Test\&Set protocol terminates at the latest when there is only one process executing the protocol. Indeed, the protocol may terminate before, that is, as soon as a process succeeds in being the winner of the winning group. However the analysis supposes the worst case execution, where the Test\&Set protocol executes until there is a unique competing process. Consequently, for the purpose of this worst case analysis, only the first boolean returned by the selector object is relevant. Recall that this boolean indicates whether the invoking process belongs to the winning group or not.

When competing processes invoke a selector instance, each one chooses a group at random (line 3 of Figure 2(b)). By the S-Exclusion property of a selector, only one group will win. The identity of the winning group ( 0 or 1 ) depends on the actual scheduling and the adversary. Hence, as the choice of the group is done at random, we assume that the two events "group 0 wins" and "group 1 wins" occur with the same probability $1 / 2$ and that the behaviors of the processes at each instant are independent of each other.

We suppose that $p \leq n$ processes concurrently access the Test\&Set object. The behavior of the algorithm can be modeled by a Markov chain $X=\left\{X_{\ell}, \ell \geq 1\right\}$, where $X_{\ell}$ represents the number of processes in competition at the $\ell$-th transition, i.e., the number of processes that execute the $\ell$-th step. Hence, the state of the Markov chain is an integer value $i(1 \leq i \leq p)$. The initial state of $X$ is state $p$, with probability 1 , that is $\mathbb{P}\left\{X_{0}=p\right\}=1$ and we denote by $P$ the transition probability matrix of $X$. The probability $P_{i, j}$ to go from state $i$ to state $j$ in one transition is equal to 0 if $i<j$. Indeed, a process that returns $b_{1}=$ no cannot any more continue the competition (see line 5 in Figure 2(b)). Now, when all the $i$ competing processes choose the same group (either 0 or 1 ) then they all restart the competition in the same state. It follows that, for $i=1, \ldots, p$,

$$
P_{i, i}=\frac{1}{2^{i}}+\frac{1}{2^{i}}=\frac{1}{2^{i-1}}
$$

Finally, for $1 \leq j<i \leq p, P_{i, j}$ is the probability that exactly $j$ processes among $i$ choose the same group and that this group wins. We thus have, in this case,

$$
P_{i, j}=\frac{1}{2}\left[\frac{1}{2^{i}}\binom{i}{j}+\frac{1}{2^{i}}\binom{i}{i-j}\right]=\frac{1}{2^{i}}\binom{i}{j} .
$$

The states $2,3 \ldots, p$ are thus transient states and state 1 is absorbing since $P_{1,1}=1$.

In the following we evaluate the average number of steps to reach state 1 , and the average total contention before termination, i.e., before reaching state 1 . By total contention we mean the following: Let us consider the sequence $n_{1}, n_{2}, \ldots, n_{p-1}$ where $n_{\ell}$ represents the number of processes that execute step $\ell$ of the Test\&Set protocol (contention on step $\ell$ ). By assumption $n_{1}=p$. We call the total contention on the whole selector objects the sum $n_{1}+n_{2}+\cdots+n_{p-1}$. We show that the average total contention is linear in $p$.

When $p$ processes are initially competing, the worst case time needed by the Test\&Set protocol to terminate is the hitting time of state 1 by Markov chain $X$. If we denote by $T_{p}$ this time, we have

$$
T_{p}=\inf \left\{\ell \geq 0 \mid X_{\ell}=1\right\}
$$

It is well-known, see for instance [14], that the expected value of $T_{p}$ is given by

$$
\mathbb{E}\left\{T_{p}\right\}=\alpha(I-Q)^{-1} \mathbb{1}
$$

where $Q$ is the matrix of dimension $p-1$ obtained from $P$ by deleting the row and the column corresponding to absorbing state $1, \alpha$ is the row vector containing the initial probabilities of the transient states, that is $\alpha_{p}=1$ and $\alpha_{i}=0$ for $i=2, \ldots, p-1$, and $\mathbb{1}$ is the column vector of dimension $p-1$ with all its entries equal to 1 . The expected value $\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}$ can also be evaluated using the well-known recurrence relation, see for instance [14],

$$
\begin{equation*}
\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}=1+\sum_{k=2}^{p} P_{p, k} \mathbb{E}\left\{T_{k} \mid X_{0}=k\right\} \tag{1}
\end{equation*}
$$

## Theorem 2 (Step Complexity of Test\&Set()): The

 expected time $\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}$ needed to terminate the Test\&Set protocol when $p$ processors are initially competing satisfies$$
\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}=O(\log (p))
$$

More precisely, there exists an integer $p_{0}>0$ such that, for all $p \geq p_{0}$, we have

$$
\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\} \leq 2 \log (p)
$$

where $\log$ denotes the logarithm function to the base 2 .
Proof: Sketch of the proof. For space reasons the full proof appears in the Appendix Introducing the notation $u_{p}=$ $\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}$ and replacing $P_{p, k}$ by its value, Formula 1 can be written as

$$
u_{p}=1+\sum_{k=2}^{p-1} 2^{-p}\binom{p}{k} u_{k}+O\left(2^{-p}\right)
$$

The key idea lies in the fact that $\binom{p}{k}$ is maximal when $k=p / 2$, and decreases rapidly away from the value $k=p / 2$, so that the above recursion formula for $u_{p}$ very roughly asserts that $u_{p} \approx 1+u_{p / 2}$. Would this simplified recursion formula hold true exactly, the bound $u_{p}=O(\log (p))$ would be obvious. Based on this rough idea, the proof is split into three main steps.

First, given a small $\alpha>0$, Stirling formula implies $2^{-p}\binom{p}{k}=$ $O\left(\exp \left(-2 p^{2 \alpha}\right)\right)$ uniformly in $k$ whenever $|k-p / 2| \geq p^{1 / 2+\alpha}$. This provides the simplified recursion

$$
u_{p}=1+\sum_{k:|k-p / 2| \leq p^{1 / 2+\alpha}} 2^{-p}\binom{p}{k} u_{k}+O\left(2^{-2 p^{\alpha}}\right)
$$

The second step consists in introducing a dyadic partition, so we define $U_{j}=\max _{2 \leq k \leq 2^{j}} u_{k}$. A detailed analysis of the above recursion formula provides,

$$
U_{j+1} \leq 1+\frac{U_{j}+U_{j+1}}{2}+O\left(2^{-2 p^{\alpha}}\right)
$$

where $p=2^{j}$. The last argument consists in proving that the above bound provides $U_{j} \leq 2 j+C$, for some constant $C$ that does not depend on $j$. This completes the proof.

Using this result and the Markov inequality, we obtain, for the positive integer $p_{0}$ of Theorem 2, for every $m \geq 1$ and $p \geq p_{0}, \mathbb{P}\left\{T_{p}>2 m \log (p)\right\} \leq 1 / m$.

We consider now the total contention before termination. For $\ell \geq 0$, we denote by $W_{\ell}(p)$ the number of processes that executed step $\ell$ of the protocol when $p$ processes are initially competing. This random variable is defined by $W_{\ell}(p)=\sum_{i=2}^{p} i 1_{\left\{X_{\ell}=i\right\}}$. Since the initial state is state $p$, we have $W_{0}(p)=p$ with probability $1 . W_{0}(p)$ represents the contention of the Test\&Set and also the contention of the first invocation of the selector object. The total contention before termination is denoted by $N(p)$ and given by $N(p)=$ $\sum_{\ell=0}^{\infty} W_{\ell}(p)$. Note that $N(p)$ is also the total contention of the whole invocations of the selector object. The next theorem gives the expectation of $N(p)$.

Theorem 3 (Total Contention): For every $p \geq 2$ and $\ell \geq$ 0 , we have

$$
\mathbb{E}\left\{W_{\ell}(p)\right\}=p / 2^{\ell} \text { and } \mathbb{E}\{N(p)\}=2 p
$$

Proof: Since $X_{0}=p$, we have, for $\ell \geq 0$,

$$
\mathbb{E}\left\{W_{\ell}(p)\right\}=\sum_{i=2}^{p} i \mathbb{P}\left\{X_{\ell}=i \mid X_{0}=p\right\}=\sum_{i=2}^{p} i\left(Q^{\ell}\right)_{p, i}
$$

For $\ell=0$, we have $\mathbb{E}\left\{W_{0}(p)\right\}=p$. For $\ell \geq 1$,
$\mathbb{E}\left\{W_{\ell}(p)\right\}=\sum_{i=2}^{p} i \sum_{j=i}^{p} Q_{p, j}\left(Q^{\ell-1}\right)_{j, i}=\sum_{j=2}^{n} Q_{p, j} \mathbb{E}\left\{W_{\ell-1}(j)\right\}$.
We pursue by recurrence over index $\ell$. The result being true for $\ell=0$, suppose that for every $j \geq 2$, we have $\mathbb{E}\left\{W_{\ell-1}(j)\right\}=$ $j / 2^{\ell-1}$. Then, for every $p \geq 2$,

$$
\begin{aligned}
\mathbb{E}\left\{W_{\ell}(p)\right\} & =Q_{p, p} \mathbb{E}\left\{W_{\ell-1}(p)\right\}+\sum_{j=2}^{p-1} Q_{p, j} \mathbb{E}\left\{W_{\ell-1}(j)\right\} \\
& =\frac{1}{2^{p-1}} \frac{p}{2^{\ell-1}}+\frac{1}{2^{p}} \sum_{j=2}^{p-1}\binom{p}{j} \frac{j}{2^{\ell-1}} \\
& =\frac{p}{2^{p} 2^{\ell-1}} \sum_{j=1}^{p}\binom{p-1}{j-1}=\frac{p}{2^{\ell}}
\end{aligned}
$$

We then have $\mathbb{E}\{N(p)\}=\sum_{\ell=0}^{\infty} \mathbb{E}\left\{W_{\ell}(p)\right\}=2 p$, which completes the proof.

Corollary 1: Each process competing for the Test\&Set object invokes 2 instances of the selector object in expectation.

Proof: Straightforward from Theorem 3 as $\mathbb{E}\{N(p)\} / p=$ 2.

Theorem 4 (Message complexity of Test\&Set()): The total number of messages exchanged by the randomized implementation of the Test\&Set object when concurrently invoked by $p \leq n$ processes is $O(n p)$.

Proof: Consider a Test\&Set execution with contention $p$. By Theorem 1, the message complexity of each invocation of the selector object requires $O(n p)$ messages. By Corollary 1, each competing process invokes the selector object twice in expectation. Consequently, the expected total number of messages exchanged by the Test\&Set algorithm with contention $p$ is $O(n p)$, and thus $O(n)$ messages are needed per competing process.

## V. Conclusion

In this paper we have presented a randomized solution to the Test\&Set operation in fully asynchronous systems prone to crash failures. This solution is built using a new building block, called selector. This solution has an adaptive step complexity. From a practical point of view, this property is very important as it guarantees that the Test\&Set operation on the attached distributed object solely depends on the number of processes $p$ that concurrently want to access this object, and not on the size $n$ of the system. Finally, the total number of messages involved by this operation is $O(n p)$, which improves upon all the existing adaptive implementations.

## REFERENCES

[1] Y. Afek, E. Gafni, J. Tromp, and P. Vitnyi. Wait-free test-and-set. In Proceedings of the International Workshop on Distributed Algorithms (WDAG, now DISC), pages 85-94, 1992.
[2] D. Alistarh and J. Aspnes. Sub-logarithmic test-and-set against a weak adversary. In Proceedings of the International Symposium on Distributed Computing (DISC), 2011.
[3] D. Alistarh, H. Attiya, S. Gilbert, A. Giurgiu, and R. Guerraoui. Fast randomized test-and-set and renaming. In Proceedings of the International Symposium on Distributed Computing (DISC), 2010.
[4] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. Journal of the ACM, 42(1):124-142, 1995.
[5] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. Journal of the ACM, 37(3):524-548, 1990.
[6] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), pages 27-30, 1983.
[7] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In Proceedings of the ACM Symposium on Theory of Computing (STOC), pages 42-51, 1993.
[8] B. Chor, M. Merritt, and D.B. Shmoys. Simple constant-time consensus protocols in realistic failure models. Journal of the ACM, 36(3):591614, 1989.
[9] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed commit with one faulty process. Journal of the ACM, 32(5):374-382, 1985.
[10] G. Giakkoupis and P. Woelfel. On the time and space complexity of randomized test-and-set. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), pages 19-28, 2012.
[11] W.M. Golab, D. Hendler, and P. Woelfel. An o(1) rmrs leader election algorithm. SIAM Journal on Computing, 39(7):2726-2760, 2010.
[12] M. Herlihy. Wait-free synchronization. ACM Transactions on Programming Languages and Systems, 13(1):124-149, 1991.
[13] A. Mostéfaoui and M. Raynal. Solving consensus using chandratoueg's unreliable failure detectors: a general quorum-based approach. In Proceedings of the International Symposium on Distributed Computing (DISC), pages 49-63, 1999.
[14] B. Sericola. Markov Chains: Theory, Algorithms and Applications. Iste Series, Wiley, 2013.
[15] J. Tromp and P. Vitanyi. A protocol for randomized anonymous twoprocess wait-free test-and-set with finite-state verification. In Proceedings of the ACM International Colloquium on Structural Information and Communication Complexity (SIROCCO), pages 275-291, 2002.

## APPENDIX

Theorem 2 The expected time $\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}$ needed to terminate the Test\&Set protocol when $p$ processors are initially competing satisfies

$$
\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}=O(\log (p))
$$

More precisely, there exists a positive integer $p_{0}$ such that, for every $p \geq p_{0}$, we have

$$
\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\} \leq 2 \log (p)
$$

Proof: Introducing the notation $u_{p}=\mathbb{E}\left\{T_{p} \mid X_{0}=p\right\}$ and $\gamma_{p, k}=\binom{p}{k} / 2^{p}$, Relation (1) becomes

$$
\left\{\begin{array}{l}
u_{2}=2  \tag{2}\\
u_{p}=\frac{2^{p-1}}{2^{p-1}-1}\left(1+\sum_{k=2}^{p-1} \gamma_{p, k} u_{k}\right), \text { for } p \geq 3
\end{array}\right.
$$

Consider first the coefficients $\gamma_{p, k}$. Using the Stirling formula, we have for all $p \geq 1$,

$$
\sqrt{2 \pi} \sqrt{p} p^{p} e^{-p} \leq p!\leq e \sqrt{p} p^{p} e^{-p}
$$

We then have, for $1 \leq k \leq p$,

$$
\gamma_{p, k}=2^{-p}\binom{p}{k} \leq e 2^{-p} \frac{\sqrt{p} p^{p} e^{-p}}{\sqrt{k} k^{k} e^{-k} \sqrt{p-k}(p-k)^{(p-k)} e^{-(p-k)}}=e \delta_{p, k}
$$

where

$$
\begin{aligned}
\delta_{p, k} & =\frac{\sqrt{p}}{\sqrt{k(p-k)}} 2^{-p} p^{p} k^{-k}(p-k)^{-(p-k)} \\
& =\frac{\sqrt{p}}{\sqrt{k(p-k)}} \exp (p \ln (p / 2)-k \ln (k)-(p-k) \ln (p-k))
\end{aligned}
$$

Now, taking any fixed value of $p \geq 3$, the two functions $\phi_{p}$ and $\psi_{p}$, defined by

$$
\begin{aligned}
& x \in[1, p-1] \longmapsto \phi_{p}(x)=\sqrt{x(p-x)} \\
& x \in[1, p-1] \longmapsto \psi_{p}(x)=p \ln (p / 2)-x \ln (x)-(p-x) \ln (p-x)
\end{aligned}
$$

both are increasing on $[1, p / 2]$ and decreasing on $[p / 2, p-1]$ (this is obvious for function $\phi_{p}$, while the derivative of $\psi_{p}$ with respect to $x$ is $\psi_{p}^{\prime}(x)=-\ln (x)+\ln (p-x)=-\ln (x /(p-x))$ which is $\geq 0$ when $1 \leq x \leq p / 2$, and $\leq 0$ when $p / 2 \leq x \leq p-1$ ).

We now take a small $\alpha, 0<\alpha<1 / 2$, and we estimate $\gamma_{p, k}$ for the $k$ 's belonging to the set $E=\left[2, p / 2-p^{\alpha+1 / 2}\right] \cup[p / 2+$ $\left.p^{\alpha+1 / 2}, p-1\right]$. Taking $0<\alpha<1 / 2$, there exists a $p_{1}$ such that for $p \geq p_{1}$, the two intervals forming $E$ are non empty. For this $\alpha$ and $p \geq p_{1}$, we have for all $x \in E$,

$$
\phi_{p}(x) \geq \sqrt{p-1} \text { and } \psi_{p}(x) \leq \psi_{p}\left(p / 2-p^{\alpha+1 / 2}\right)=\psi_{p}\left(p / 2+p^{\alpha+1 / 2}\right)
$$

with

$$
\begin{aligned}
& \psi_{p}\left(p / 2+p^{\alpha+1 / 2}\right) \\
& \quad=p \ln (p / 2)-\left(p / 2-p^{\alpha+1 / 2}\right) \ln \left(p / 2-p^{\alpha+1 / 2}\right)-\left(p / 2+p^{\alpha+1 / 2}\right) \ln \left(p / 2+p^{\alpha+1 / 2}\right) \\
& \quad=-\left(p / 2-p^{\alpha+1 / 2}\right) \ln \left(1-2 p^{\alpha-1 / 2}\right)-\left(p / 2+p^{\alpha+1 / 2}\right) \ln \left(1+2 p^{\alpha-1 / 2}\right)
\end{aligned}
$$

Using the Taylor-Lagrange formula, we get

$$
\begin{gathered}
-\ln \left(1+2 p^{\alpha-1 / 2}\right)=-2 p^{\alpha-1 / 2}+K_{1}\left(2 p^{\alpha-1 / 2}\right)^{2}, \text { with } 0 \leq K_{1} \leq 1 / 2 \text { for } p \geq p_{2} \\
-\ln \left(1-2 p^{\alpha-1 / 2}\right)=2 p^{\alpha-1 / 2}+K_{2}\left(2 p^{\alpha-1 / 2}\right)^{2}, \text { with } 0 \leq K_{2} \leq 1 \text { for } p \geq p_{3}
\end{gathered}
$$

This leads, for $p \geq \max \left(p_{1}, p_{2}, p_{3}\right)$, to

$$
\begin{aligned}
\psi_{p} & \left(p / 2+p^{\alpha+1 / 2}\right) \\
& =\left(p / 2-p^{\alpha+1 / 2}\right)\left(2 p^{\alpha-1 / 2}+4 K_{2} p^{2 \alpha-1}\right)-\left(p / 2+p^{\alpha+1 / 2}\right)\left(2 p^{\alpha-1 / 2}+4 K_{1} p^{2 \alpha-1}\right) \\
& =-4 p^{2 \alpha}+2\left(K_{2}-K_{1}\right) p^{2 \alpha}-\left(K_{2}+K_{1}\right) p^{3 \alpha-1} \\
& \leq-4 p^{2 \alpha}+2\left(K_{2}-K_{1}\right) p^{2 \alpha} \\
& \leq-2 p^{2 \alpha}
\end{aligned}
$$

We thus obtain, for all $k \in E$,

$$
\gamma_{p, k} \leq \frac{e \sqrt{p} \exp \left(\psi_{p}(k)\right)}{\phi_{p}(k)} \leq e \frac{\sqrt{p}}{\sqrt{p-1}} \exp \left(-2 p^{2 \alpha}\right) \leq e \sqrt{2} \exp \left(-2 p^{2 \alpha}\right)
$$

Using this bound and introducing

$$
M_{p}=\max _{2 \leq k \leq p} u_{k}
$$

we obtain from Relation (2),

$$
u_{p} \leq \frac{1}{1-2^{-(p-1)}}\left(1+e \sqrt{2} p e^{-2 p^{2 \alpha}} M_{p}+\sum_{k:|k-p / 2| \leq p^{\alpha+1 / 2}} \gamma_{p, k} u_{k}\right)
$$

Since $1 /(1-x) \leq 1+2 x$, for $0<x \leq 1 / 2$ and since $e \sqrt{2}\left(1+2^{-(p-2)}\right) p e^{-p^{2 \alpha}} \leq 1$, for $p$ large enough (i.e. $p \geq p_{4}$ for some $p_{4}$ whose precise value is irrelevant), we obtain

$$
\begin{equation*}
u_{p} \leq\left(1+2^{-(p-2)}\right)\left(1+\sum_{k:|k-p / 2| \leq p^{\alpha+1 / 2}} \gamma_{p, k} u_{k}\right)+e^{-p^{\alpha}} M_{p} \tag{3}
\end{equation*}
$$

We introduce a dyadic partition of the indices $p$, and set, for any $j \geq 1$, the notation

$$
U_{j}=\max _{2 \leq k \leq 2^{j}} u_{k}
$$

We take a fixed index $j \geq 2$, or $j \geq j_{0}$ for some $j_{0}$ whose value is irrelevant, and estimate $U_{j+1}$ as a function of $U_{j}$. To do so, we take $p$ such that $2^{j}<p \leq 2^{j+1}$ and we write

$$
\begin{align*}
\sum_{k:|k-p / 2| \leq p^{\alpha+1 / 2}} \gamma_{p, k} u_{k} & =\sum_{\substack{k:|k-p / 2| \leq p^{\alpha+1 / 2} \\
k \leq 2^{j}}} \gamma_{p, k} u_{k}+\sum_{\substack{k:|k-p / 2| \leq p^{\alpha+1 / 2} \\
2^{j}<k<p}} \gamma_{p, k} u_{k} \\
& \leq U_{j}\left(\sum_{\substack{ \\
0 \leq k \leq 2^{j}}} \gamma_{p, k}\right)+U_{j+1}\left(\sum_{2^{j}<k \leq p} \gamma_{p, k}\right) \\
& =U_{j} s_{p, j}+U_{j+1}\left(1-s_{p, j}\right) \tag{4}
\end{align*}
$$

where $s_{p, j}$ is defined by

$$
s_{p, j}=\sum_{0 \leq k \leq 2^{j}} \gamma_{p, k}
$$

and we have used the fact that $\sum_{k=0}^{p} \gamma_{p, k}=1$. Note the obvious estimate $0 \leq s_{p, j} \leq 1$ and note also that, since $p \in\left(2^{j}, 2^{j+1}\right]$, we have

$$
s_{p, j}=\sum_{0 \leq k \leq 2^{j}} \gamma_{p, k} \geq \sum_{0 \leq k \leq p / 2} \gamma_{p, k},
$$

while

$$
1=\sum_{k=0}^{p} \gamma_{p, k}=\sum_{0 \leq k \leq p / 2} \gamma_{p, k}+\sum_{p / 2<k \leq p} \gamma_{p, k}=\sum_{0 \leq k \leq p / 2} \gamma_{p, k}+\sum_{0 \leq k<p / 2} \gamma_{p, k} \leq 2 \sum_{0 \leq k \leq p / 2} \gamma_{p, k},
$$

from which it comes

$$
s_{p, j} \geq 1 / 2
$$

Relations (3) and (4) eventually provide

$$
\begin{aligned}
U_{j+1} & \leq\left(1+2^{-(p-2)}\right)\left(1+U_{j} s_{p, j}+U_{j+1}\left(1-s_{p, j}\right)\right)+e^{-p^{\alpha}} M_{p} \\
& \leq\left(1+2^{-(p-2)}\right)\left(1+\frac{U_{j}+U_{j+1}}{2}\right)+e^{-p^{\alpha}} M_{p}
\end{aligned}
$$

where we have used $s_{p, j} \geq 1 / 2$ together with $U_{j} \leq U_{j+1}$. In other words, and taking into account $p \in\left(2^{j}, 2^{j+1}\right]$, we have

$$
U_{j+1} \leq\left(1+2^{-\left(2^{j}-2\right)}\right)\left(1+\frac{U_{j}+U_{j+1}}{2}\right)+e^{-2^{j \alpha}} U_{j+1}
$$

Hence we arrived at

$$
\begin{equation*}
U_{j+1} \leq \frac{1+2^{-\left(2^{j}-2\right)}}{1-2^{-\left(2^{j}-2\right)}-2 e^{-2^{j \alpha}}}\left(2+U_{j}\right) \tag{5}
\end{equation*}
$$

For later convenience, we rewrite (5) in a more convenient form. To do so, we fix some $\beta$ such that $0<\beta<\alpha$ and we introduce

$$
\beta_{j}=e^{-2^{j \beta}}
$$

It is clear that for $j$ large enough $\left(j \geq j_{1}\right.$ for some $\left.j_{1}\right)$, Relation (5) implies that

$$
\begin{equation*}
U_{j+1} \leq\left(1+\beta_{j}\right)\left(2+U_{j}\right) \tag{6}
\end{equation*}
$$

We are now in position to conclude. Formula (6) implies that

$$
\begin{aligned}
U_{j+1} & \leq 2\left[\left(1+\beta_{j}\right)+\left(1+\beta_{j}\right)\left(1+\beta_{j-1}\right)+\cdots+\left(1+\beta_{j}\right)\left(1+\beta_{j-1}\right) \cdots\left(1+\beta_{j_{1}}\right)\right] \\
& +\left(1+\beta_{j}\right)\left(1+\beta_{j-1}\right) \cdots\left(1+\beta_{j_{1}}\right) U_{j_{1}}
\end{aligned}
$$

Introducing the quantities

$$
\Pi_{j}:=\prod_{k=j_{0}-1}^{j}\left(1+\beta_{k}\right)
$$

the above bound rewrites

$$
U_{j+1} \leq 2\left[\frac{\Pi_{j}}{\Pi_{j-1}}+\frac{\Pi_{j}}{\Pi_{j-2}}+\cdots+\frac{\Pi_{j}}{\Pi_{j_{1}}}\right]+\frac{\Pi_{j}}{\Pi_{j_{1}}} U_{j_{1}}
$$

Hence, since the infinite product $\prod_{j \geq j_{1}}\left(1+\beta_{j}\right)$ clearly converges, we have $\Pi_{j} \rightarrow \Pi>0$ as $j \rightarrow \infty$ and we may write,

$$
U_{j+1} \leq 2 \Pi_{j} \sum_{\ell=j_{1}}^{j-1} \frac{1}{\Pi_{\ell}}+\frac{\Pi_{j}}{\Pi_{j_{1}}} U_{j_{1}}
$$

We denote by $\widetilde{U}_{j}$ the right hand side of this last inequality. It is clear, using a standard fact about diverging series, that

$$
\begin{aligned}
& \Pi_{j} \sum_{\ell=j_{1}}^{j-1} \frac{1}{\Pi_{\ell}} \underset{j \rightarrow \infty}{\sim} \Pi \sum_{\ell=j_{1}}^{j-1} \frac{1}{\Pi} \underset{j \rightarrow \infty}{\sim} j \\
& \frac{\Pi_{j}}{\Pi_{j_{1}}} U_{j_{1}} \underset{j \rightarrow \infty}{\sim} \frac{\Pi}{\Pi_{j_{1}}} U_{j_{1}} .
\end{aligned}
$$

We deduce that

$$
\begin{equation*}
U_{j+1} \leq \widetilde{U}_{j} \quad \text { with } \widetilde{U}_{j} \underset{j \rightarrow \infty}{\sim} 2 j \tag{7}
\end{equation*}
$$

In particular, defining $\widetilde{u}_{p}=\widetilde{U}_{j}$, for $2^{j}<p \leq 2^{j+1}$, we easily deduce that

$$
\begin{equation*}
u_{p} \leq \widetilde{u}_{p} \text { with } \widetilde{u}_{p} \underset{p \rightarrow \infty}{\sim} 2 \log (p) \tag{8}
\end{equation*}
$$

This completes the proof.


[^0]:    ${ }^{1}$ In the consensus problem, each process proposes a value and every correct processes eventually decides a value, such that a unique value is decided and this value has been proposed by at least one process.

