Optimal Storage under
Unsynchronized Mobile Byzantine Faults

Silvia Bonomi*, Antonella Del Pozzo*, Maria Potop-Butucaru†, and Sébastien Tixeuil‡
*Sapienza Università di Roma, Via Ariosto 25, 00185 Roma, Italy †UPMC Sorbonne Universités

Abstract—In this paper we prove lower and matching upper bounds for the number of servers required to implement a regular shared register that tolerates unsynchronized Mobile Byzantine failures. We consider the strongest model of Mobile Byzantine failures to date: agents are moved arbitrarily by an omnipresent adversary from a server to another in order to deviate their computation in an unforeseen manner. When a server is infected by a Byzantine agent, it behaves arbitrarily until the adversary decides to “move” the agent to another server. Previous approaches considered asynchronous servers with synchronous mobile Byzantine agents (yielding impossibility results), and synchronous servers with synchronous mobile Byzantine agents (yielding optimal solutions for regular register implementation, even in the case where servers and agents periods are decoupled).

We consider the remaining open case of synchronous servers with unsynchronized agents, that can move at their own pace, and change their pace during the execution of the protocol. Most of our findings relate to lower bounds, and characterizing the model parameters that make the problem solvable. It turns out that unsynchronized mobile Byzantine agent movements requires completely new proof arguments, that can be of independent interest when studying other problems in this model. Additionally, we propose a generic server-based algorithm that emulates a regular register in this model, that is tight with respect to the number of mobile Byzantine agents that can be tolerated. Our emulation spans two awareness models: servers with and without self-diagnose mechanisms. In the first case servers are aware that the mobile Byzantine agent has left and hence they can stop running the protocol until they recover a correct state while in the second case, servers are not aware of their faulty state and continue to run the protocol using an incorrect local state.

Index Terms—mobile Byzantine failures, regular register, round free synchronous computation

I. INTRODUCTION

Byzantine fault tolerance is a fundamental building block in distributed system, as Byzantine failures include all possible faults, attacks, virus infections and arbitrary behaviors that can occur in practice (even unforeseen ones). The classical setting considers Byzantine participants remain so during the entire execution, yet software rejuvenation techniques increase the possibility that a corrupted node does not remain corrupted during the whole system execution and may be aware of its previously compromised status.

Mobile Byzantine Failures (MBF) models have been recently introduced to integrate those concerns. Then, faults are represented by Byzantine agents that are managed by an omniscient adversary that “moves” them from a host process to another, an agent being able to corrupt its host in an unforeseen manner. MBF investigated so far consider mostly round-based computations, and can be classified according to Byzantine mobility constraints: (i) constrained mobility agents may only move from one host to another when protocol messages are sent (similarly to how viruses would propagate), while (ii) unconstrained mobility agents move independently of protocol messages. In the case of unconstrained mobility, several variants were investigated: (i) mobile Byzantine agents leave. Finally, Garay and Sasaki and Bonnet et al. consider that processes execute synchronous rounds composed of three phases: send, receive, and compute. Between two consecutive such synchronous rounds, Byzantine agents can move from one node to another. Hence the set of faulty hosts at any given time has a bounded size, yet its membership may evolve from one round to the next. The main difference between the aforementioned four works lies in the knowledge that hosts have about their previous infection by a Byzantine agent. In Garay’s model, a host is able to detect its own infection after the Byzantine agent left it. Sasaki et al. investigate a model where hosts cannot detect when Byzantine agents leave. Finally, Bonnet et al. considers an intermediate setting where cured hosts remain in control on the messages they send (in particular, they send the same message to all destinations, and they do not send obviously fake information, e.g. fake id). Those subtle differences on the power of Byzantine agents turns out to have an important impact on the bounds for solving distributed problems.

A first step toward decoupling algorithm rounds from mobile Byzantine moves is due to Bonomi et al. In their model, mobile Byzantine movements are either: (i) synchronized, but the period of movement is independent to that of algorithm rounds, (ii) independent time bounded, meaning that Byzantine agents are only requested to remain some minimum amount of time at any occupied node, or (iii) independent time unbounded, which can be seen as a special case of (ii) when the minimum amount of time is one time unit. In particular, the Bonomi et al. model implies that Byzantine moves are no more related to messages that are exchanged through the protocol.

a) Register Emulation.: Traditional solutions to build a Byzantine tolerant storage service (a.k.a. register emulation) can be divided into two categories: replicated state machines and Byzantine quorum systems. Both approaches are based on the idea that the current state of the
we assume that an arbitrary number of clients may crash, and with smart choices of quorums thresholds command optimal resilience (with respect to the number of Byzantine agents). As expected, the lower bounds results require completely new proof techniques that are of independent interest while studying other classical problems in the context of unsynchronized mobile Byzantine agents. The round-free model [8] where Byzantine moves are decoupled from protocol rounds also enables optimal solutions (with respect to the number of Byzantine agents) for the implementation of regular registers. However, this last solution requires Byzantine agents to move in synchronous steps, whose duration for the entire execution is fixed, so the movements of Byzantine agents is essentially synchronous. As it is impossible to solve the register emulation problem when processes are asynchronous Byzantine agents are synchronous [8], the only case remaining open is that of asynchronous processes and unsynchronized Byzantine agents.

b) Our Contribution. We relax the main assumption made for obtaining positive results in the round-free model: Byzantine moves are no more synchronized. The main contribution of this paper is to thoroughly study the impact of unsynchronized mobile Byzantine agents on the register emulation problem. We present lower and matching upper bounds for implementing a regular register in the unsynchronized mobile Byzantine model. We first explore and characterize the key parameters of the model that enable problem solvability. As expected, the lower bounds results require completely new proof techniques that are of independent interest while studying other classical problems in the context of unsynchronized mobile Byzantine agents. When the problem is solvable, it turns out that minor changes to existing quorum-based protocols joint with smart choices of quorums thresholds command optimal resilience (with respect to the number of Byzantine agents). Table I summarizes all the lower bounds for the various models, the newly obtained results are presented in boldface.

II. SYSTEM MODEL

We consider a distributed system composed of an arbitrary large set of client processes $C$, and a set of $n$ server processes $S = \{s_1, s_2 \ldots s_n\}$. Each process in the distributed system (i.e., both servers and clients) is identified by a unique identifier. Servers run a distributed protocol emulating a shared memory abstraction, and clients are unaware of the protocol run by the servers. The passage of time is measured by a fictional global clock (e.g., that spans the set of natural integers), whose processes are unaware of. At each time instant $t$, each process (either client or server) is characterized by its internal state, i.e., by the set of its local variables and their assigned values. We assume that an arbitrary number of clients may crash, and that up to $f$ servers host, at any time $t$, a Byzantine agent. Furthermore, servers processes execute the same algorithm, and cannot rely on high level primitives such as consensus or total order broadcast.

Communication model. Processes communicate through message passing. In particular, we assume that: (i) each client $c_i \in C$ can communicate with every server through a broadcast primitive, (ii) each server can communicate with every other server through a broadcast primitive, and (iii) each server can communicate with a particular client through a send unicast primitive. We assume that communications are authenticated (i.e., given a message $m$, the identity of its sender cannot be forged) and reliable (i.e., spurious messages are not created and sent messages are neither lost nor duplicated).

Timing Assumptions. The system is round-free synchronous in the sense that: (i) the processing time of local computations (except for Wait statements) are negligible with respect to communication delays, and are assumed to be equal to 0, and (ii) messages take time to travel to their destination processes. In particular, concerning point-to-point communications, we assume that if a process sends a message $m$ at time $t$ then it is delivered by time $t + \delta_p$ (with $\delta_p > 0$). Similarly, let $t$ be the time at which a process $p$ invokes the broadcast primitive, then there is a constant $\delta_b$ (with $\delta_b \geq \delta_p$) such that all servers have delivered $m$ at time $t + \delta_b$. For the sake of presentation, in the following we consider a unique message delivery delay $\delta$ (equal to $\delta_b \geq \delta_p$), and assume $\delta$ is known to every process.

Computation model. Each process of the distributed system executes a distributed protocol $\mathcal{P}$ that is composed by a set of distributed algorithms. Each algorithm in $\mathcal{P}$ is represented by a finite state automaton and it is composed of a sequence of computation and communication steps. A computation step is represented by the computation executed locally to each process while a communication step is represented by the sending and the delivering events of a message. Computation steps and communication steps are generally called events.

Definition 1 (Execution History): Let $\mathcal{P}$ be a distributed protocol. Let $H$ be the set of all the events generated by $\mathcal{P}$ at any process $p_i$ in the distributed system and let $\to$ be the happened-before relation. An execution history (or simply history) $H = (H, \to)$ is a partial order on $H$ satisfying the relation $\to$.

Definition 2 (Valid State at time $t$): Let $\hat{H} = (H, \to)$ be an execution history of a generic computation and let $\mathcal{P}$ be the corresponding protocol. Let $p_i$ be a process and let $\text{state}_{p_i}$ be the state of $p_i$ at some time $t$. $\text{state}_{p_i}$ is said to be valid at time $t$ if it can be generated by executing $\mathcal{P}$ on $\hat{H}$.

MBF model. We now recall the generalized Mobile Byzantine Failure model [8]. Informally, in the MBF model, when a Byzantine agent is hosted by a process, the agent takes entire control of its host making it Byzantine faulty (i.e., it can corrupt the host’s local variables, forces it to send arbitrary messages, etc.). Then, the Byzantine agent leaves its host with a possible corrupted state (that host is called cured) before reaching another host. We assume that any process previously hosting a Byzantine agent has access to a tamper-proof memory storing the correct protocol code. However, a cured server may still have a corrupted internal state, and thus cannot be considered
correct. The moves of a Byzantine agent are controlled by an omniscient adversary.

**Definition 3 (Correct process at time t):** Let \( \hat{H} = (H, \rightarrow) \) be a history, and let \( P \) be the protocol generating \( \hat{H} \). A process is correct at time \( t \) if (i) it is correctly executing \( P \), and (ii) its state is valid at time \( t \). We denote by \( Co(t) \) the set of correct processes at time \( t \). Given a time interval \( [t, t'] \), we denote by \( Co([t, t']) \) the set of all processes that remain correct during \( [t, t'] \) (i.e., \( Co([t, t']) = \cap_{\tau \in [t, t']} Co(\tau) \)).

**Definition 4 (Byzantine process at time t):** Let \( \hat{H} = (H, \rightarrow) \) be a history, and let \( P \) be the protocol generating \( \hat{H} \). A process is Byzantine at time \( t \) if it is controlled by a Byzantine agent and does not execute \( P \). We denote by \( B(t) \) the set of Byzantine processes at time \( t \). Given a time interval \( [t, t'] \), we denote by \( B([t, t']) \) the set of all processes that remain Byzantine during \( [t, t'] \) (i.e., \( B([t, t']) = \cap_{\tau \in [t, t']} B(\tau) \)).

**Definition 5 (Cured process at time t):** Let \( \hat{H} = (H, \rightarrow) \) be a history, and let \( P \) be the protocol generating \( \hat{H} \). A process is cured at time \( t \) if (i) it is correctly executing \( P \), and (ii) its state is not valid at time \( t \). We denote by \( Cu(t) \) the set of cured processes at time \( t \). Given a time interval \( [t, t'] \), we denote by \( Cu([t, t']) \) the set of all processes that remain cured during \( [t, t'] \) (i.e., \( Cu([t, t']) = \cap_{\tau \in [t, t']} Cu(\tau) \)).

With respect to the movements of agents, we consider the independent time-bounded (ITB) model: each mobile Byzantine agent \( ma_i \) is forced to remain on a host for at least a period \( \Delta_i \). Given two mobile Byzantine Agents \( ma_i \) and \( ma_j \), their movement periods \( \Delta_i \) and \( \Delta_j \) may be different. Note that previous results considering decoupled Byzantine moves \( \delta \) were established in the weaker \( \Delta \)-synchronized model, where the external adversary moves all controlled mobile Byzantine agents at the same time \( t \), and their movements happen periodically with period \( \Delta \). None of those properties remain valid in our model.

Concerning the knowledge that each process has about its failure state, we distinguish the following two cases: **Cured Aware Model (CAM):** at any time \( t \), every process is aware about its failure state; **Cured Unaware Model (CUM):** at any time \( t \), every process is not aware about its failure state.

We assume that the adversary can control at most \( f \) Byzantine agents at any time (i.e., Byzantine agents do not replicate while moving). In our work, only servers can be affected by the mobile Byzantine agent. It follows that, at any time \( t \mid B(t) \mid \leq f \). However, during the system lifetime, all servers may be hosting a Byzantine agent at some point (i.e., none of the servers is guaranteed to remain correct forever).

**Register Specification.**

A register is a shared variable accessed by a set of processes, called clients, through two operations, namely read and write. Informally, the write operation updates the value stored in the shared variable, while the read obtains the value contained in the variable (i.e., the last written value). Every operation issued on a register is, generally, not instantaneous and it can be characterized by two events occurring at its boundaries: an invocation event and a reply event. These events occur at two time instants (called the invocation time and the reply time) according to the fictional global time.

An operation \( op \) is complete if both the invocation event and the reply event occurred, otherwise, it failed. Given two operations \( op \) and \( op' \), their invocation times \( (t_B(op) \) and \( t_B(op')) \) and reply times \( (t_E(op) \) and \( t_E(op') \) ). \( op \) precedes \( op' \) (\( op < op' \)) if and only if \( t_E(op) < t_E(op') \). If \( op \) does not precede \( op' \) and \( op' \) does not precede \( op \) then \( op \) and \( op' \) are concurrent (noted \( op \parallel op' \)). Given a write \( v \) operation, the value \( v \) is said to be written when the operation is complete.

In this paper, we consider a single-writer/multi-reader (SWMR) regular register, as defined by Lamport [11], which is specified as follows:

- Termination: if a correct client invokes an operation \( op \), \( op \) completes.
- Validity: A read returns the last written value before its invocation (i.e. the value written by the latest completed write preceding it), or a value written by a write concurrent with it.

### III. Lower Bounds

In this section we provide lower bounds for \( (ITB, CAM) \) and \( (ITB, CUM) \) models. In particular we first prove lower bounds for the \( (\Delta S, CAM) \) and \( (\Delta S, CUM) \) models and then we extend those results to \( (ITB, CAM) \) and \( (ITB, CUM) \) models. The detailed proofs accompanied by corresponding explanations can be found in Appendix A. We consider a read operation whose duration \( T_r \) is such that \( T_r \geq 2\delta \) (according

---

<table>
<thead>
<tr>
<th>Agents moves</th>
<th>Synchronized [8]</th>
<th>Unsynchronized [this paper]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cured state awareness</td>
<td>Aware</td>
<td>Unaware</td>
</tr>
<tr>
<td>( \delta \leq \Delta &lt; 2\delta )</td>
<td>( 2f + 1 )</td>
<td>( 8f + 1 )</td>
</tr>
<tr>
<td>( 2\delta \leq \Delta &lt; 3\delta )</td>
<td>( 4f + 1 )</td>
<td>( 5f + 1 )</td>
</tr>
</tbody>
</table>

---

*1* It is trivial to prove that in our model, if clients are Byzantine, it is impossible to implement deterministically even a safe register. A Byzantine client may always introduce a corrupted value, and a server cannot distinguish between a correct client and a Byzantine one.
to previous work [8], each read operation requires at least a request-reply exchange. Our lower bound proof constructs two indistinguishable executions. The tricky part is to characterize the set of messages sent by correct and incorrect servers when the read operation has variable duration. In the following, we first characterize such sets with respect to \( \Delta, T_r \) and \( \gamma \) and \( \gamma \), which is the required time to finalize the maintenance operation. Then, we show how to build two indistinguishable executions. For clarity, in the sequel we note correct message/request/reply a message that carries a valid value when it is sent. Otherwise, the message is incorrect.

It has been proven [8] that a protocol \( \mathcal{P}_{\text{reg}} \) implementing a regular register in a mobile Byzantine setting must include in addition to the mandatory read and write operations an additional operation, maintenance, defined below.

**Definition 6 (Maintenance operation maintenance):** A maintenance operation is an operation that, when executed by a process \( p_i \), terminates at some time \( t \) leaving \( p_i \) with a valid state at time \( t \) (i.e., it guarantees that \( p_i \) is correct at time \( t \)).

The following definition defines \( \gamma \), the duration of a maintenance operation.

**Definition 7 (Curing time, \( \gamma \)):** We define \( \gamma \) as the maximum time a server can be in a cured state. More formally, let \( T_c \) the time at which server \( s_k \) is left by a mobile agent, let \( op_M \) the first maintenance operation that correctly terminates, then \( t_E(op_M) - T_c \leq \gamma \).

In order to build our indistinguishable executions, we define below a scenario to characterize agents movements. Then with respect to this scenario, we construct two indistinguishable executions.

**Definition 8 (Scenario \( S^* \)):** Let \( S^* \) be the following scenario: for each time \( T_i, i \geq 0 \) the affected servers are \( s_{(i \ mod \ n)f+1}, \ldots, s_{(i \ mod \ n)f+f} \).

Let us characterize the \( \mathcal{P}_{\text{reg}} \) protocol in the most general possible way. By definition a register abstraction involves read() and write() operations issued by clients. A read operation involves at least a request – reply communication pattern (i.e., two communication steps). Thus, given the system synchrony, a read() operation \( op_R \) lasts at least \( T_c \geq 2\delta \) time. Moreover we consider that a correct server sends a reply message in two occasions: (i) after the delivery of a request message, and (ii) right after it changes its state, at the end of the maintenance operation if an \( op_R \) is occurring. The latter case exploits the maintenance operation allowing servers to reply with a valid value in case they were Byzantine at the beginning of the read operation. Moreover we assume that in \((*, CAM)\) model servers in a cured state do not participate to the read operation.

Notice that those servers are aware of their current cured state and are aware of their impossibility to send correct replies. Even though those may seems not very general assumptions, let us just consider that we are allowing servers to correctly contribute to the computation as soon as they can and stay silent when they can not and under those assumptions we prove lower bounds. Thus if we remove those assumptions the lower bounds do not decreases. Scenario and protocol has been characterized.

Now we aim to characterize the set of servers, regarding their failure states, that can appear during the execution of the protocol, in particular during the read() operation. Those sets allow us to characterize correct and incorrect messages that a client delivers during a read() operation.

**Definition 9 (Failure State of servers in a time interval):** Let \( [t,t+T_r] \) be a time interval and let \( t', t' > 0 \), be a time instant.

Let \( s_i \) be a server and \( state_i \) be \( s_i \) state, \( state_i \in \{\text{correct, cured, Byzantine}\} \). Let \( S(t') \) be the set of servers \( s_i \) that are in the state \( state_i \) at \( t' \), \( S(t') \in \{\text{Co}(t'), \text{Cu}(t'), \text{B}(t')\} \).

\( S(t,t+T_r) \) is the set of servers that have been in the state \( state_i \) for at least one time unit during \( [t,t+T_r] \). More formally, \( S(t,t+T_r) = \bigcup_{t \leq t' \leq t+T_r} S(t') \).

**Definition 10 (CBC(t,t+T_r)):** Let \( [t,t+T_r] \) be a time interval, \( CBC(t,t+T_r) \) denotes servers that during a time interval \( [t,t+T_r] \) belong first to \( \text{B}(t,t+T_r) \) or \( \text{Cu}(t) \) (only in \((\Delta S, CU M)\) model) and then to \( \text{Co}(t,\delta,t+T_r-\delta) \) or vice versa.

In particular let us denote:
- \( \text{BC}(t,t+T_r) \) servers that during a time interval \( [t,t+T_r] \) belong to \( \text{B}(t,t+T_r) \) or \( \text{Cu}(t) \) (only in \((\Delta S, CU M)\) model) and to \( \text{Co}(t,\delta, t+T_r-\delta) \).
- \( \text{CB}(t,t+T_r) \) servers that during a time interval \( [t,t+T_r] \) belong to \( \text{Co}(t,\delta, t+T_r-\delta) \) and to \( \text{B}(t,t+T_r) \).

**Definition 11 (Sil(t,t+T_r)):** Let \( [t,t+T_r] \) be a time interval.

\( \text{Sil}(t,t+T_r) \) is the set of servers in \( \text{Cu}(t,\delta, t+T_r-\delta) \).

**Definition 12 (Max\( \text{B}(t,t+T_r) \)):** Let \( S \) be a scenario and \( [t,t+T_r] \) a time interval. The cardinality of \( \text{B}(t,t+T_r) \) is maximum with respect to \( S \) if for any \( t', t' > 0 \), we have that \( |\text{B}(t,t+T_r)| \geq |\text{B}(t',t'+T_r)| \). Then we call the value of such cardinality as \( \text{Max}\text{B}(t,t+T_r) \).

**Definition 13 (Max\( \text{Sil}(t,t+T_r) \)):** Let \( S \) be a scenario and \( [t,t+T_r] \) a time interval. The cardinality of \( \text{Sil}(t,t+T_r) \) is maximum with respect to \( S \) if for any \( t', t' > 0 \), we have that \( |\text{Sil}(t,t+T_r)| \geq |\text{Sil}(t',t'+T_r)| \) and \( \text{B}(t,t+T_r) = \text{Max}\text{B}(t,t+T_r) \). Then we call the value of such cardinality as \( \text{Max}\text{Sil}(t,t+T_r) \).

**Definition 14 (Max\( \text{Cu}(t) \)):** Let \( S \) be a scenario and \( t \) be a time instant. The cardinality of \( \text{Cu}(t) \) is maximum with respect to \( S \) if for any \( t', t' > 0 \), we have that \( |\text{Cu}(t,t+T_r)| \leq |\text{Cu}(t',t'+T_r)| \) and \( \text{B}(t,t+T_r) = \text{Max}\text{B}(t,t+T_r) \). We call the value of such cardinality as \( \text{Max}\text{Cu}(t) \).

**Definition 15 (min\( \text{Co}(t,t+T_r) \)):** Let \( S \) be a scenario and \( [t,t+T_r] \) be a time interval then \( \min\text{Co}(t,t+T_r) \) denotes the minimum number of correct servers during a time interval \( [t,\delta, t+T_r-\delta] \).

**Definition 16 (min\( \text{BC}(t,t+T_r) \)):** Let \( [t,t+T_r] \) be a time interval then \( \min\text{BC}(t,t+T_r) \) denotes the minimum number of servers that during a time interval \( [t,t+T_r] \), belong first to \( \text{B}(t,t+T_r) \) or \( \text{Cu}(t) \) (only in \((\Delta S, CU M)\) model) and then to \( \text{Co}(t,\delta, t+T_r-\delta) \) or vice versa and \( \text{B}(t,t+T_r) = \text{Max}\text{B}(t,t+T_r) \).

In particular let us denote as:
- \( \min\text{BC}(t,t+T_r) \) the minimum number of servers that during a time interval \( [t,t+T_r] \) belong to \( \text{B}(t,t+T_r) \) or...
Cu(t) (only in $\Delta S, CUM$ model) and to $\tilde{Co}(t+\delta, t+T_r - \delta)$.

- $\tilde{minCBC}(t, t+T_r)$ the minimum number of servers that during a time interval $[t, t+T_r]$ belong to $\tilde{Co}(t+\delta, t+T_r - \delta)$ and to $\tilde{B}(t, t+T_r)$.

Since we consider only scenario $S^*$, then we can omit the subscript related to $S^*$ and write directly $\tilde{maxB}(t, t+T_r), \tilde{minSil}(t, t+T_r), \tilde{maxCu}(t), \tilde{minCo}(t, t+T_r)$ and $\tilde{minCBC}(t, t+T_r)$.

Lemma 1: $\tilde{maxB}(t, t+T_r) = ([T_{\Delta r}] + 1)f$

At this point we can compute how many correct and incorrect replies a client $c_k$ can deliver in the worst case scenario during a time interval $[t, t+T_r]$. Trivially, $c_k$ in order to distinguish correct and incorrect replies needs to get $\tilde{minReplies}_{Co}(t, t+T_r)_k > \tilde{maxReplies}_{NCo}(t, t+T_r)_k$. It follows that the number of correct servers has to be enough to guarantee this condition. Table I follows directly from this observation. In a model with $b$ Byzantine (non mobile) a client $c_i$ requires to get at least $2b + 1$ replies to break the symmetry and thus $n \geq 2b + 1$. In presence of mobile Byzantine we have to sum also servers that do not reply (silent) and do not count twice servers that reply with both incorrect and correct values.

Definition 17 ($\tilde{maxReplies}_{NCo}(t, t+T_r)_k$): Let $\tilde{maxReplies}_{NCo}(t, t+T_r)_k$ be the multi-set maintained by client $c_k$ containing $m_{ij}$ elements, where $m_{ij}$ is the $i$-th message delivered by $c_k$ and sent at time $t'$, $t' \in [t, t+T_r]$ by $s_j$ such that $s_j \notin Co(t')$.

Considering the definitions of both $\tilde{maxB}(t, t+T_r)$ and $\tilde{maxCu}(t)$ the next Corollary follows:

Corollary 1: In the worst case scenario, during a read operation lasting $T_r \geq 2\delta$ issued by client $c_i$, $c_i$ delivers $\tilde{maxB}(t, t+T_r)$ incorrect replies in the $(\Delta S, CAM)$ model and $\tilde{maxB}(t, t+T_r) + \tilde{maxCu}(t)$ incorrect replies in the $(\Delta S, CUM)$ model.

Definition 18 ($\tilde{minReplies}_{Co}(t, t+T_r)_k$): Let $\tilde{minReplies}_{Co}(t, t+T_r)_k$ be the multi-set maintained by client $c_k$ containing $m_{ij}$ elements, where $m_{ij}$ is the $i$-th message delivered by $c_k$ and sent at time $t'$, $t' \in [t, t+T_r]$ by $s_j$ such that $s_j \in Co(t')$.

Theorem 1: If $n < n_{CA, LB} (n < n_{CUM, LB})$ as defined in Table VI then there does not exist a protocol $P_{reg}$ solving the safe register specification in $(\Delta S, CAM)$ model ($(\Delta S, CUM)$ model respectively).

Proof Let us suppose that $n < n_{CA, LB} (n < n_{CUM, LB})$ and that protocol $P_{reg}$ does exist. If a client $c_i$ invokes a read operation $op$, lasting $T_r \geq 2\delta$ time, if no write operations occur, then $c_i$ returns a valid value at time $t_B(op)$. Let us consider an execution $E_0$ where $c_i$ invokes a read operation $op$ and let $0$ be the valid value at $t_B(op)$. Let us assume that all Byzantine servers involved in such operation reply once with 1. From Corollaries 4 and 5 $c_i$ collects $\tilde{maxReplies}_{NCo}(t, t+T_r)_i$ occurrences of 1 and $\tilde{minReplies}_{Co}(t, t+T_r)_i$ occurrences of 0. Since $P_{reg}$ exists and no write operations occur, then $c_i$ returns 0. Let us now consider a another execution $E_1$ where $c_i$ invokes a read operation $op$ and let 1 be the valid value at $t_B(op)$. Let us assume that all Byzantine servers involved in such operation replies once with 0. From Corollaries 4 and 5 and Corollary 6 and Corollary 7 $c_i$ collects $\tilde{maxReplies}_{NCo}(t, t+T_r)_i$ occurrences of 0 and $\tilde{minReplies}_{Co}(t, t+T_r)_i$ occurrences of 1. Since $P_{reg}$ exists and no write operations occur, then $c_i$ returns 1.

From Lemma 3 and using values in Table VI we obtain following equations for both models:

- $(\Delta S, CAM)$:
  - $\tilde{maxReplies}_{NCo}(t, t+T_r)_i = \tilde{maxB}(t, t+T_r) = ([T_{\Delta r}] + 1)f$
  - $\tilde{minReplies}_{Co}(t, t+T_r)_i = n - [\tilde{maxB}(t, t+T_r) + \tilde{maxSil}(t, t+T_r)] + \tilde{minCBC}(t, t+T_r) = [2(\tilde{maxB}(t, t+T_r)) + \tilde{maxSil}(t, t+T_r)] - \tilde{minCBC}(t, t+T_r)$
  - $+ \tilde{minCBC}(t, t+T_r) =$
  - $\tilde{maxB}(t, t+T_r) + \tilde{maxCu}(t) =$

- $(\Delta S, CUM)$:
  - $\tilde{maxReplies}_{NCo}(t, t+T_r)_i =$
  - $\tilde{maxReplies}_{Co}(t, t+T_r)_i =$

It follows that in $E_0$ and $E_1$ $c_i$ delivers the same occurrences of 0 and 1, both executions are indistinguishable leading to a contradiction.

\[ \square \]

$\tilde{maxReplies}_{NCo}(t, t+T_r)_i$ and $\tilde{minReplies}_{Co}(t, t+T_r)_i$ are equal independently from the value assumed by $T_r$, the read() operation duration. From the equation just used in the previous lemma the next Corollary follows.

Corollary 2: For each $T_r \geq 2\delta$ if $n > n_{CA, LB} (n > n_{CUM, LB})$ then $\tilde{maxReplies}_{NCo}(t, t+T_r)_i < \tilde{minReplies}_{Co}(t, t+T_r)_i$.

At this point we compute $\tilde{maxCu}(t), \tilde{maxSil}(t, t+T_r)$ and $\tilde{minCBC}(t, t+T_r)$ to finally state exact lower bounds depending on the system parameters, in particular depending on $\Delta, \gamma$ and the servers awareness. An interested reader can find proofs in Appendix A. In Table III are reported all the results for $(\Delta S, *)$ models.

Lemma 2: Let $n_{CA, LB} \leq \alpha_*(\Delta, \delta, \gamma)f$ be the impossibility results holding in the $(\Delta S, *)$ model for $f = 1$. If there exists a tight protocol $P_{reg}$ solving the safe register for $n \geq \alpha_*(\Delta, \delta, \gamma)f + 1$ ($f \geq 1$) then all the Safe Register impossibility results that hold in the $(\Delta S, *)$ models hold also in the $(ITB, *)$ and $(ITU, *)$ models.
Proof. Let us consider the scenario $S^*$ for $f = 1$ and a read() operation time interval $[t, t + T_r]$, $t \geq 0$. Depending on the value of $t$ there can be different (but finite) read scenarios, $rs_1, rs_2, \ldots, rs_s$. By hypothesis there exists $P_{\text{reg}}$ solving the safe register for $n \geq \alpha_s f \Delta, \delta, \gamma + 1$ then among the read scenarios $\mathcal{RS} = \{rs_1, rs_2, \ldots, rs_s\}$ all the possible worst case scenarios $\{wrs_1, \ldots, wrs_w\} \subseteq \mathcal{RS}$ hold for $n = \alpha_s \Delta, \delta, \gamma + f$ (meaning that $P_{\text{reg}}$ does not exist). We can say that those worst scenarios are equivalent in terms of replicas, i.e., for each $wrs_k$ it is possible to build an impossibility run if $n = \alpha_s \Delta, \delta, \gamma$ but $P_{\text{reg}}$ works if $n = \alpha_s \Delta, \delta, \gamma + 1$ (if we consider $f = 1$). Let us now consider $(\Delta S, *)$ for $f > 1$. In this case, mobile agents move all together, thus the same $wrs_k$ scenario is reproduced $f$ times. For each $wrs_k$ scenario it is possible to build an impossibility run if $n = \alpha_s \Delta, \delta, \gamma + f$ i.e., $(\Delta S, *)$ is not enough to cope with $1$ Byzantine server, thus it is straightforward that $\alpha_s \Delta, \delta, \gamma + f$ non Byzantine servers are not enough to cope with $f$ Byzantine servers, the same scenario is reproduced $f$ times.

In the case of unsynchronized movements (ITB and ITU) we consider $\Delta = \min \{\Delta_1, \ldots, \Delta_f\}$. Each mobile agent generates a different read scenario, those scenario can be up to $f$ as we just stated, if $P_{\text{reg}}$ exists, those worst case scenarios are equivalent each other in terms of replicas. Since all the worst case scenarios are equivalent in terms of replicas, thus impossibility results holding for mobile agents moving together hold also for mobile agent moving in an uncoordinated way.

In [8], for $n \geq \alpha_s \Delta, \delta, \gamma + 1$ ($f \geq 1$), it has been presented a tight protocol $P_{\text{reg}}$ that solves the Regular Register problem whose bounds match the safe register lower bounds. Thus the next corollary follows.

Corollary 3: Let $\alpha_s n_{\text{cb}} \leq \alpha_s \Delta, \delta, \gamma + f$ be the impossibility results holding in the $(\Delta S, *)$ model for $f = 1$. All the Safe Register impossibility results hold also in the $(ITB, *)$ and $(ITU, *)$ models.

### IV. Upper Bounds

In this section, we present an overview of the optimal protocols that implement a SWMR Regular Register in a round-free synchronous system respectively for $(ITB, CAM)$ and $(ITB, CUM)$ instances of the proposed MBF model. The detailed description of the protocols and their correctness are presented in the Appendix B and C.

Following the same approach we used in [8] for the $(\Delta S, CAM)$ model, our solution is based on the following two key points: (1) we implement a maintenance() operation, in this case executed on demand; (2) we implement read() and write() operations following the classical quorum-based approach. The size of the quorum needed to carry on the operations, and consequently the total number of servers required by the computation, is dependent on the time to terminate the maintenance() operation, $\delta$ and $\Delta$ (see Table II).

The difference with respect $(\Delta S, CAM)$ model is that the time at which mobile agents move is unknown. Notice that each mobile $ma_i$ agent has its own $\Delta_i$. Since we do not have any other information we consider $\Delta = \min \{\Delta_1, \ldots, \Delta_f\}$. The maintenance() operation for $(ITB, CAM)$ model. This operation is executed by servers on demand (request-reply) when the oracle notifies them that are in a cured state. Notice that in the $(*, CAM)$ models servers know when a mobile agent leaves them, thus depending on such knowledge they execute different actions. In particular, if a server $s_i$ is not in a cured state then it does nothing, just replies to ECHO_REQ() messages. Otherwise, if a server $s_i$ is in a cured state it first cleans its local variables and broadcasts to other servers a request. Then, after $2\delta$ time units it removes values that may come from servers that were Byzantine before the maintenance() and updates its state by checking the number of occurrences of each value received from the other servers. Contrarily to the $(\Delta S, CAM)$ case, a cured server notifies to all servers that it was Byzantine in the previous $\delta$ time period.
This is done invoking the awareAll function that broadcasts a default value \( \perp \) after \( \delta \) time a server discovered to be in a cured state. This is done to prevent a cured server to collect “slow” replies coming from servers that were affected before the execution of the maintenance() operation. In this model, the curing time \( \gamma \leq 2\delta \).

The maintenance() operation for \((ITB,CUM)\) model. In this case servers are not aware of their failure state, thus they have to run such operation even if they are correct or cured. In addition, in the \((ITB,CUM)\) model, the moment at which mobile agents move is not known, thus as for the \((ITB,CAM)\) case, a request-reply pattern is used to implement the maintenance() operation. Such operation is executed by servers every \( 2\delta \) times. In this case, to prevent a cured server to collect “slow” replies coming from servers that were affected before the execution of the maintenance() operation, a server chooses a random number to associate to such particular maintenance() operation instance broadcast the \( \text{ECHO}_\text{REQ}() \) message and waits \( 2\delta \) before restarting the operation. When there is a value whose occurrence overcomes the \#\text{echo}_{CUM} \) threshold, such value is stored at the server side.

Notice that, contrarily to all the previous models, servers are not aware of their failure state and do not synchronize the maintenance() operation with each other. The first consequence is that a mobile agent may leave a cured server running such operation with garbage in server variables, making the operation unfruitful. Such server has to wait \( 2\delta \) to run again the maintenance() operation with clean variables, so that next time it will be effective, which implies \( \gamma \leq 4\delta \).

The write operation. To write a new value \( v \), the writer increments its sequence number \( csn \) and propagates \( v \) and \( csn \) to all servers via a WRITE messages. Then, it waits for \( \delta \) time units (the maximum message transfer delay) before returning. When a server \( s_i \) delivers a WRITE, it updates its local variables and sends a \( \text{REPLY} \) message to all clients that are currently reading to allow them to complete their read operation.

The read operation. When a client wants to read, it broadcasts a READ request to all servers and then waits \( 2\delta \) time \( (i.e., \) one round trip delay) to collect replies. When it is unblocked from the wait statement, it selects a value \( v \) occurring enough number of times \( (\geq \#\text{reply}_{CAM} \) from Table IV\) from the replies set, sends an acknowledgement message to servers to inform that its operation is now terminated and returns \( v \) as result of the operation. When a server \( s_i \) delivers a \( \text{READ}(j) \) message from client \( c_j \), it first puts its identifier in the pending read set to remember that \( c_j \) is reading and needs to receive possible concurrent updates and it sends a reply back to \( c_j \).

\( \text{P}_{\text{reg}} \) Detailed Description The protocol \( \text{P}_{\text{reg}} \) for the \((ITB,CUM)\) model is described in Figures 1 - 3, which present the maintenance(), write(), and read() operations, respectively. Table IV reports the parameters for the protocol. In particular, \( n_{CUM} \) is the bound on the number of servers, \#\text{reply}_{CUM} \) is minimum number of occurrences from different servers of a value to be accepted as a reply during a read() operation and \#\text{echo}_{CUM} \) is the minimum number of occurrences from different servers of a value to be accepted during the maintenance() operation.

Local variables at client \( c_i \). Each client \( c_i \) maintains a set \#\text{reply} that is used during the read() operation to collect the three tuples \( (j, (v,sn), i) \) sent back from servers. In particular \( v \) is the value, \( sn \) is the associated sequence number and \( j \) is the identifier of server \( s_j \) that sent the reply back. Additionally, \( c_i \) also maintains a local sequence number \( csn \) that is incremented each time it invokes a write() operation and is used to timestamp such operations monotonically.

Local variables at server \( s_i \). Each server \( s_i \) maintains the following local variables (we assume these variables are initialized to zero, false or empty sets according their type):

- \( V_v \) : an ordered set containing 3 tuples \( (v,sn) \), where \( v \) is a value and \( sn \) is the corresponding sequence number. Such tuples are ordered incrementally according to their \( sn \) values.
- \( V_{safe} \) : this set has the same characteristic as \( V_v \). The insert\( (V_{safe}, (v_k,sn_k)) \) function places the new value in \( V_{safe} \), according to the incremental order and if dimensions exceed 3 then it discards from \( V_{safe} \), the value associated to the lowest \( sn \).
- \( W_v \) : is the set where servers store values coming directly from the writer, associating to it a timer, \( (v,sn,timer) \). Values from this set are deleted when the timer expires or has a value non compliant with the protocol.
- \( \text{pending}\_\text{read} \) : set variable used to collect identifiers of the clients that are currently reading.
- \( \text{echo}_\text{vals} \) and \( \text{echo}_\text{read} \): two sets used to collect information propagated through \( \text{ECHO} \) messages. The first one stores tuple \( (j,(v,sn)) \) propagated by servers just

<table>
<thead>
<tr>
<th>( (CAM, ITB) )</th>
<th>( 2\delta \leq \Delta&lt;3\delta )</th>
<th>( n_{CAM} )</th>
<th>#\text{reply}_{CAM}</th>
<th>4f+1</th>
<th>2f+1</th>
<th>( \delta \leq \Delta&lt;2\delta )</th>
<th>( n_{CAM} )</th>
<th>#\text{reply}_{CAM}</th>
<th>6f+1</th>
<th>3f+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (CUM, ITB) )</td>
<td>( 2\delta \leq \Delta&lt;3\delta )</td>
<td>( n_{CUM} )</td>
<td>#\text{reply}_{CUM}</td>
<td>7f+1</td>
<td>4f+1</td>
<td>( \delta \leq \Delta&lt;2\delta )</td>
<td>( n_{CUM} )</td>
<td>#\text{reply}_{CUM}</td>
<td>12f+1</td>
<td>7f+1</td>
</tr>
</tbody>
</table>
after the mobile Byzantine agents moved, while the second stores the set of concurrently reading clients in order to notify cured servers and expedite termination of \text{read}()

- \textit{curing}: set used to collect servers running the \text{maintenance()} operation. Notice, to keep the code simple we do not explicitly manage how to empty such set since has not impact on safety properties.

In order to simplify the code of the algorithm, let us define the following functions:

- \text{select\_three\_pairs\_max\_sn(\text{echo\_vals}_i)}: this function takes as input the set \text{echo\_vals}_i and returns, if they exist, three tuples \(\langle v, sn \rangle\), such that there exist at least \#\text{echo\_CUM} occurrences in \text{echo\_vals}_i of such tuple. If more than three such tuples exist, the function returns the tuples with the highest sequence number.

- \text{select\_value(\text{reply}_j)}: this function takes as input the \text{reply}_j set of replies collected by client \(c_i\) and returns the pair \(\langle v, sn \rangle\) occurring occurring at least \#\text{reply\_CUM} times. If there are more pairs with the same occurrence, it returns the one with the highest sequence number.

- \text{conCut}(V_i, V_{safe}, W_i): this function takes as input three 3 dimension ordered sets and returns another 3 dimension ordered set. The returned set is composed by the concatenation of \(V_{safe} \circ V_i \circ W_i\), without duplicates, truncated after the first 3 newest values (with respect to the timestamp). e.g., \(V_i = \{\langle v_a, 1 \rangle, \langle v_b, 2 \rangle, \langle v_c, 3 \rangle, \langle v_d, 4 \rangle\}\) and \(V_{safe} = \{\langle v_e, 2 \rangle, \langle v_d, 4 \rangle, \langle v_f, 5 \rangle\}\) and \(W_i = \emptyset\), then the returned set is \(\langle \langle v_e, 3 \rangle, \langle v_d, 4 \rangle, \langle v_f, 5 \rangle \rangle\).

The \text{maintenance()} operation. Such operation is executed by servers every 26 times. Each time \(s_i\) resets its variables, except for \(W_i\) (that is continuously checked by the function \text{timerCheck}()) and the content of \(V_{safe}\), which overides the content of \(V_i\), before to be reset. Then \(s_i\) choses a random number to associate to such particular maintenance() operation instance \footnote{Is it out of the scope of this work to describe such function, we assume that Byzantine server can not predict the random number chosen next. The aim of such number is to prevent Byzantine servers to send reply to maintenance() operations before their invocation, or, in other words, it prevents correct servers to accept those replies.} broadcast the \text{ECHO\_REQ()} message and waits 2\(\delta\) before to restart the operation. In the meantime \text{ECHO()} messages are delivered and stored in the \text{echo\_vals}_i set. When there is value \(v\) whose occurrence overcomes the \#\text{echo\_CUM} threshold, such value is stored in \(V_{safe}\), and a \text{REPLY()} message with \(v\) is sent to current reader clients (if any).

Notice that, contrarily to all the previous models, servers are not aware about their failure state and do not synchronize the \text{maintenance()} operation with each other. The first consequence is that a mobile agent may leave a cured server running such operation with garbage in server variables, making the operation unfruitful. Such server has to wait 2\(\delta\) to run again the \text{maintenance()} operation with clean variables, so that next time it will be effective, which implies \(\gamma \leq 4\delta\).

\textbf{The write()} operation. When the writer wants to write a value \(v\), it increments its sequence number \(csn\) and propagates \(v\) to all servers. Then it waits for \(\delta\) time units (the maximum message transfer delay) before returning.

When a server \(s_i\) delivers a \text{WRITE} message, it updates \(W_i\), associating to such value a timer 4\(\delta\). 4\(\delta\) it is a consequence of the double \text{maintenance()} operation that a cured server has to run in order to be sure to be correct. Thus if a server is correct it keeps \(v\) in \(W_i\) during 4\(\delta\), which is enough for our purposes. On the other side a cured servers keeps a value (not necessarily coming from a \text{write}() operation) no more than the time it is in a cured state, 4\(\delta\), which is safe. After storing \(v\) in \(W_i\), such value is inserted in \text{reply()} message to all clients that are currently reading (clients in \text{pending\_read}_i) to notify them about the concurrent \text{write()} operation and to any server executing the \text{maintenance()} operation (servers in \text{curing}).

\textbf{The read()} operation. When a client wants to read, it broadcasts a \text{READ} request to all servers and waits 2\(\delta\) time (i.e., one round trip delay) to collect replies. When it is unblocked from the wait statement, it selects a value \(v\) invoking the \text{select\_value} function on \text{reply}_j set, sends an acknowledgement message to servers to inform that its operation is now terminated and returns \(v\) as result of the operation.

When a server \(s_i\) delivers a \text{READ} message from client \(c_j\) it first puts its identifier in the set \text{pending\_read}_i to remember that \(c_j\) is reading and needs to receive possible concurrent updates, then \(s_i\) sends a reply back to \(c_j\). Note that, in the \text{REPLY()} message is carried the result of \text{conCut}(V_i, V_{safe}, W_i). In this case, if the server is correct then \(V_i\) contains valid values, and \(V_{safe}\) contains valid values by construction, since it comes from values sent during the current \text{maintenance()} operation. If the server is cured, then \(V_i\) and \(W_i\) may contain any value. Finally, \(s_i\) forwards a \text{READ\_FW} message to inform other servers about \(c_j\) read request. This is useful in case some server missed the \text{READ} message as it was affected by mobile Byzantine agent when such message has been delivered.

When a \text{READ\_ACK} message is delivered, \(c_j\) identifier is removed from both \text{pending\_read}_i set as it does not need anymore to receive updates for the current \text{read()} operation.

\section{Concluding remarks}

We proposed lower bounds and matching upper bounds for the emulation of a regular register in the round free synchronous communication model under \textit{unsynchronized}
moves of Byzantine agents. The computed lower bounds are significantly higher than those computed for synchronized Byzantine agents model. Investigating other classical problems in the same fault model is a challenging path for future research.

REFERENCES


[10] J. A. Garay. Reaching (and maintaining) agreement in the presence


APPENDIX A: LOWER BOUNDS

In this Appendix we prove lower bounds with respect to the minimum fraction of correct servers to implement safe registers in presence of mobile Byzantine failures. In particular we first prove lower bounds for the \((\Delta S, CAM)\) and \((\Delta S, CUM)\) models and then we extend those results to \((ITB, CAM)\) and \((ITB, CUM)\) models. The first observation that raises is that in presence of mobile agents in the round-free models there are several parameters to take into account with respect to the round-based model. Let us start considering that the set of Byzantine servers changes its composition dynamically over time. This yields to the following question: does it impact on the read() duration? Or, in other words, such operation has to last as less as possible or until it eventually terminates? In this chapter we consider the read() operation duration as a parameter itself, allowing us to easily verify when the variation of such parameter has any impact on lower bounds. Here below the list of parameters we take into account.

- servers knowledge about their failures state \((CAM, CUM)\);
- the relationship between \(\delta\) and \(\Delta\) (that states how many Byzantine servers there may be during an operation);
- \(T_r\), the read() operations duration;
- \(\gamma\), the upper bound on the time during which a server can be in a cured state (the design of an optimal maintenance() operation is out of the scope of this thesis, thus we use such upper bound as another parameter).

Those parameters allow us to describe different failure models and help us to provide a general framework that produces lower bounds for each specific instance of the MBF models. In the sequel it will be clear that \(\gamma\) varies depending on the coordinated/uncoordinated mobile agents movements \((\Delta S, ITB, ITU)\). In other words, in this parameter is hidden the movements model taken into account, so we do not need to explicitly parametrize it. Before to start let us precise that we do not consider the following algorithm families: (i) full information algorithm families (processes exchange information at each time instant); (ii) algorithms characterized by a read operation that does not require a request-reply pattern; (iii) algorithms with non quiescent operation (the message exchange triggered by an operation eventually terminates); and finally (iv) algorithms where clients interact with each other. All results presented in the sequel consider a families of algorithms such that previous characteristics do not hold. The lower bounds proof leverages on the classical construction of two indistinguishable executions. The tricky part is to characterize the set of messages delivered by a client from correct and incorrect servers depending of the read() operation duration. Let \(T_r\), \(T_r \geq 2\delta\) be such duration, each read() operation requires at least a request-reply pattern. We first characterize the correct and incorrect sets of messages, delivered during \(T_r\) time, with respect to \(\Delta\) and \(\gamma\). For clarity, in the sequel we note correct message/request/reply a message that carries a valid

---

**Fig. 3.** An algorithm implementing the read() operation in the \((ITB, CUM)\) model.

---

\[ \text{operation read(\{} \]
\[ (01) \text{reply}_y \leftarrow \emptyset; \]
\[ (02) \text{broadcast read}(i); \]
\[ (03) \text{wait (2\delta);} \]
\[ (04) (v, sn) \leftarrow \text{select value}(\text{reply}_y); \]
\[ (05) \text{broadcast read}_\text{ACK}(i); \]
\[ (06) \text{return } v; \]

when \(\text{reply}(j, v_j)\) is received:
\[ (07) \text{for each } (v, sn) \in V_j, \text{do} \]
\[ (08) \text{reply}_y \leftarrow \text{reply}_y \cup \{ (j, (v, sn))\}; \]

---

**Server code**

when \(\text{read}_\text{FW}(j)\) is received:
\[ (10) \text{pending}_\text{read}_\text{fw} \leftarrow \text{pending}_\text{read}_\text{fw} \cup \{ j \}; \]
\[ (11) \text{send reply}_i (j, \text{conCut}(V_i, \text{safe}_i, W_i)); \]
\[ (12) \text{broadcast read}_\text{FW}(j); \]

when \(\text{read}_\text{ACK}(j)\) is received:
\[ (14) \text{pending}_\text{read}_\text{fw} \leftarrow \text{pending}_\text{read}_\text{fw} \setminus \{ j \}; \]
\[ (15) \text{echo}_\text{read} \leftarrow \text{echo}_\text{read} \setminus \{ j \}; \]
value when it is sent (i.e., sent by a correct process). Otherwise, the message is incorrect. It has been proven [8] that a protocol \( \mathcal{P}_\text{reg} \) implementing a regular register in a mobile Byzantine setting must include in addition to the mandatory read and write operations an additional operation, maintenance, defined below.

**Definition 19 (Maintenance operation maintenance):** A maintenance operation is an operation that, when executed by a process \( p_i \), terminates at some time \( t \) leaving \( p_i \) with a valid state at time \( t \) (i.e., it guarantees that \( p_i \) is correct at time \( t \)).

Such operation has a direct impact on the number of correct processes in any time instant. For that reason it is important to characterize its duration, in particular its upper bound in terms of time. The following definition defines \( \gamma \), the upper bound of the time during which a server can be in a cured state.

**Definition 20 (Curing time, \( \gamma \)):** We define \( \gamma \) as the maximum time a server can be in a cured state. More formally, let \( T_c \) be the time at which server \( s_i \) is left by a mobile agent, let \( \text{op}_\text{M} \) the first maintenance operation that correctly terminates, then \( t_E(\text{op}_\text{M}) - T_c \leq \gamma \).

In order to build our indistinguishable execution, we define below a scenario of agents movement. Then, with respect this scenario, we construct two indistinguishable executions.

**Definition 21 (Scenario \( S^* \)):** Let \( S^* \) be the following scenario: for each time \( T_i, i \geq 0 \) the affected servers are \( s_{(i \mod n)} f + 1, \ldots, s_{(i \mod n)} f + f \).

In Figure 4 is depicted \( S^* \). In particular, the red part is the time where \( f \) agents are affecting \( f \) servers and the gray part is the time servers are running the maintenance operation.

Let us characterize the \( \mathcal{P}_\text{reg} \) protocol in the most general possible way. By definition a register abstraction involves read() and write() operations issued by clients. A read operation involves at least a request – reply communication pattern (i.e., two communication steps). Thus, given the system synchrony, a read() operation \( \text{op}_R \) lasts at least \( T_r \geq 2\delta \) time. Moreover we consider that a correct server sends a reply message in two occasions: (i) after the delivery of a request message, and (ii) right after it changes its state, at the end of the maintenance operation if an \( \text{op}_R \) is occurring. The latter case exploits the maintenance operation allowing servers to reply with a valid value in case they were Byzantine at the beginning of the read operation. Moreover we assume that in \((*, \text{CAM})\) model servers in a cured state do not participate to the read operation. Notice that those servers are aware of their current cured state and are aware of their impossibility to send correct replies. Even though those may seems not very general assumptions, let us just consider that we are allowing servers to correctly contribute to the computation as soon as they can and stay silent when they can not and under those assumptions we prove lower bounds. Thus if we remove those assumptions the lower bounds do not decreases. Scenario and protocol has been characterized.

Now we aim to characterize the set of servers, regarding their failure states, that can appear during the execution of the protocol, in particular during the read() operation. Those sets allow us to characterize correct and incorrect messages that a client delivers during a read() operation.

**Definition 22 (Failure State of servers in a time interval):** Let \( [t, t + T_r] \) be a time interval and let \( t', t' > 0 \), be a time instant. Let \( s_i \) be a server and \( \text{state}_i \) be \( s_i \) state, \( \text{state}_i \in \{\text{correct, cured, Byzantine}\} \). Let \( S(t') \) be the set of servers \( s_i \) that are in the state \( \text{state}_i \) at \( t' \), \( S(t') \in \{\text{Co}(t'), \text{Cu}(t'), B(t')\} \). \( S(t, t + T_r) \) is the set of servers that have been in the state \( \text{state}_i \) for at least one time unit during \( [t, t + T_r] \). More formally, \( S(t, t + T_r) = \bigcup_{t' \leq T_r} S(t') \).

**Definition 23 (CBC \((t, t + T_r)\)):** Let \( [t, t + T_r] \) be a time interval, \( \text{CBC}(t, t + T_r) \) denotes servers that during a time interval \( [t, t + T_r] \) belong first to \( B(t, t + T_r) \) or \( \text{Cu}(t) \) (only in \((\Delta S, \text{CU}M)\) model) and then to \( \text{Co}(t + \delta, t + T_r - \delta) \) or vice versa.

In particular let us denote:

- \( \text{BC}(t, t + T_r) \) servers that during a time interval \( [t, t + T_r] \) belong to \( B(t, t + T_r) \) or \( \text{Cu}(t) \) (only in \((\Delta S, \text{CU}M)\) model) and to \( \text{Co}(t + \delta, t + T_r - \delta) \).
- \( \text{BC}(t, t + T_r) \) servers that during a time interval \( [t, t + T_r] \) belong to \( \text{Co}(t + \delta, t + T_r - \delta) \) and to \( B(t, t + T_r) \).

**Definition 24 (Sil\((t, t + T_r)\)):** Let \( [t, t + T_r] \) be a time interval. \( \text{Sil}(t, t + T_r) \) is the set of servers in \( \text{Cu}(t, t + T_r - \delta) \).

Servers belonging to \( \text{Sil}(t_B(\text{op}_R), t_E(\text{op}_R)) \) are servers that do no participate to \( \text{op}_R \). In oder words, those servers in the worst case scenario became correct after \( t_E(\text{op}_R) - \delta \), thus if they send back a correct reply it is not sure that client delivers such reply before the end of \( T_r \) time. Now we can define the worst case scenarios for the sets we defined so far with respect to \( S^* \).

**Definition 25 (Max\( \bar{B}(t, t + T_r)\)):** Let \( S \) be a scenario and \( [t, t + T_r] \) a time interval. The cardinality of \( \bar{B}_S(t, t + T_r) \) is maximum with respect to \( S \) if for any \( t' > 0 \), we have that \( |\bar{B}_S(t, t + T_r)| \geq |\bar{B}_S(t', t' + T_r)| \). Then we call the value of such cardinality as \( \text{Max}\( \bar{B}_S(t, t + T_r)\)\). If we consider only one scenario per time then we can omit the subscript related to the scenario and write directly \( \text{Max}\( \bar{B}(t, t + T_r)\)\).

This value quantifies in the worst case scenario how many servers can be Byzantine, for at least one time unit, during a read() operation. Figure 5 depicts a scenario where \( T_r = 3\delta \) and during the time interval \( [t', t' + T_r] \) there is a maximum number of Byzantine servers while in \( [t'', t'' + T_r] \) this number is not maximal.

**Definition 26 (MaxSil\((t, t + T_r)\)):** Let \( S \) be a scenario and \( [t, t + T_r] \) a time interval. The cardinality of \( \text{Sil}_S(t, t + T_r) \) is maximum with respect to \( S \) if for any \( t' > 0 \) we have that \( |\text{Sil}(t, t + T_r)| \geq |\text{Sil}(t', t' + T_r)| \) and \( \bar{B}(t, t + T_r) = \text{Max}\( \bar{B}(t, t + T_r)\)\). Then we call the value of such cardinality as \( \text{MaxSil}_S(t, t + T_r) \). If we consider only one scenario per time then we can omit the subscript related to the scenario and write directly \( \text{minSil}(t, t + T_r) \).

This value quantifies the maximum number of servers that begin in a cured state a read() operation and are still cured after \( T_r - \delta \) time. So that any correct reply sent after such period has no guarantees to be delivered by the client and such servers are assumed to be silent.
Definition 27 ($\text{MaxCu}(t)$): Let $S$ be a scenario and $t$ be a time instant. The cardinality of $\text{Cu}_S(t)$ is maximum with respect to $S$ if for any $t', t' \geq 0$, we have that $|\text{Cu}_S(t')| \leq |\text{Cu}_S(t)|$ and $\text{B}(t, t + \tau_r) = \text{MaxB}(t, t + \tau_r)$. We call the value of such cardinality as $\text{MaxCu}(t)$. If we consider only one scenario per time then we can omit the subscript related to it and write directly $\text{MaxCu}(t)$.

This value quantifies, in the worst case scenario, how many cured servers there may be at the beginning of a read() operation. Figure 5 depicts a scenario where at time $t'$ there are the maximum number of cured server while at $t''$ this value is not maximum. Notice that in such figure, in case of a shorter time interval $[t', t' + 2\delta]$ $s_0$ would be silent.

Definition 28 ($\text{minCo}(t, t + \tau_r)$): Let $S$ be a scenario and $[t, t + \tau_r]$ be a time interval then $\text{minCo}(t, t + \tau_r)$ denotes the minimum number of correct servers during a time interval $[t + \delta, t + \tau_r - \delta]$. If we consider only one scenario per time then we can omit the subscript related to it and write directly $\text{minCo}(t, t + \tau_r)$.

Figure 7 depicts a scenario where during the both intervals $[t', t' + \tau_r]$ and $[t'', t'' + \tau_r]$ the number of correct servers is minimum.

Definition 29 ($\text{minCBC}(t, t + \tau_r)$): Let $[t, t + \tau_r]$ be a time interval then $\text{minCBC}(t, t + \tau_r)$ denotes the minimum number of servers that during a time interval $[t, t + \tau_r]$ belong first to $\text{B}(t, t + \tau_r)$ or $\text{Cu}(t)$ (only in $\Delta S, \text{CUM}$ model) and then to $\text{Co}(t + \delta, t + \tau_r - \delta)$ or vice versa and $\text{B}(t, t + \tau_r) = \text{MaxB}(t, t + \tau_r)$.

In particular let us denote as:

- $\text{minBC}(t, t + \tau_r)$ the minimum number of servers that during a time interval $[t, t + \tau_r]$ belong to $\text{B}(t, t + \tau_r)$ or $\text{Cu}(t)$ (only in $\Delta S, \text{CUM}$ model) and to $\text{Co}(t + \delta, t + \tau_r - \delta)$.
- $\text{minCB}(t, t + \tau_r)$ the minimum number of servers that during a time interval $[t, t + \tau_r]$ belong to $\text{Co}(t + \delta, t + \tau_r - \delta)$ and to $\text{B}(t, t + \tau_r)$.

As we stated before, Byzantine servers set changes during the read() operation $\text{op}_R$, so there can be servers that are in a Byzantine state at $t_B(\text{op}_R)$ and in a correct state before $t_F(\text{op}_R) - \delta$ (cf. case $s_0$ during $[t', t' + 3\delta]$ time interval in Figure 6). Those servers contribute with an incorrect message at the beginning and with a correct message after. The same may
happen with servers that are correct from $t_B(op_R)$ to at least $t_B(op_R) + \delta$ (so that for sure deliver the read request message and send the reply back) and are affected by a mobile agent after $t_B(op_R) + \delta$ (cf. $s_0$ during $[t', t' + 3\delta]$ time interval in Figure 8).

Lemma 3: \( \text{Max} \tilde{B}(t, t + T_r) = (\lfloor \frac{T_r}{\Delta} \rfloor + 1) f. \)

Proof: For simplicity let us consider a single agent \( ma_k \), then we extend the same reasoning to all the \( f \) agents. In \( [t, t + T_r] \) time interval, with \( T_r \geq 2\delta \), \( ma_k \) can affect a different server each \( \Delta \) time. It follows that the number of times it may change server is \( \frac{T_r}{\Delta} \). Thus the affected servers are \( \lfloor \frac{T_r}{\Delta} \rfloor \) plus the server that was affected at \( t \). Finally, extending the reasoning to \( f \) agents, \( \text{Max} \tilde{B}(t, t + T_r) = (\lfloor \frac{T_r}{\Delta} \rfloor + 1) f \), which concludes the proof. \(
\)

As we see in the sequel, the value of \( \text{Max} \tilde{B}(t, t + T_r) \) is enough to compute the lower bound. Now we can define the worst case scenario for a read() operation with respect to \( S^* \).

Let \( op \) be a read operation issued by \( c_i \). We want to define, among the messages that can be deliver by \( c_i \) during \( op \), the minimum amount of messages sent by server when they are in a correct state and the maximum amount of messages sent by servers when they are not in a correct state.

In each scenario, we assume that each message sent to or by Byzantine servers is instantaneously delivered, while each message sent to or by correct servers requires \( \delta \) time. Without loss of generality, let us assume that all Byzantine servers send the same value and send it only once, for each period where they are Byzantine. Moreover, we make the assumption that each cured server (in the CAM model) does not reply as long as it is cured. Yet, in the CUM model, it behaves similarly to Byzantine servers, with the same assumptions on message delivery time.

Definition 30 (MaxReplies_NCo(t, t + T_r)k): Let \( \text{MaxReplies}_NCo(t, t + T_r)k \) be the multi-set maintained by client \( c_k \) containing \( m_{ij} \) elements, where \( m_{ij} \) is the \( i \)-th message delivered by \( c_k \) and sent at time \( t', t' \in [t, t + T_r] \) by \( s_j \) such that \( s_j \notin Co(t') \).

Considering the definitions of both \( \text{Max} \tilde{B}(t, t + T_r) \) and \( \text{Max} Cu(t) \) the next Corollary follows:

Corollary 4: In the worst case scenario, during a read operation lasting \( T_r \geq 2\delta \) issued by client \( c_i \), \( c_i \) delivers \( \text{Max} \tilde{B}(t, t + T_r) \) incorrect replies in the \( (\Delta S, CAM) \) model and \( \text{Max} \tilde{B}(t, t + T_r) + \text{Max} Cu(t) \) incorrect replies in the \( (\Delta S, CUM) \) model.

Definition 31 (MinReplies_Co(t, t + T_r)k): Let \( \text{MinReplies}_C(t, t + T_r)k \) be the multi-set maintained by client \( c_k \) containing \( m_{ij} \) elements, where \( m_{ij} \) is the \( i \)-th message delivered by \( c_k \) and sent at time \( t', t' \in [t, t + T_r] \) by \( s_j \) such that \( s_j \notin Co(t') \).

Note that correct replies come from servers that \( (i) \) have never been affected during the time interval \( [t, t + T_r] \), or \( (ii) \) where in a cured state at \( t \) but do not belong to the \( S_0l(t, t + T_r) \) set, or \( (iii) \) servers that reply both correctly and incorrectly.

The next Corollary follows:

Corollary 5: In the worst case scenario, during a read operation lasting \( T_r \geq 2\delta \) issued by client \( c_i \), \( c_i \) delivers \( n - (\text{Max} \tilde{B}(t, t + T_r) + \text{Max} Su(t, t + T_r)) + \text{Min} BC(t, t + T_r) \) correct replies in the \( (\Delta S, CAM) \) model and \( n - (\text{Max} \tilde{B}(t, t + T_r) + \text{Max} Cu(t)) + \text{Min} BC(t, t + T_r) \) correct replies in the \( (\Delta S, CUM) \) model.

In the following, given a time interval, we characterize correct and incorrect servers involved in such interval. Concerning correct servers, let us first analyze when a client collects \( x \leq n \) different replies and then we extend such result to \( x > n \). Then we do the same for incorrect replies.

Lemma 4: Let \( op \) be a read operation issued by client \( c_i \) in a scenario \( S^* \), whose duration is \( T_r \geq 2\delta \). Let \( x, x \geq 2 \), be the number of messages delivered by \( c_i \) during \( op \). If \( x \leq n \) then \( \text{MinReplies}_C(t, t + T_r)k \) contains replies from \( x \) different servers.

Proof: Let us suppose that \( \text{MinReplies}_C(t, t + T_r)k \) contains replies from \( x - 1 \) different servers (trivially it can not be greater than \( x \)). Without loss of generality, let us suppose that \( c_i \) collects replies from \( s_1, \ldots, s_{x-1} \) that replied twice and a server \( s_x \) that did not replied. Let us also suppose w.l.g. that there is one Byzantine mobile agent \( ma_k \) (i.e., \( f = 1 \)). If during the time interval \( [t, t + T_r] \) \( s_x \) never replied, then \( s_x \) has been affected at least during \( [t + \delta, t + \Delta - \gamma + 1] \). This implies that \( T_r \leq \Delta + 2\delta + \gamma \).

If a client delivers \( n > x \) messages then we can apply the same reasoning of the previous Lemma to the first chunk of \( n \) messages, then to the second chunk of \( n \) messages and so on. Roughly speaking, if \( n = 5 \) and a client delivers 11 messages...
from correct processes, then there are 3 occurrences of the messages coming from the first server and 2 occurrences of the messages coming from the remaining servers. Thus the next Corollary directly follows.

**Corollary 6:** Let op be a read operation issued by client \(c_i\) in a scenario \(S^*\), op duration is \(T_r \geq 2\delta\). Let \(x, x \geq 2\), be the number of messages delivered by \(c_i\) during op, then \(\text{maxReplies}_\text{Co}(t, t+T_r)_k\) contains \(x \mod n\) messages \(m_{ij}\) whose occurrences is \(\left\lfloor \frac{x}{n} \right\rfloor + 1\) and \((n - x \mod n)\) messages whose occurrences is \(\left\lfloor \frac{x}{n} \right\rfloor\).

The case of \(\text{maxReplies}_\text{NCo}(t, t+T_r)_k\) directly follows from scenario \(S^*\), since by hypotheses mobile Byzantine agents move circularly from servers to servers, never passing on the same server before having affected all the others. Thus, the following corollary holds.

**Corollary 7:** Let op be a read operation issued by client \(c_i\) in a scenario \(S^*\), op duration is \(T_r \geq 2\delta\). Let \(x, x \geq 2\), be the number of messages delivered by \(c_i\) during op, then \(\text{maxReplies}_\text{NCo}(t, t+T_r)_k\) contains \(x \mod n\) messages \(m_{ij}\) whose occurrences is \(\left\lfloor \frac{x}{n} \right\rfloor + 1\) and \((n - x \mod n)\) messages whose occurrences is \(\left\lfloor \frac{x}{n} \right\rfloor\).

At this point we can compute how many correct and incorrect replies a client \(c_k\) can deliver in the worst case scenario during a time interval \([t, t+T_r]\). Trivially, \(c_k\) is in order to distinguish correct and incorrect replies needs to get \(\text{minReplies}_\text{Co}(t, t+T_r)_k > \text{maxReplies}_\text{NCo}(t, t+T_r)_k\). It follows that the number of correct servers has to be enough to guarantee this condition. Table VI follows directly from this observation.

**Theorem 2:** If \(n < n_{\text{CAM}, B} (n < n_{\text{Cum}, B})\) as defined in Table VI then there not exists a protocol \(\text{Pr}_{\text{reg}}\) solving the safe register specification in \((\Delta S, \text{CAM})\) model ((\(\Delta S, \text{CUM})\) model respectively).

**Proof** Let us suppose that \(n < n_{\text{CAM}, B} (n < n_{\text{Cum}, B})\) and that protocol \(\text{Pr}_{\text{reg}}\) does exist. If client \(c_i\) invokes a read operation op, lasting \(T_r \geq 2\delta\) time, if no write operations occur, then \(c_i\) returns a valid value at time \(t_B(op)\). Let us consider an execution \(E_0\) where \(c_i\) invokes a read operation op and let 0 be the valid value at \(t_B(op)\). Let us assume that all Byzantine servers involved in such operation reply once with 1. From Corollaries 4 and 5 \(c_i\) collects \(\text{maxReplies}_\text{NCo}(t, t+T_r)_i\) occurrences of 1 and \(\text{minReplies}_\text{Co}(t, t+T_r)_i\) occurrences of 0. Since \(\text{Pr}_{\text{reg}}\) exists and no write operations occur, then \(c_i\) returns 0. Let us now consider a another execution \(E_1\) where \(c_i\) invokes a read operation op and let 1 be the valid value at \(t_B(op)\). Let us assume that all Byzantine servers involved in such operation replies once with 0. From Corollaries 4 and 5 and Corollary 6 \(c_i\) collects \(\text{maxReplies}_\text{NCo}(t, t+T_r)_i\) occurrences of 0 and \(\text{minReplies}_\text{Co}(t, t+T_r)_i\) occurrences of 1. Since \(\text{Pr}_{\text{reg}}\) exists and no write operations occur, then \(c_i\) returns 1.

From Lemma 3 and using values in Table VI we obtain following equations for both models:

- \((\Delta S, \text{CAM})\):
  
  \[
  \text{maxReplies}_\text{NCo}(t, t+T_r)_i = \text{max}\tilde{B}(t, t+T_r) = (\left\lceil \frac{T_r}{\Delta} \right\rceil + 1)f
  \]
  
  \[
  \text{minReplies}_\text{Co}(t, t+T_r)_i = n - \text{max}\tilde{B}(t, t+T_r) + \text{MaxSil}(t, t+T_r) + \text{minCBC}(t, t+T_r) = [2(\text{Max}\tilde{B}(t, t+T_r)) + \text{MaxSil}(t, t+T_r) - \text{minCBC}(t, t+T_r)]
  \]
  
  \[
  \text{Max}\tilde{B}(t, t+T_r) = \text{minCBC}(t, t+T_r)
  \]

- \((\Delta S, \text{CUM})\):
  
  \[
  \text{maxReplies}_\text{NCo}(t, t+T_r)_i = \text{max}\tilde{B}(t, t+T_r) + \text{MaxCu}(t) = (\left\lceil \frac{T_r}{\Delta} \right\rceil + 1)f + \text{MaxCu}(t)
  \]
  
  \[
  \text{minReplies}_\text{Co}(t, t+T_r)_i = n - \text{max}\tilde{B}(t, t+T_r) + \text{MaxCu}(t) + \text{minCBC}(t, t+T_r) = [2\text{Max}\tilde{B}(t, t+T_r) + 2\text{MaxCu}(t) - \text{minCBC}(t, t+T_r)] + [\text{Max}\tilde{B}(t, t+T_r) + \text{MaxCu}(t) + \text{minCBC}(t, t+T_r)]
  \]
  
  \[
  \text{Max}\tilde{B}(t, t+T_r) + \text{MaxCu}(t) = (\left\lceil \frac{T_r}{\Delta} \right\rceil + 1)f + \text{MaxCu}(t)
  \]

It follows that in \(E_0\) and \(E_1\) \(c_i\) delivers the same occurrences of 0 and 1, both executions are indistinguishable leading to a contradiction.

**Corollary 8:** For each \(T_r \geq 2\delta\) if \(n > n_{\text{CAM}, B}\) \(n > n_{\text{Cum}, B}\) then \(\text{maxReplies}_\text{NCo}(t, t+T_r)_i < \text{minReplies}_\text{Co}(t, t+T_r)_i\).

At this point we compute \(\text{minCu}(t), \text{MaxSil}(t, t+T_r)\) and \(\text{minCBC}(t, t+T_r)\) to finally state exact lower bounds depending on the system parameters, in particular depending on \(\Delta, \gamma\) and the servers awareness, i.e., \((\Delta S, \text{CAM})\) and \((\Delta S, \text{CUM})\).

Let us adopt the following notation. Given the time interval \([t, t+T_r]\) let \(\{s_1, s_2, \ldots, s_b\} \in B(t, t+T_r)\) be the servers affected sequentially during \(T_r\) by the mobile agent \(m_{ak}\).

Let \(\{s_{-1}, s_{-2}, \ldots, s_{-c}\} \in C(t)\) be the servers in a cured state at time \(t\) such that \(s_{-1}\) is the last server that entered in such state and \(s_1\) the first server that became cured. Let \(t_B(s_i)\) and \(t_E(s_i)\) be respectively the time instant in which \(s_i\) become Byzantine and the time in which the Byzantine agent left. \(t_BCu(s_i)\) and \(t_ECu(s_i)\) are respectively the time instant in which \(s_i\) became cured and the time instant in which it became correct. Considering that \(m_{ak}\) moves each \(\Delta\) time then we have that \(t_B(s_{-1}) - t_B(s_i) = \Delta\) and \(t_ECu(s_{-j}) - t_BCu(s_{-j+1}) = \Delta\). The same holds for the \(t_E\)
As we defined, model $\text{MaxCu}$ in a cured state, with respect to the considered time interval. Thus given an arbitrarily small number $\epsilon > 0$, let $t = t_E B(s_1) - \epsilon$. By construction we know that $t_E B(s_1) = t_E B(s_1) - \Delta = t_E B(s_1)$. Substituting $t_E B(s_1) = t + \epsilon - \Delta$, since we consider $\gamma$ the upper bound for the curing time, then $t_E B(s_1) = t + \epsilon - \Delta + \gamma$. So finally, $\text{MaxCu}(t) = \left\lfloor \frac{t_E B(s_1) - t}{\Delta} \right\rfloor = \left\lfloor \frac{\gamma - \Delta + \epsilon}{\Delta} \right\rfloor$ and since there can be no a negative result then $\text{MaxCu}(t) = R\left(\frac{\gamma - \Delta + \epsilon}{\Delta}\right)$. This concludes the proof.

Lemma 6: Let us consider a time interval $[t, t+T_r]$, $T_r > 2\delta$ and an arbitrarily small number $\epsilon > 0$, then in the $(\Delta S, CUB)$ model $\text{MaxCu}(t) = R\left(\frac{T_r - \epsilon - \frac{\epsilon}{\Delta} - \gamma}{\Delta}\right)$. 

Proof As we defined, $s_{-1}$ is the most recent server that entered in a cured state, with respect to the considered interval. Intuitively, $s_{-j}$ is in $\text{Cu}(t)$ if $t_E B(s_{-j}) > t$. Considering that $t_E B(s_{-j}) - t_E B(s_{-j-1}) = \Delta$ then the number of servers in a cured state at $t$ is $\text{MaxCu}(t) = \left\lfloor \frac{t_E B(s_1) - t}{\Delta} \right\rfloor$. As we stated, for $(\star, CAM)$ models we consider scenarios in which $t$, the beginning of the considered time interval, is just before $t_E B(s_1)$. Thus given an arbitrarily small number $\epsilon > 0$, let $t = t_E B(s_1) - \epsilon$. By construction we know that $t_E B(s_1) = t_E B(s_1) - \Delta = t_E B(s_1)$. Substituting $t_E B(s_{-j}) = t + \epsilon - \Delta$, since we consider $\gamma$ the upper bound for the curing time, then $t_E B(s_{-j}) = t + \epsilon - \Delta + \gamma$. So finally, $\text{MaxCu}(t) = \left\lfloor \frac{t_E B(s_1) - t}{\Delta} \right\rfloor = \left\lfloor \frac{\gamma - \Delta + \epsilon}{\Delta} \right\rfloor$ and since there can be no a negative result then $\text{MaxCu}(t) = R\left(\frac{\gamma - \Delta + \epsilon}{\Delta}\right)$. This concludes the proof.

Lemma 7: Let us consider a time interval $[t, t+T_r]$, $T_r > 2\delta$ and an arbitrarily small number $\epsilon > 0$, then in the $(\Delta S, CUM)$ model $\text{MaxSil}(t, t + T_r) = R\left(\frac{\gamma - \Delta + \epsilon - T_r + \delta}{\Delta}\right)$. 

Proof As we defined, $s_{-1}$ is the most recent server that entered in a cured state, with respect to the considered interval. Intuitively, $s_{-j}$ is in $\text{Sil}(t, t + \delta)$ if $t_E B(s_{-j}) > T_r - \delta$. Considering that $t_E B(s_{-j}) - t_E B(s_{-j-1}) = \Delta$ then the number of servers in a silent state at $t$ is $\text{MaxSil}(t, t + \delta) = \left\lfloor \frac{t_E B(s_1) - T_r + \delta}{\Delta} \right\rfloor$. As we stated for $(\Delta S, CAM)$ models we consider scenarios in which $t$, the beginning of the considered time interval, is just before $t_E B(s_1)$. Thus given an arbitrarily small number $\epsilon > 0$, let $t = t_E B(s_1) - \epsilon$. By construction we know that $t_E B(s_1) = t_E B(s_1) - \epsilon$. By construction we know that $t_E B(s_1) = t_E B(s_1) - \Delta = t_E B(s_1)$.
Substituting $t_ECu(s_{-1}) = t + \epsilon - \Delta$, since we consider $\gamma$ the upper bound for curieing time, then $t_ECu(s_{-1}) = t + \epsilon - \Delta + \gamma$. So finally, $MaxSil(t, t + T_r) = \lceil t_ECu(s_{-1}) - T_r + \frac{\Delta}{\gamma} \rceil = \lceil \frac{\Delta}{\gamma} + T_r + \frac{\Delta}{\gamma} \rceil$, since there can not be a negative result $MaxSil(t, t + 2\delta) = \mathcal{R}(\lceil \frac{\Delta}{\gamma} + T_r + \frac{\Delta}{\gamma} \rceil)$. □

**Lemma 8:** Let us consider a time interval $[t, t + T_r]$, $T_r \geq 2\delta$ and an arbitrarily small number $\epsilon > 0$, then in the $(\Delta S, CUM)$ model $MaxSil(t, t + T_r) = \lceil t_ECu(s_{-1}) - T_r + \frac{\Delta}{\gamma} \rceil$. Thus we are interested in servers that correctly reply after in time such that $t + T_r - \Delta$.

Proof As we defined, $s_{-1}$ is the most recent server that entered in a cured state, with respect to the considered interval. Intuitively, $s_{-1}$ is in $Sil(t, t + T_r)$ if $t_ECu(s_{-1}) > T_r - \Delta$. Considering that $t_ECu(s_{-1}) - t_ECu(s_{-1} - 1) = \Delta$ then the number of servers in a silent state at $t$ is $MaxSil(t, t + T_r) = \lceil t_ECu(s_{-1} - 1) - T_r + \frac{\Delta}{\gamma} \rceil$. As we stated for $(\Delta S, CUM)$ models we consider scenarios in which $t + T_r$, the end of the considered time interval, is just after $t_EB(s_{-1})$. Thus given an arbitrarily small number $\epsilon > 0$, let $t_EB(s_{-1}) = t + T_r - \epsilon$. By construction we know that $t_EB(s_{-1}) = t_EB(s_{-1} - 1) + \gamma$. Finally $MaxSil(t, t + T_r) = \lceil t_ECu(s_{-1}) - T_r + \frac{\Delta}{\gamma} \rceil = \lceil \frac{\Delta}{\gamma} + T_r + \frac{\Delta}{\gamma} \rceil$, since there can not be a negative result, $MaxSil(t, t + T_r) = \lceil \frac{\Delta}{\gamma} + T_r + \frac{\Delta}{\gamma} \rceil$. □

**Lemma 9:** Let us consider a time interval $[t, t + T_r]$, $T_r \geq 2\delta$ then in the $(\Delta S, CAM)$ model. $minCB = \mathcal{R}(\lceil \frac{\Delta}{\gamma} \rceil)$ + $\mathcal{R}(\lceil \frac{\Delta}{\gamma} - 2\delta \rceil)$.

Proof By definition $minCB = minCB(t, t + T_r)$ is the minimum number of servers that correctly reply and then, before $t + T_r$ are affected and incorrectly reply. Let us observe that a correct server correctly reply when it belongs to $Co(t, t + \Delta)$, it follows that servers in $B(t, t + \Delta)$ do not correctly reply. Thus, $minCB(t, t + T_r) = MaxB(t, t + T_r) - MaxB(t, t + T_r - \Delta)$, it may happen that $MaxB(t, t + T_r) < MaxB(t, t + T_r - \Delta)$, but obviously there can not be negative servers, so we consider only non negative values, $minCB(t, t + T_r) = \mathcal{R}(MaxB(t, t + T_r) - MaxB(t, t + T_r - \Delta))$.

**Lemma 10:** Let us consider a time interval $[t, t + T_r]$, $T_r \geq 2\delta$, let $\epsilon > 0$ be an arbitrarily small number. If $maxCu(t) > 0$ or $\gamma > \Delta$ then in the $(\Delta S, CUM)$ model $minCB = \lceil \frac{\Delta}{\gamma} - \Delta \rceil$, otherwise $minCB = \mathcal{R}(MaxB(t, t + T_r) - MaxB(t, t + T_r - \Delta))$.

Proof $minCB(t, t + T_r)$ is the minimum number of servers that correctly reply and then, before $t + T_r$ are affected by a mobile agent and incorrectly reply. We are interested in the maximum number of Byzantine servers in $B(t, t + T_r - \Delta)$, so that the remaining ones belong to $B(t + \Delta, t + T_r - \Delta)$, which means that servers in $B(t + T_r - \Delta, t + T_r)$ are in $Co(t, t + \Delta)$ (considering the scenario $\gamma$). Thus, considering that in the $(\Delta S, CAM)$ model we consider $t_EB(s_{-1}) = t + T_r - \epsilon (\epsilon > 0$ and arbitrarily small) then we consider the maximum number of “jumps” there could be in the time interval $[t + \Delta, t + T_r - \Delta]$. Thus $minCB(t, t + T_r) = \lceil \frac{\Delta}{\gamma} - \Delta \rceil = \lceil \frac{\Delta}{\gamma} \rceil$. If $MaxCu(t) = 0$ or $\gamma > \Delta$ then it has no sense to consider the $(\Delta S, CUM)$ worst case scenario that aims to maximize cured servers. Thus in this case we consider the $(\Delta S, CAM)$ worst case scenario, $minCB = \mathcal{R}(MaxB(t, t + T_r) - MaxB(t, t + T_r - \Delta))$, concluding the proof. □

**Lemma 11:** Let us consider a time interval $[t, t + T_r]$, $T_r \geq 2\delta$ then in the $(\Delta S, CUM)$ model then if $maxCu(t) > 0$ $minCB = \lceil \frac{\Delta}{\gamma} - \Delta \rceil + \mathcal{R}(\lceil \frac{\Delta}{\gamma} \rceil - \lceil \frac{\Delta}{\gamma} - 2\delta \rceil) + (MaxB(t) - MaxSil(t, t + T_r))$, otherwise $minCB$ assumes the same values as in the $(\Delta S, CAM)$ case.

Proof By definition $minCB(t, t + T_r) = minCB(t, t + T_r) + minCB(t, t + T_r)$ From Lemma 10 if $maxCu(t) > 0$ or $\Delta > \gamma$ then in the $(\Delta S, CUM)$ model $minCB = \lceil \frac{\Delta}{\gamma} - \Delta \rceil$, otherwise $minCB = \mathcal{R}(MaxB(t, t + T_r) - MaxB(t, t + T_r - \Delta))$.

$minCB(t, t + T_r)$ is the minimum number of servers that incorrectly reply and then, before $t + T_r - \Delta$ become correct so that are able to correctly reply in time such that their reply is delivered. In the $(\Delta S, CUM)$ model servers may incorrectly reply because affect by a mobile agent or because in a cured state. In the first case, a server is able to correctly reply if it become correct before $t + T_r - \Delta$ (the reply message needs at most $\Delta$ time to be delivered). Thus we consider the maximum number of servers that can be affected in the period $t + T_r - \gamma - \Delta, t + T_r$, which is $\lceil \frac{2\Delta}{\gamma} \rceil$. Thus, among the Byzantine servers (i.e., $MaxB(t, t + T_r)$) we consider servers not affected in the time interval $[t + \Delta, t + T_r - \gamma - \Delta]$. In other words such servers have $\gamma$ time to become correct and $\Delta$ time to reply before the end of the operation. Thus $MaxB(t, t + T_r) - MaxB(t, t + T_r - \gamma - \Delta, t + T_r)$, Again we can not have a negative number, so it becomes $\mathcal{R}(\lceil \frac{\Delta}{\gamma} \rceil)$. Concerning servers that incorrectly reply when in a cured state, we are interested in servers that correctly reply after in time such that the reply is delivered by the client, i.e., they are not silent. This number is easily computable, $MaxCu(t) - MaxSil(t, t + T_r)$. Thus $minCB(t, t + 2\delta) = (MaxCu(t) - MaxSil(t, t + T_r))$. Summing up if $maxCu(t) > 0$ or $\Delta > \gamma$, then $minCB = $
\[
\left[\frac{T_e - c - d}{\Delta}\right] + R\left(\left[\frac{T_e}{\Delta}\right] - \left[\frac{c - d}{\Delta}\right]\right) + (\text{MaxCu}(t) - \text{MaxSil}(t, t + 2\delta)),\] otherwise minCBC assumes the same values as in the \((\Delta S,CAM)\) model, which concludes the proof. \(\square\)

In Table \[\text{VII}\] are reported all the results found so far for \((\Delta S,\ast)\) models.

Such results have been proved considering \(f = 1\). Extending such results to scenario for \(f > 1\) is straightforward in the \((\Delta S,\ast)\) model. The extension to \(f > 1\) in the \((ITB,\ast)\) and \((ITU,\ast)\) models is less direct. What is left to prove is that the results found for \(f = 1\) can be applied to all other models in which mobile agents move independently from each other. In the following Lemma we employ \(\ast\) to indicate that the result holds for \(\ast\) assuming consistently the value \(CAM\) or \(CUM\).

**Lemma 12:** Let \(n_{*_{lb}} = \alpha_s(\Delta, \delta, \gamma) f\) be the impossibility results holding in the \((\Delta S,\ast)\) model for \(f = 1\). If there exists a tight protocol \(P_{reg}\) solving the safe register for \(n = \alpha_s(\Delta, \delta, \gamma) f + 1\) then all the Safe Register impossibility results that hold in the \((\Delta S,\ast)\) models hold also in the \((ITB,\ast)\) and \((ITU,\ast)\) models.

**Proof** Let us consider the scenario \(S^\ast\) for \(f = 1\) and a read() operation time interval \([t, t + T_e], t \geq 0\). Depending on the value of \(t\) there can be different (but finite) read scenarios, \(r_{s_1}, r_{s_2}, \ldots, r_{s_k}\). By hypothesis there exists \(P_{reg}\) solving the safe register for \(n = \alpha_s(\Delta, \delta, \gamma) f + 1\) then among the read scenarios \(\mathcal{R}S = \{r_{s_1}, r_{s_2}, \ldots, r_{s_k}\}\) all the possible worst case scenarios \(\{w_{rs_1}, \ldots, w_{rs_k}\} \subseteq \mathcal{R}S\) hold for \(n = \alpha_s(\Delta, \delta, \gamma) f\) (meaning that \(P_{reg}\) does not exist). We can say that those worst scenarios are equivalent in terms of replicas, i.e., for each \(w_{rs_k}\) it is possible to build an impossibility run if \(n = \alpha_s(\Delta, \delta, \gamma) f\) but \(P_{reg}\) works if \(n = \alpha_s(\Delta, \delta, \gamma) f + 1\) (if we consider \(f = 1\)). Let us now consider \((\Delta S,\ast)\) for \(f > 1\). In this case, mobile agents move all together, thus the same \(w_{rs_k}\) scenario is reproduced \(f\) times. For each \(w_{rs_k}\) scenario it is possible to build an impossibility run if \(n = \alpha_s(\Delta, \delta, \gamma) f\), i.e., \(\alpha_s(\Delta, \delta, \gamma) - f\) non Byzantine servers are not enough to cope with 1 Byzantine server, then it is straightforward that \(\alpha_s(\Delta, \delta, \gamma) - f\) non Byzantine servers are not enough to cope with \(f\) Byzantine servers, the same scenario is reproduced \(f\) times.

In the case of unsynchronized movements (ITB and ITU) we consider \(\Delta = \min\{\Delta_1, \ldots, \Delta_f\}\). Each mobile agent generates a different read scenarios, those scenario can be up to \(f\). As we just stated, if \(P_{reg}\) exists, those worst case scenarios are equivalent each others in terms of replicas. Since all the worst case scenarios are equivalent in terms of replicas, thus impossibility results holding for mobile agents moving together hold also for mobile agent moving in an uncoordinated way. \(\square\)

In \[\text{[8]}\], for \(n \geq \alpha_s(\Delta, \delta, \gamma) f + 1\) \((f \geq 1)\), it has been presented a tight protocol \(P_{reg}\) that solves the Regular Register problem whose bounds match the safe register lower bounds. Thus the next corollary follows.

**Corollary 9:** Let \(n_{*_{lb}} = \alpha_s(\Delta, \delta, \gamma) f\) be the impossibility results holding in the \((\Delta S,\ast)\) model for \(f = 1\). All the Safe Register impossibility results hold also in the \((ITB,\ast)\) and \((ITU,\ast)\) models.

**APPENDIX B: UPPER BOUND FOR \((ITB, CAM)\) MODEL**

In this section, we present an optimal protocol \(P_{reg}\) with respect to the number of replicas, that implements a SWMR Regular Register in a round-free synchronous system for \((ITB,CAM)\) and \((ITU,CAM)\) instances of the proposed MBF model. The difference with respect \((\Delta S,CAM)\) model is that the time at which mobile agents move is unknown. Notice that each mobile \(ma_i\) agent has it own \(\Delta_i\). Since we do not have any other information we consider \(\Delta = \min\{\Delta_1, \ldots, \Delta_f\}\). Following the approach used in the \((\Delta S,CAM)\) model, our solution is still based on the following two key points: (1) we implement a maintenance() operation, in this case executed on demand; (2) we implement read() and write() operations following the classical quorum-based approach. The size of the quorum needed to carry on the operations, and consequently the total number of servers required by the computation, is dependent by the time to terminate the maintenance() operation, \(\delta\) and \(\Delta\) (see Table \[\text{VIII}\]). Contrarily to the solution presented in \[\text{[8]}\], we do not employ a a forwarding mechanism. Such mechanism is it not necessary since being the maintenance() operation on demand (i.e., \(\gamma \leq 2\delta\)) its duration increases and, informally speaking, there is no more need to rush to help cured servers to retrieve a lost value as soon as possible. In this case, the only propagating mechanism is on the maintenance() operation. In Table \[\text{VIII}\] \(n_{CAM}\) is the minimum number of required replicas, \#reply_{CAM} is the minimum number of expected reply messages carrying the same value from \#reply_{CAM} different servers and \#echo_{CAM} is the minimum number of echo messages carrying the same value from \#echo_{CAM} different servers. The last difference, with respect the \((\Delta S,CAM)\) model is the increased \#echo_{CAM}.

**A. \(P_{reg}\) Detailed Description.**

The protocol \(P_{reg}\) for the \((ITB,CAM)\) model is described in Figures \[\text{[11]}\] - \[\text{[13]}\] which present the maintenance(), write(), and read() operations, respectively.

**Local variables at client** \(c_i\). Each client \(c_i\) maintains a set \#reply_{j} that is used during the read() operation to collect the three tuples \(\langle j, \langle v, sn \rangle \rangle\) sent back from servers. In particular \(v\) is the value, \(sn\) is the associated sequence number and \(j\) is the identifier of server \(s_j\) that sent the reply back. Additionally, \(c_i\) also maintains a local sequence number \(csn\) that is incremented each time it invokes a write() operation and is used to timestamp such operations monotonically.

**Local variables at server** \(s_i\). Each server \(s_i\) maintains the following local variables (we assume these variables are initialized to zero, false or empty sets according their type):

- \(V_i\): an ordered set containing \(d\) tuples \(\langle v, sn \rangle\), where \(v\) is a value and \(sn\) the corresponding sequence number.
- Such tuples are ordered incrementally according to their \(sn\) values. The function insert\(\langle V_i, \langle v_k, sn_k \rangle \rangle\) places the
new value in \( V_i \) according to the incremental order and, if there are more than \( d \) values, it discards from \( V_i \) the value associated to the lowest \( sn \).

- **pending_read**: set variable used to collect identifiers of the clients that are currently reading.
- **cured**: boolean flag updated by the cured_state oracle. In particular, such variable is set to true when \( s_i \) becomes aware of its cured state and it is reset during the algorithm when \( s_i \) becomes correct.
- **echo_vals\(_i\)** and **echo_read**: two sets used to collect information propagated through \( ECHO \) messages. The first one stores tuple \((j, \langle v, sn \rangle)\) propagated by servers just after the mobile Byzantine agents moved, while the second stores the set of concurrently reading clients in order to notify cured servers and expedite termination of \texttt{read}().
- **curing**: set used to collect servers running the maintenance() operation. Notice, to keep the code simple we do not explicitly manage how to empty such set since has not impact on safety properties.

In order to simplify the code of the algorithm, let us define the following functions:

- **select_d_pairs_max_sn(echo_vals\(_i\))**: this function takes as input the set \( echo\_vals\(_i\) \) and returns, if they exist, three tuples \((v, sn)\), such that there exist at least \#\( echo\_CAM \) occurrences in \( echo\_vals\(_i\) \) of such tuple. If more than three of such tuple exist, the function returns the tuples with the highest sequence numbers.
- **select_value(reply\(_i\))**: this function takes as input the reply\(_i\) set of replies collected by client \( c_i \) and returns the pair \((v, sn)\) occurring at least \#\( reply\_CAM \) times (see Table VII). If there are more pairs satisfying such condition, it returns the one with the highest sequence number.
- **delete_cured_values(echo_vals)**: this function takes as input \( echo\_vals\(_i\) \) and removes from \( fw\_vals\(_i\) \) all values coming from servers that sent an \texttt{ECHO()} message containing \( \perp \).

### The maintenance() operation

Such operation is executed by servers on demand when the oracle notifies them that are in a cured state. Notice that in the \((*, CAM)\) models servers knows when a mobile agent leaves them, thus depending on such knowledge they execute different actions. In particular, if a server \( s_i \) is not in a cured state then it does nothing, it just replies to \texttt{ECHO_REQ()} messages. Otherwise, if a server \( s_i \) is in a cured state it first cleans its local variables and broadcasts to other servers an echo request then, after \( 2\delta \) time units it removes value that may come from servers that were Byzantine before the maintenance() and updates its state by checking the number of occurrences of each pair \((v, sn)\) received with \( ECHO \) messages. In particular, it updates \( V_i \) invoking the select_three_pairs_max_sn(echo_vals\(_i\)) function that populates \( V_i \) with \( d \) tuples \((v, sn)\). At the end it assigns false to cured\(_i\) variable, meaning that it is now correct and the echo_vals\(_i\) can now be emptied. Contrarily to the \((\Delta S, CAM)\) case, cured server notifies to all that it has been Byzantine in the previous \( \delta \) time period. This is done invoking the awareAll function that broadcast a default value \( \perp \) after \( \delta \) time that a server discovered to be in a cured state.

### The write() operation

When the writer wants to write a value \( v \), it increments its sequence number \( csn \) and propagates \( v \) and \( csn \) to all servers. Then it waits for \( \delta \) time units (the maximum message transfer delay) before returning.

When a server \( s_i \) delivers a WRITE, it updates its local variables and sends a \texttt{REPLY()} message to all clients that are currently reading (clients in pending_read\(_i\)) to notify them about the concurrent write() operation and to each server executing the maintenance() operation (servers in curing\(_i\)).

### The read() operation

When a client wants to read, it broadcasts a READ() request to all servers and waits \( 2\delta \) time (i.e., one round trip delay) to collect replies. When it is unblocked from the wait statement, it selects a value \( v \) invoking the select_value function on reply\(_i\), set, sends
function awareAll():
(01) broadcast ECHO(i, 1)
(02) wait(δ);
(03) broadcast ECHO(i, 1)

operation maintenance() executed while (TRUE) :
(04) curedi ← report_cured_state();
(05) if (curedi) then
(06) curedi ← false;
(07) curing_statei ← true;
(08) Vi ← ∅; echo_valsi ← ∅; pending_readi ← ∅; curingi ← ∅;
(09) broadcast ECHO_REQ();
(10) awareAll();
(11) wait(2δ);
(12) delete_cured_values(echo_vals);
(13) insert(Vi, select_three_pairs_max sn(echo_valsi));
(14) for each (j ∈ (curingi)) do
(15) send ECHO (i, Vi) to sj;
(16) endFor
(17) curing_statei ← false;
(18) endif

when ECHO (j, Vj) is received:
(19) for each ((v, sn) ∈ Vj) do
(20) echo_vals, ← echo_vals, ∪ (v, sn)j;
(21) endFor

when ECHO_REQ (j) is received:
(22) curingi ← curingi ∪ j;
(23) if (Vj ≠ ∅)
(24) send ECHO(i, Vj);
(25) endif


Fig. 11. \(A_M\) algorithm implementing the maintenance() operation (code for server \(s_i\)) in the \((ITB, CAM)\) model.

\underline{Client code}

\underline{operation write(v)}:
(01) csn ← csn + 1;
(02) broadcast WRITE(v, csn);
(03) wait (δ);
(04) return write_confirmation;

\underline{Server code}

when WRITE(v, csn) is received:
(05) insert(Vi, (v, csn));
(06) for each j ∈ (pending_readi) do
(07) send REPLY (i, {<v, csn>});
(08) endFor
(09) for each j ∈ (curingi) do
(10) send ECHO (i, Vi);
(11) endFor

Fig. 12. \(A_W\) algorithm implementing the write(v) operation in the \((ITB, CAM)\) model.

an acknowledgement message to servers to inform that its operation is now terminated and returns \(v\) as result of the operation.

When a server \(s_i\) delivers a READ\((j)\) message from client \(c_j\) it first puts its identifier in the set \(\text{pending_read}_i\) to remember that \(c_j\) is reading and needs to receive possible concurrent updates, then \(s_i\) checks if it is in a cured state and if not, it sends a reply back to \(c_j\). Note that, the \(\text{REPLY}()\) message carries the set \(V_i\).

When a \(\text{READ\_ACK}(j)\) message is delivered, \(c_j\) identifier is removed from both \(\text{pending_read}_i\) set as it does not need anymore to receive updates for the current \(\text{read}(\) operation).

\textbf{B. Correctness (ITB, CAM)}

To prove the correctness of \(P_{reg}\), we first show that the termination property is satisfied i.e, that \(\text{read}()\) and \(\text{write}()\) operations terminates.

\textbf{Lemma 13:} If a correct client \(c_j\) invokes \(\text{write}(v)\) operation at time \(t\) then this operation terminates at time \(t + δ\).

\textbf{Proof} The claim follows by considering that a write\_confirmation event is returned to the writer client \(c_i\) after \(δ\) time, independently of the behavior of the servers (see lines [03][04] Figure [12]).

\textbf{Lemma 14:} If a correct client \(c_j\) invokes \(\text{read}()\) operation at time \(t\) then this operation terminates at time \(t + 2δ\).

\textbf{Proof} The claim follows by considering that a \(\text{read}()\) returns a value to the client after \(2δ\) time, independently of the behavior of the servers (see lines [03][06] Figure [13]).

\textbf{Theorem 3 (Termination):} If a correct client \(c_j\) invokes an operation, \(c_j\) returns from that operation in finite time.

\textbf{Proof} The proof follows from Lemma [13] and Lemma [14].

Validity property is proved with the following steps:
Proof The proof follows considering that:
- the maintenance() operation works (i.e., at the end of the operation $n - f$ servers store valid values). In particular, for a given value $v$ stored by #echo correct servers at the beginning of the maintenance() operation, there are $n - f$ servers that may store $v$ at the end of the operation;
- 2. given a write() operation that writes $v$ at time $t$ and terminates at time $t + \delta$, there is a time $t' > t + \delta$ after which #reply correct servers store $v$. 
- 3. at the next maintenance() operation after $t'$ there are #reply $\rightarrow$ #echo correct servers that store $v$, for step (1) this value is maintained.
- 4. the validity follows considering that the read() operation is long enough to include the $t'$ of the last written value before the read() and $V$ is big enough to do not be full filled with new values before $t'$.

Before to prove the correctness of the maintenance() operation let us see how many Byzantine agent there may be during such operation. Since the cured server run it as soon as the mobile agent $ma$ leaves it, then $ma$, movement are aligned to such operation, this agent contribution is $\frac{2k}{2k} = k$. All the others $f-1$ mobile agent are not aligned, thus their contribution is $Max B(t, 2+2\delta) = k+1$. Thus there are $k+(k+1) \times (f-1)$ Byzantine servers during the $2\delta$ time maintenance() operation.

Lemma 15 (Step 1): Let $T_i = t$ be the time at which mobile agent $ma_i$ leave $s_i$. Let $v$ be the value stored at #echoCAM servers $s_j \notin B(t, t+2\delta) \wedge s_j \in Co(t+\delta), v \in V_j \forall s_j \in Co(t+\delta)$. At time $t + 2\delta$, at the end of the maintenance(), $v$ is returned to $s_i$ by the function select_d_pairs_max_sn(echo_val$s_i$).

Proof The proof follows considering that:
- the maintenance() employs a request-reply pattern and during such operation, by hypothesis, there are #echoCAM servers that are never affected during the $[T_i, T_i + \delta]$ time period and are correct at time $T_i + \delta$. i.e., there are #echoCAM servers that deliver the ECHO_REQ() message (the can be either correct or cured) but are correct at time $T_i + \delta$ such that the reply is delivered by $s_i$ by time $T_i + 2\delta$.
- during the maintenance() operation there are $k+(k+1) \times (f-1)$ Byzantine servers, and $(\frac{k}{2})f$ servers that were Byzantine in $[t-\delta, t]$ time period, thus they could have sent incorrect messages as well.
- each cured servers, invokes AWAREALL() function, sends a $\perp$ message twice: when they are aware to be cured and $\delta$ time after. Thus by time $t + 2\delta$ server running the maintenance removes from echo_val$s_i$ the $(\frac{k}{2})f$ messages sent by those servers. In the end there are $k+(k+1) \times (f-1) = (k+1) f - 1$ messages coming from Byzantine servers in the echo_val$s_i$ set.

$#echoCAM = (k+1)f > (k+1)f - 1$ thus Byzantine servers can not force the select_d_pairs_max_sn(echo_val$s_i$) function to return a not valid value and select_d_pairs_max_sn(echo_val$s_i$) returns $v$ that occurs #replyCAM times, concluding the proof.

Lemma 16 (Step 2): Let opw be a write() operation invoked by a client $c_k$ at time $t_B(opw) = t$ then at time $t + \delta$ there are at least #replyCAM servers $s_j \notin B(t+\delta)$ such that $v \in V_j$.

Proof The proof follows considering that during the write() operation, $[t, t+\delta]$, there can be at most $(\frac{k}{2}+1)f$ mobile agents. Thus, during such time there are $n- (\frac{k}{2}+1)f = 2(k+1)f - (\frac{k}{2}+1)f = (k+\frac{k}{2}+1)f + 1$ servers $s_j$ that being either cured or correct, execute code in Figure [12] line [05] inserting $v$ in $V_j$. Finally, $(k+\frac{k}{2}+1)f+1 > (k+1)f+1 = #replyCAM$ concluding the proof.

For simplicity, for now on, given a write() operation $opw$ we call $t_B(opw) + \delta = twC$ the completion time of $opw$, the time at which there are at least #replyCAM servers storing the value written by $opw$.

Lemma 17 (Step 3): Let opw be a write() operation occurring at $t_B(opw) = t$ and let $v$ be the written value and let $twC$ be its completion time. Then if there are no other write() operations after $opw$, the value written by $opw$ is stored by all correct servers forever.

Proof Following the same reasoning as Lemma 16 at time $t + \delta$, assuming that in $[t, t+\delta]$ there are $(\frac{k}{2}+1)f$, then there are at least $(k+\frac{k}{2}+1)f+1$ servers $s_j$ that being either cured or correct, execute code in Figure [12] line [05] inserting $v$ in $V_j$. Now let us consider the following:
- Let $B_1 = B(t, t+\delta)$ be the set containing the $(\frac{k}{2}+1)f$ Byzantine servers during $[t, t+\delta]$, so that there are $2(k+1)f + 1 - \frac{k}{2} = (k+\frac{k}{2}+1)f + 1 \geq #replyCAM$ non faulty servers storing $v$;
  - there are $(\frac{k}{2})f$ Byzantine servers in $B_1$ that begin the maintenance() operation. At that time there are #replyCAM non faulty servers storing $v$, being #replyCAM > #echoCAM, for Lemma [15] at the end of the maintenance() operation, by time $t + 3\delta$, those servers obtain $v$ a re-
result of \(\text{select}_d\text{\_pairs\_max\_sn(\text{echo\_vals})}\) invocation, whose is stored in \(V\) since there are no other write() operation and since \(v\) has the highest associated sequence number.

- Let \(B_2 = \bar{B}(t + \delta, t + 2\delta)\) be the set containing Byzantine servers in the next \(\delta\) period. Those servers are \(\frac{k}{2} f\) at \(t + \delta\) (it is not \(\frac{k}{2} f + 1\), otherwise we would count the Byzantine servers at \(t + \delta\) twice). Thus, at \(t + 2\delta\) there are \((k + \frac{k}{2} f + 1) - \frac{k}{2} f = (k + 1) f = 1\) non faulty servers storing \(v\);
- there are \((\frac{k}{2}) f\) Byzantine servers in \(B_3\) that begin the maintenance() operation during \(t + \delta, t + 2\delta\) time interval. There are \(\#\text{\_rep\_CAM}\) non faulty servers storing \(v\), being \(\#\text{\_rep\_CAM} > \#\text{\_echo\_CAM}\), for Lemma 15 at the end of the maintenance() operation, by time \(t + 3\delta\), those servers, get \(v\) invoking select_d\_pairs\_max\_sn(\text{echo\_vals}),\) whose is stored in \(V\) since there are no other write() operation and since \(v\) has the highest associated sequence number.

- Let \(B_3 = \bar{B}(t + 2\delta, t + 3\delta)\) be the set containing Byzantine servers in the next \(\delta\) period. Those servers are \(\frac{k}{2} f\) at \(t + 3\delta\) there are \((k + 1) f + 1 - \frac{k}{2} f < \#\text{\_rep\_CAM}\) non faulty servers storing \(v\) and the there are \((\frac{k}{2}) f\) servers in \(B_1\) that terminated the maintenance() operation storing \(v\). Summing up there are \((k + 1) f + 1 - \frac{k}{2} f + \frac{k}{2} f = \#\text{\_rep\_CAM}\) servers storing \(v\).

Thus, after \(t + 3\delta\) period there are servers becoming affected that lose \(v\), but there are other \(f\) servers that become correct storing \(v\), so that all correct servers store \(v\). Since there are no more write() operation, this reasoning can be extended forever, concluding the proof.

**Lemma 18 (Step 3.):** Let \(op_{W_0}, op_{W_1}, \ldots, op_{W_{k-1}}, op_{W_k}, \ldots\) be the sequence of write() operations issued on the regular register. Let us consider a particular \(op_{W_k}\), let \(v\) be the value written by \(op_{W_k}\) and let \(t_{BU}W_k\) be its completion time. Then the register stores \(v\) (there are at least \(\#\text{\_rep\_CAM}\) correct servers storing \(v\)) up to at least time \(t_jW_{k+3}\).

**Proof** The proof simply follows considering that:
- for Lemma 17 if there are no more write() operation then 
  \(v\), after \(t_{BU}C\), is in the register forever.
- any new written value is store in an ordered set \(V\) (cf. Figure 12) whose dimension is 3.
- write() operations occur sequentially.

It follows that after the beginning of 3 write() operations, \(op_{W_{k+1}}, op_{W_{k+2}}, op_{W_{k+3}}, v\) it may be no more stored in the regular register.

**Theorem 4 (Step 4.):** Any read() operation returns the last value written before its invocation, or a value written by a write() operation concurrent with it.

**Proof** Let us consider a read() operation \(op_R\). We are interested in the time interval \([t_B(op_R), t_B(op_R) + \delta]\). Since such operation lasts \(2\delta\), the reply messages sent by correct servers within \(t_B(op_R) + \delta\) are delivered by the reading client. For \(\delta \leq \Delta < 3\delta\) during \([t, t + \delta]\) time interval there are \(n - \frac{\Delta}{\delta} - 1 \geq \#\text{\_rep\_CAM}\) correct servers that have the time to deliver the read request and reply. Now we have to prove that what those correct servers reply with is a valid value. There are two cases, \(op_R\) is concurrent with some write() operations or not.

- \(op_R\) is not concurrent with any write() operation. Let \(op_W\) be the last write() operation such that \(t_E(op_W) < t_B(op_R)\) and let \(v\) be the last written value. For Lemma 17 after the write completion time \(t_{CW}\) there are \(\#\text{\_rep\_CAM}\) non faulty servers storing \(v\). Since \(t_B(op_R) + \delta \geq t_{CW}\), then there are \(\#\text{\_rep\_CAM}\) non faulty servers replying with \(v\) (Figure 13). So the last written value is returned.

- \(op_R\) is concurrent with some write() operation. Let us consider the time interval \([t_B(op_R), t_B(op_R) + \delta]\). In such time there can be at most two write() operations. Thus for Lemma 18 the last written value before \(t_B(op_R)\) is still present in \(\#\text{\_rep\_CAM}\) non faulty servers. Thus at least the last written value is returned.

To conclude, for Lemma 3 during the read() operation there are at most \((k + 1) f\) Byzantine servers, being \(\#\text{\_rep\_CAM}\) \(\geq (k + 1) f\) then Byzantine servers may not force the reader to read another or older value and even if an older values has \(\#\text{\_rep\_CAM}\) occurrences the one with the highest sequence number is chosen.

**Theorem 5:** Let \(n\) be the number of servers emulating the register and let \(f\) be the number of Byzantine agents in the (ITB, CAM) round-free Mobile Byzantine Failure model. Let \(\delta\) be the upper bound on the communication latencies in the synchronous system. If \(n = n_{CAM}\) according to Table VIII then \(P_{reg}\) implements a SWMR Regular Register in the (ITB, CAM) and (ITU, CAM) round-free Mobile Byzantine Failure model.

**Proof** The proof simply follows from Theorem 3, and Theorem 4 considering \(\Delta = 1\) in the case of (ITU, CAM) model.

**Lemma 19:** Protocol \(P_{reg}\) for \(\delta \leq \Delta < 3\delta\) is tight with respect to \(\gamma \leq 2\delta\).

**Proof** The proof follows from Theorem 5 and Theorem 2, i.e., upper bound and lower bound match. In particular Lower bounds are computed using the values in Table VII to compute \(n_{CAM, LB}\) as defined in Table VI for \(\gamma \leq 2\delta\) (cf. Lemma 15).

**APPENDIX C: UPPER BOUND FOR (ITB, CUM) MODEL**

To prove the correctness of \(P_{reg}\) we demonstrate that the termination property is satisfied i.e., that read() and write() operations terminates. For the validity property we follow the same four steps as defined in Appendix B.

**Lemma 20:** If a correct client \(c_i\) invokes \(write(v)\) operation at time \(t\) then this operation terminates at time \(t + \delta\).

**Proof** The claim simply follows by considering that a write_confirmation event is returned to the writer client \(c_i\)
Lemma 21: If a correct client $c_i$ invokes \texttt{read()} operation at time $t$ then this operation terminates at time $t + 2\delta$.

Proof: The claim simply follows by considering that a \texttt{read()} returns a value to the client after $2\delta$ time, independently of the behaviour of the servers (see lines 15, 14, Figure 2).

Theorem 6 (Termination): If a correct client $c_i$ invokes an operation, $c_i$ returns from that operation in finite time.

Proof: The proof simply follows from Lemma 20 and Lemma 21.

To easy the next Lemmas let us use state the following result.

Lemma 22: Let $[t, t+2\delta]$ be a generic interval, then there are always at least \#replay\_CUM correct servers that reply during the $[t, t+\delta]$ time interval.

Proof: This follows considering the definition of minimum number of correct replies during a time interval (cf. Corollary 5). Since does exist a tight protocol $P$ solving a regular register in the $(\Delta S_C, CAM)$ model, then for Lemma 12 it is possible to apply values from Table VII to compute the minimum number of correct replies during the considered time interval, substituting values in each case the result is always at least \#replay\_CUM.

Lemma 23 (Step 1): Let $T_i$ be the time at which mobile agent $ma_i$ leave $s_c$ and let $t \leq T_i + 2\delta$ the time at which $s_c$ run the second maintenance() operation. Let $v$ be the value stored at \#echo\_CUM servers $s_j \in B(t, t+\delta), v \in V_j \forall s_j \notin B(t, t+\delta)$. At time $t + 2\delta$, at the end of the maintenance(), $v$ is returned to $s_c$ by the function select_three_pairs_max_sn(\texttt{echo}\_\texttt{vals}_c).

Proof: The proof follows considering that:

- the maintenance() employs a request-reply pattern and during such operation, by hypothesis, there are \#echo\_CUM servers that are never affected during the $[t, t+\delta]$ time period and are storing $v$ at time $t + \delta$, i.e., there are \#echo\_CUM servers that deliver the ECHO\_REQ() message (the echo is either correct or cured) but are storing $v$ in $V$ at time $t + \delta$ such that the reply is delivered by $s_c$ by time $t + 2\delta$.
- during the maintenance() operation can incorrectly contribute $(k+1)f$ Byzantine servers, and $(2k)$ servers that were Byzantine in $[t - 4\delta, t]$ time period, thus they could be still in a cured state.
- when the ECHO\_REQ() message is sent, $s_c$ uses a random number in order to be able to accept only ECHO() message sent after $t$.

\#echo\_CUM = $(3k)f + 1 > 3k$ Thus Byzantine servers can not force the select_three_pairs_max_sn(\texttt{echo}\_\texttt{vals}_s) function to return a not valid value so it returns $v$ that occurs \#replay\_CUM times, which is true since there exist \#echo\_CUM non faulty servers that reply to the ECHO\_REQ() message sending back $v$, concluding the proof.

In the sequel we consider $\gamma \leq 4\delta$. In the previous Lemma we proved that cured servers $s_c$ can get valid values in $2\delta$ time. Contrarily to all the previous model, the maintenance() operation is triggered each $2\delta$. Thus a mobile agent, just before to leave could leave $s_c$ with the timer just reset and garbage in the echo\_set\_c and $V_c$ sets, which does not allow $s_c$ to correctly terminate the operation. Thus $s_c$ has to wait $2\delta$ before to effectively start a correct maintenance() operation. In the sequel we refer to the first maintenance as the operation that may be ineffective and we refer to the second maintenance as the operation that allows a cured server to retrieve and store valid values. It is straightforward that $\gamma \leq 4\delta$ and the next Corollary just follows.

Corollary 10: Protocol $P$ implements a maintenance() operation that implies $\gamma \leq 4\delta$.

Lemma 24 (Step 2): Let $\texttt{op}_W$ be a write() operation invoked by a client $c_k$ at time $t_B(\texttt{op}_W) = t$ then at time $t + \delta$ there are at least $n - 2f > \#replay\_CUM$ non faulty servers $s_i$ such that $v \in W_i$ (so that when $s_i$ invokes conCut($V_i, V_{\text{safe}_i}$) $v$ is returned).

Proof: When the write() message is delivered by non faulty servers $s_i$, such message is stored in $W_i$ and a timer associated to it is set to $4\delta$, after that the value expires. For Lemma 3 in the $[t, t+\delta]$ time interval there are maximum $2f$ Byzantine servers. All the remaining $n - 2f$ non faulty servers execute the correct protocol code, Figure 2 line 15 inserting $v$ in $W_i$. Since write() operations are sequential, during $[t, t+\delta]$ there is only one new value inserted in $W_i$, which is returned by the function conCut() by construction.

For simplicity, for now on, given a write() operation $\texttt{op}_W$ we call $t_D(\texttt{op}_W) + \delta = t_{wc}$ the \textit{completion time} of $\texttt{op}_W$, the time at which there are at least \#replay\_CUM servers storing the value written by $\texttt{op}_W$.

Lemma 25 (Step 3): Let $\texttt{op}_W$ be a write() operation and let $v$ be the written value and let $t_{wc}$ be its time completion. Then if there are no other write() operation, the value written by $\texttt{op}_W$ is stored by all correct servers forever (i.e., $v \in \text{conCut}(V_i, V_{\text{safe}_i})$).

Proof: From Lemma 24 at time $t_{wc}$ there are at least $n - 2f > \#replay\_CUM$ non faulty servers $s_i$ such that $v \in W_i$. For sake of simplicity let us consider Figure 14. Let us consider that:

- for Lemma 24 all non faulty servers $s_i$ have $v \in W_i$ at most at $t_{wc}$:
- when $s_i$ runs the next maintenance(), $v$ is returned by select_three_pairs_max_sn(\texttt{echo}\_\texttt{vals}_s) function at the end of such operation, and since it is the value with the highest sequence number (there are no other write() operation) then $v$ is inserted in $V_{\text{safe}_i}$ (cf. Figure 1 line 10), thus such value is present in the ECHO() message replies for the next $2\delta$ time;
- this is trivially true up to time $t' = t + 4\delta$, for the timer associated to each $v \in W_i$. In $[t, t']$ there are $2k + 1$
Byzantine servers, thus $v \in W_j$ at $n - (2k + 1)$ non faulty servers, and $n - (2k + 1) = (3k + 1)f + 1 = \#replic_{CUM} \geq \#echo_{CUM}$.

- for each non faulty server the next maintenance() operation $op_M$ can happen either in $[t', t' + \delta]$ or in $[t' + \delta, t' + 2\delta]$ (cf. Figure 14) s10 and s11 respectively:
  
  \[-t_B(op_M) \in [t', t' + \delta] \quad \text{(cf. s10 Figure 14): s10 starts} \quad \text{before } t' + \delta, \quad \text{let us name it server type A. This}
  \text{means that } t_B(op_{M-1}) + \delta < t' - \delta, \quad \text{thus for}
  \text{Lemma 24 at the end of the operation } v \in V_{safe_0}
  \text{and during } op_M, v \in V_{10};
  \]
  
  \[-t_B(op_M) \in [t' + \delta, t' + 2\delta] \quad \text{(cf. s11 Figure 14): s11 starts} \quad \text{after } t' + 2\delta, \quad \text{let us name it server type B. This}
  \text{means that } t_B(op_{M-1}) + \delta > t', \quad \text{thus at the end}
  \text{of the operation we can not say that } v \in V_{safe_0} \quad \text{but at least during } op_{M-1}, v \in V_{51}.\]

If all non faulty servers are type A, during $op_M$, all non faulty servers have $v \in V$ and insert $v$ in the ECHO() message. The same happens if all non faulty servers are type B, during $op_{M-1}$, all of them inter $v$ in the ECHO() message and the maintenance() operation terminates with such value. If the situation is mixed, then servers type B, when run $op_{M-1}$, deliver ECHO() messages from both type A and type B servers. Thus if there are enough occurrence of $v$ they can store $v \in V_{safe_0}$ and during $op_M$, $v \in V_{51}$. During such operation both servers type A and type B have $vinV$. Again, if there are enough occurrences of $v$, the operation ends with $v \in V_{safe_0}$. It follows that servers type A, when run $op_M$, delivers ECHO() messages containing $v$ from both type A and type B servers. During the time interval $[t', t' + 2\delta]$ there are $k$ correct servers that are affected by mobile agent, cf. Figure 14 s5 and s6. At the same time there is server $s_0$, type A, that terminate its maintenance() with $v \in V_{safe_0}$, and thus compensates $s_5$, allowing $s_1$, type B, to terminate the maintenancem() operation with $v \in V_{safe_1}$, which compensates $s_6$. This cycle, between type A and type B servers can be extended forever. By hypothesis there are no more write() operation, thus all correct servers have $v \in V_{safe}$ or $V$, and $v$ is returned when servers invoke function conCut().

Lemma 26 (Step 3.): Let $op_{W_0}, op_{W_1}, \ldots, op_{W_{k-1}}, op_{W_k}, op_{W_{k+1}}, \ldots$ be the sequence of write() operation issued on the regular register. Let us consider a generic $op_{W_k}$, let $v$ be the written value by such operation and let $t_wC$ be its completion time. Then $v$ is in the register (there are $\#replic_{CUM}$ correct servers that return it when invoke the function conCut()) up to time at least $t_BW_{k+3}$.

Proof The proof simply follows considering that:

- for Lemma 25 if there are no more write() operation then $v$, after $t_wC$, is in the register forever.
- any new written value eventually is stored in an ordered set $V_{safe}$ and then $V$ (cf. Figure 1 line 6 or line 10) whose dimension is three.
- write() operation occur sequentially.

It follows that after three write() operations, $op_{W_{k+1}}, op_{W_{k+2}}, op_{W_{k+3}}$ in $V V_{safe}$ and $W$ there are three values whose sequence number is higher than the one associated to $v$, thus by construction conCut() does not return $v$ anymore, $v$ is no more stored in the regular register.

**Theorem 7 (Step 4.):** Any read() operation returns the last value written before its invocation, or a value written by a write() operation concurrent with it.

**Proof** Let us consider a read() operation $op_R$. We are interested in the time interval $[t_B(op_R), t_B(op_R) + \delta]$. Since such operation lasts $\delta$, the reply messages sent by correct servers within $t_B(op_R) + \delta$ are delivered by the reading client. During $[t, t + \delta]$, for Lemma 22 there are at least $\#replic_{CUM}$ correct servers that reply. Now we have to prove that what those correct servers reply with is a valid value. There are two cases, $op_R$ is concurrent with some write() operations or not.

- $op_R$ is not concurrent with any write() operation. Let $op_W$ be the last write() operation such that $t_E(op_W) \leq t_B(op_R)$ and let $v$ be the last written value. For Lemma 25 after the write completion time $t_wC$ there are at least $\#replic_{CUM}$ correct servers storing $v$ (i.e., $v \in conCut(V_j, V_{safe_0})$). Since $t_B(op_R) + \delta - t_wC \geq \delta$, then there are $\#replic_{CUM}$ correct servers replying with $v$ (cf. Lemma 22), by hypothesis there are no further write() operation and $v$ has the highest sequence number. It follows that the last written value $v$ is returned.

- $op_R$ is concurrent with some write() operation. Let us consider the time interval $[t_B(op_R), t_B(op_R) + \delta]$. In such time there can be at most two write() operations. Thus for Lemma 26 the last written value before $t_B(op_R)$ is still present in $\#replic_{CUM}$ correct servers and all of them reply (cf. Lemma 22) thus at least the last written value is returned. To conclude, for Lemma 2 during the read() operation there are at most $(k + 1)f$ Byzantine servers and $2k$ cured servers being $\#replic_{CUM} = (3k + 1)f + 1 > (3k + 1)f$ then Byzantine servers may not force the reader to read another or older value and even if an older values has $\#replic_{CUM}$ occurrences the one with the highest sequence number is returned, concluding the proof.

**Theorem 8:** Let $n$ be the number of servers emulating the register and let $f$ be the number of Byzantine agents in the (ITB, CUM) round-free Mobile Byzantine Failure model. Let $\delta$ be the upper bound on the communication latencies in the synchronous system. If $n \geq (5k + 2)f + 1$, then $P_{reg}$ implements a SWMR Regular Register in the (ITB, CUM) round-free Mobile Byzantine Failure model.

**Proof** The proof simply follows from Theorem 6 and Theorem 7.

Lemma 27: Protocol $P_{reg}$ is tight in the (ITB, CUM) model with respect to $\gamma \leq \delta$.

**Proof** The proof follows from Theorem 8 and Theorem 2 i.e., upper bound and lower bound match. In particular Lower 7Servers where affected in the previous $4\delta$ time period, thus they are still running the two maintenance() operations, that last at most $4\delta$.
write\((v)\) \hspace{1cm} t' \hspace{1cm} t' + 2\delta

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig14}
\caption{Maintenance operation op_{M_1} analysis after a write operation, \(t' = t + 4\delta\). White rectangles are maintenance operation run by correct servers. In particular \(s_{10}\) runs such operation during the first \(\delta\) period after \(t'\), while \(s_{11}\) runs it during the second \(\delta\) period.}
\end{figure}

Fig. 14. maintenance\() operation op_{M_1} analysis after a write\() operation, t' = t + 4\delta\). White rectangles are maintenance\() operation run by correct servers. In particular s_{10} runs such operation during the first \(\delta\) period after \(t'\), while s_{11} runs it during the second \(\delta\) period.

bounds are computed using the values in Table VII to compute \(n_{CUM, LB}\) as defined in Table \(\text{VII}\) for \(\gamma \leq 4\delta\) (cf. Corollary \(\text{I}\)).