Optimal Storage under Unsynchrononized Mobile Byzantine Faults

Silvia Bonomi, Antonella del Pozzo, Maria Potop-Butucaru, Sébastien Tixeuil

▶ **To cite this version:**

Silvia Bonomi, Antonella del Pozzo, Maria Potop-Butucaru, Sébastien Tixeuil. Optimal Storage under Unsynchrononized Mobile Byzantine Faults. [Research Report] UPMC - Université Paris 6 Pierre et Marie Curie; Sapienza Università di Roma (Italie). 2017. hal-01561776

HAL Id: hal-01561776
https://hal.science/hal-01561776

Submitted on 13 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Optimal Storage under Unsynchrononized Mobile Byzantine Faults

Silvia Bonomi*, Antonella Del Pozzo*†, Maria Potop-Butucaru†, Sébastien Tixeuil†

*Sapienza Università di Roma,Via Ariosto 25, 00185 Roma, Italy
{bonomi, delpozzo}@dis.uniroma1.it
†Université Pierre & Marie Curie (UPMC) – Paris 6, France
{maria.potop-butucaru, sebastien.tixeuil}@lip6.fr

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 omniscient adversary from a server to another in order to deviate their computation in an unforeseen manner. When a server is infected by an 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.

1 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 [19].

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 [9] agents may only move from one host to another when protocol messages are sent (similarly to how viruses would propagate), while *(ii)* unconstrained mobility [2, 4, 10, 15, 16, 17] agents may move independently of protocol messages. In the case of unconstrained mobility, several variants were investigated [2, 4, 10, 15, 16, 17]: Reischuk [16] considers that malicious agents are stationary for a given period of time, Ostrovsky and Yung [15] introduce the notion of mobile viruses and define the adversary as an entity that can inject and distribute faults; finally, Garay [10], and more recently Banu *et al.* [2], and Sasaki *et al.* [17] and Bonnet *et al.* [4] 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 [2, 4, 10, 17] lies in the knowledge that hosts have about their previous infection by a Byzantine agent. In Garay's model [10], a host is able to detect its own infection after the Byzantine agent left it. Sasaki *et al.* [17] investigate a model where hosts cannot detect when Byzantine agents leave. Finally, Bonnet *et al.* [4] 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.* [8]. 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.* [8] model implies that Byzantine moves are no more related to messages that are exchanged through the protocol.

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* [18], and *Byzantine quorum systems* [3, 12, 14, 13]. Both approaches are based on the idea that the current state of the storage is replicated among processes, and the main difference lies in the number of replicas that are simultaneously involved in the state maintenance protocol. Several works investigated the emulation of self-stabilizing or pseudo-stabilizing Byzantine tolerant SWMR or MWMR registers [1, 7, 6]. All these works do not consider the complex case of mobile Byzantine faults. Recently, Bonomi *et al.* [5] proposed optimal self-stabilizing atomic register implementations for round-based synchronous systems under the four Mobile Byzantine models described in [2, 4, 10, 17]. 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 and Byzantine agents are synchronous [8], the only case remaining open is that of synchronous processes and unsynchronized Byzantine agents.

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

Table 1: Summary of lower bounds in different system models. δ is the upper bound on the message delay, and Δ is the period for synchronized agent moves (in the synchronous agents setting) or the lower bound for an agent to remain on a server (in the unsynchronized agents setting).

Round-based model [5]			
Burhman	Garay	Bonnet	Sasaki
$2f+1$	$3f+1$	$4f+1$	$4f+1$

	Round-free model			
Agents moves	Synchronized [8]		**Unsynchronized [this paper]**	
Cured state awareness	Aware	Unaware	**Aware**	**Unaware**
$\delta \leq \Delta < 2\delta$	$5f+1$	$8f+1$	$\mathbf{6f+1}$	$\mathbf{12f+1}$
$2\delta \leq \Delta < 3\delta$	$4f+1$	$5f+1$	$\mathbf{4f+1}$	$\mathbf{7f+1}$

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 1 summarizes all the lower bounds for the various models, the newly obtained results are presented in boldface.

2 System Model

We consider a distributed system composed of an arbitrary large set of client processes $\mathcal{C}$, and a set of n server processes $\mathcal{S} = \{s_1, s_2 \dots s_n\}$. 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 $c_i \in \mathcal{C}$ 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(m) primitive, then there is a constant δ_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 δ (equal to $\delta_b \geq \delta_p$), and assume δ 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 $\rightarrow$ be the happened-before relation. An execution history (or simply history) $\hat{H} = (H, \rightarrow)$ is a partial order on H satisfying the relation $\rightarrow$.*

Definition 2 (Valid State at time t) *Let $\hat{H} = (H, \rightarrow)$ be an execution history of a generic computation and let $\mathcal{P}$ be the corresponding protocol. Let p_i be a process and let $state_{p_i}$ be the state of p_i at some time t. $state_{p_i}$ 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 $\mathcal{P}$ be the protocol generating $\hat{H}$. A process is* correct at time t *if* (i) *it is correctly executing $\mathcal{P}$, 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.*, $Co([t, t']) = \bigcap_{\tau \in [t,t']} Co(\tau)$).*

Definition 4 (Byzantine process at time t) *Let $\hat{H} = (H, \rightarrow)$ be a history, and let $\mathcal{P}$ 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 $\mathcal{P}$. 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.*, $B([t, t']) = \bigcap_{\tau \in [t,t']} B(\tau)$).*

Definition 5 (Cured process at time t) *Let $\hat{H} = (H, \rightarrow)$ be a history, and let $\mathcal{P}$ be the protocol generating $\hat{H}$. A process is* cured at time t *if* (i) *it is correctly executing $\mathcal{P}$, 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.*, $Cu([t, t']) = \bigcap_{\tau \in [t,t']} Cu(\tau)$).*

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 Δ_i. Given two mobile Byzantine Agents ma_i and ma_j, their movement periods Δ_i and Δ_j may be different. Note that previous results considering decoupled Byzantine moves [8] were established in the weaker Δ-synchronized model, where the external adversary moves *all* controlled mobile Byzantine agents at the same time t, and their movements happen periodically with period Δ. 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 agents[1]. It follows that, at any time t $|B(t)| \leq f$. 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' ($op \prec op'$) if and only if $t_E(op) < t_B(op')$. If op does not precede op' and op' does not precede op, then op and op' are *concurrent* (noted $op||op'$). 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.

3 Lower bounds

In this section we prove lower bounds with respect to the minimum fraction of correct servers to implement safe registers in presence of mobile Byzantine failures [2]. 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 time to 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 δ and Δ (that states how many Byzantine servers there may be during an operation);
- T_r, the read() operations duration;

[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.

[2]Results on safe register can be directly extended to the other register specifications.

- γ, 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 γ 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 Δ and γ. For clarity, in the sequel we note *correct message/request/reply* a message that carries a valid 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 $\mathcal{P}_{reg}$ 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).*

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 γ, the upper bound of the time during which a server can be in a cured state.

Definition 7 (Curing time, γ) *We define γ as the maximum time a server can be in a cured state. More formally, let T_c the time at which server s_c 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 execution, we define below a scenario of agents movement. Then, with respect 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 $s_{(i \mod n)f+1}, \ldots, s_{(i \mod n)f+f}$.*

In Figure 1 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 $\mathcal{P}_{reg}$ 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 $T_r \geq 2\delta$ time. Moreover we consider that a correct server sends a $reply$

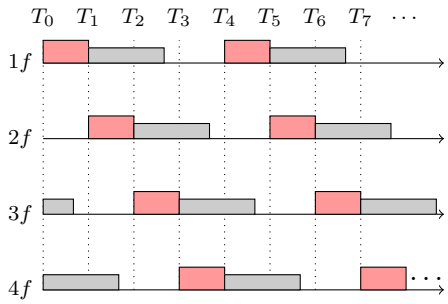

Figure 1: Representation of S^* where mobile agents affect groups of f different servers each T_i period. In particular here $\gamma > \Delta$.

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 $[t, t+T_t]$ 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, $state_i \in \{correct, cured, Byzantine\}$. Let $S(t')$ be the set of servers s_i that are in the state $state_i$ at t', $S(t') \in \{Co(t'), Cu(t'), B(t')\}$. $\tilde{S}(t, t+T_r)$ 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, $\tilde{S}(t, t+T_r) = \bigcup_{t \leq t' \leq t+T_r} S(t')$.*

Definition 10 ($C\tilde{B}C(t, t+T_r)$) *Let $[t, t+T_r]$ be a time interval, $C\tilde{B}C(t, t+T_r)$ denotes servers that during a time interval $[t, t+T_r]$ belong first to $\tilde{B}(t, t+T_r)$ or $Cu(t)$ (only in $(\Delta S, CUM)$ model) and then to $Co(t+\delta, t+T_r-\delta)$ or vice versa.*
In particular let us denote:

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

Definition 11 ($Sil(t, t+T_r)$) *Let $[t, t+T_r]$ be a time interval. $Sil(t, t+T_r)$ is the set of servers in $Cu(t, t+T_R-\delta)$.*

Servers belonging to $Sil(t_B(op_R), t_E(op_R))$ are servers that do no participate to op_R. In oder words, those servers in the worst case scenario became correct after $t_E(opR) - \delta$, thus if they send back a correct

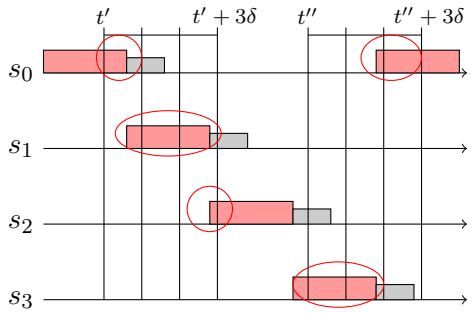

Figure 2: Let $[t, t+T_r]$ be time a interval such that in the given scenario $|\tilde{B}(t, t+T_r)| = Max\tilde{B}(t, t+T_r)$. In particular we have that in the time interval $[t', t'+T_r]$, $|\tilde{B}(t', t'+T_r)| = Max\tilde{B}(t, t+T_r)$. While in the time interval $[t'', t''+T_r]$, $|\tilde{B}(t'', t''+T_r)| < Max\tilde{B}(t, t+T_r)$.

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

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 2 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 13 ($MaxSil(t, t+T_r)$) *Let S be a scenario and $[t, t+T_r]$ a time interval. The cardinality of $Sil_S(t, t+T_r)$ is maximum with respect to S if for any t', $t' \geq 0$ we have that $|Sil(t, t+T_r)| \geq |Sil(t', t'+T_r)|$ and $\tilde{B}(t, t+T_r) = Max\tilde{B}(t, t+T_r)$. Then we call the value of such cardinality as $MaxSil_S(t, t+T_r)$. If we consider only one scenario per time then we can omit the subscript related to the scenario and write directly $minSil(t, t+T_r)$.*

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 14 ($MaxCu(t)$) *Let S be a scenario and t be a time instant. The cardinality of $Cu_S(t)$ is maximum with respect to S if for any t', $t' \geq 0$, we have that $|Cu_S(t')| \leq |Cu_S(t)|$ and $\tilde{B}(t, t+T_r) = Max\tilde{B}(t, t+T_r)$. We call the value of such cardinality as $MaxCu_S(t)$. If we consider only one scenario per time then we can omit the subscript related to it and write directly $MaxCu(t)$.*

This value quantifies, in the worst case scenario, how many cured servers there may be at the beginning of a read() operation. Figure 3 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.

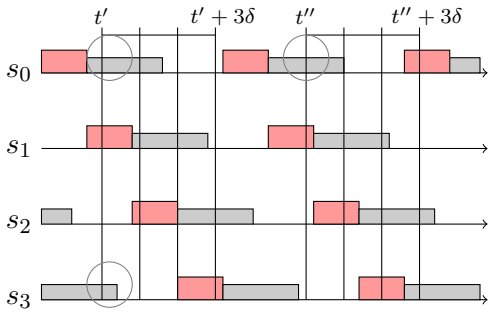

Figure 3: Let us consider the time instant t and the depicted scenario such that $|Cu(t)| = MaxCu(t)$. In particular, in this case $|Cu(t')| = MaxCu(t)$ and $|Cu(t'')| < MaxCu(t)$.

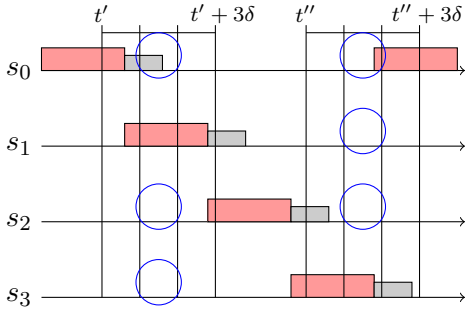

Figure 4: Let $[t, t+T_r]$ be a time interval such that in the depicted scenario $|\tilde{Co}(t, t+T_r)| = min\tilde{Co}(t, t+T_r)$. Then in both time intervals $[t', t'+T_r]$ and $[t'', t''+T_r]$ we have that $|\tilde{Co}(t', t'+T_r)| = |\tilde{Co}(t'', t''+T_r)| = min\tilde{Co}(t, t+T_r)$.

Definition 15 ($min\tilde{Co}(t, t+T_r)$) *Let S be a scenario and $[t, t+T_r]$ be a time interval then $min\tilde{C}_S(t, t+T_r)$ denotes the minimum number of correct servers during a time interval $[t+\delta, t+T_r-\delta]$. If we consider only one scenario per time then we can omit the subscript related to it and write directly $min\tilde{C}(t, t+T_r)$.*

Figure 4 depicts a scenario where during the both intervals $[t', t'+T_r]$ and $[t'', t''+T_r]$ the number of correct servers is minimum.

Definition 16 ($minC\tilde{B}C(t, t+T_r)$) *Let $[t, t+T_r]$ be a time interval then $minC\tilde{B}C(t, t+T_r)$ denotes the minimum number of servers that during a time interval $[t, t+T_r]$ belong first to $\tilde{B}(t, t+T_r)$ or $Cu(t)$ (only in $(\Delta S, CUM)$ model) and then to $Co(t+\delta, t+T_r-\delta)$ or vice versa and $\tilde{B}(t, t+T_r) = Max\tilde{B}(t, t+T_r)$. In particular let us denote as:*

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

As we stated before, Byzantine servers set changes during the read() operation op_R, so there can be servers that are in a Byzantine state at $t_B(op_R)$ and in a correct state before $t_E(op_R) - \delta$ (cf. s_0 during

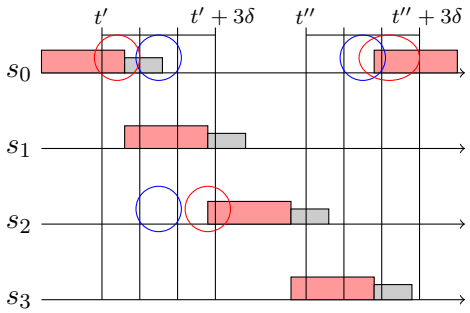

Figure 5: Let $[t, t+T_r]$ a time interval such that in the depicted scenario $C\tilde{B}C(t, t+T_r) = minC\tilde{B}C(t, t+T_r)$. Then $C\tilde{B}C(t', t'+T_r) > minC\tilde{B}C(t, t+T_r)$ and $C\tilde{B}C(t'', t''+T_r) = minC\tilde{B}C(t, t+T_r)$.

$[t', t'+3\delta]$ time interval in Figure 5). 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 5).

Lemma 1 $Max\tilde{B}(t, t+T_r) = (\lceil \frac{T_r}{\Delta} \rceil + 1)f$.

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 Δ time. It follows that the number of times it may change server is $\frac{T_r}{\Delta}$. Thus the affected servers are $\lceil \frac{T_r}{\Delta} \rceil$ plus the server that was affected at t. Finally, extending the reasoning to f agents, $Max\tilde{B}(t, t+T_r) = (\lceil \frac{T_r}{\Delta} \rceil + 1)f$, which concludes the proof. □

As we see in the sequel, the value of $Max\tilde{B}(t, t+T_r)$ 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 δ 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 17 ($MaxReplies_NCo(t, t+T_r)_k$) *Let $MaxReplies_NCo(t, t+T_r)_k$ 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 $Max\tilde{B}(t, t+T_r)$ and $MaxCu(t)$ 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 $Max\tilde{B}(t, t+T_r)$ incorrect replies in the $(\Delta S, CAM)$ model and $Max\tilde{B}(t, t+T_r) + MaxCu(t)$ incorrect replies in the $(\Delta S, CUM)$ model .*

Definition 18 ($minReplies_Co(t, t+T_r)_k$) *Let $minReplies_Co(t, t+T_r)_k$ 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 $s_j \in Co(t')$.*

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 $Sil(t, t+T_r)$ set, or *(iii)* servers that reply both correctly and incorrectly. The next Corollary follows.

Corollary 2 *In the worst case scenario, during a* read *operation lasting $T_r \geq 2\delta$ issued by client c_i, c_i delivers $n-(Max\tilde{B}(t,t+T_r)+MaxSil(t,t+T_r))+minC\tilde{B}C(t,t+T_r)$ correct replies in the $(\Delta S, CAM)$ model and $n-[Max\tilde{B}(t,t+T_r)+MaxCu(t)]+minC\tilde{B}C(t,t+T_r)$ 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 2 *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 $minReplies_Co(t, t+T_r)_k$ contains replies from x different servers.*

Proof Let us suppose that $minReplies_Co(t, t+T_r)_k$ contains replies from $x-1$ different servers (trivially it can not be greater than x). Without lost of generality, let us suppose that c_i collects replies from $s_1, \ldots, s_{x-1}$. It follows that there is a server $s_i, i \in [1, x-1]$ 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 $[t+\delta, t+T_r-\delta-\gamma+1]$. This implies that $T_r \leq \Delta + 2\delta + \gamma$. Since s_i replies twice then two scenarios are possible during *op*: *(i)* s_i was first affected by ma_k and then became correct (so it replied once), then affected again and then correct again (so it replied twice); *(ii)* s_i was correct (so it replied once), then it was affected by ma_k and then correct again (so it replied twice). Let us consider case *(ii)* (case *(i)* follows trivially). Since s_i had the time to reply (δ), to be affected and then became correct ($\Delta + \gamma$) and reply again (δ) this means that $T_r > \Delta + 2\delta + \gamma$. A similar result we get in case *(i)* where the considered execution requires a longer time. This is in contradiction with $T_r \leq \Delta + 2\delta + \gamma$ thus c_i gets replies for x different servers. □

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 message coming from the first server and 2 occurrences of the messages coming from the remaining servers. Thus the next Corollary directly follows.

Corollary 3 *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 $minReplies_Co(t, t+T_r)_k$ contains $x \mod n$ messages m_{ij} whose occurrences is $\lfloor \frac{x}{n} \rfloor + 1$ and $(n - x \ (mod\ n))$ messages whose occurrences is $\lfloor \frac{x}{n} \rfloor$.*

The case of $MaxReplies_NCo(t, t+T_r)_k$ 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.

Table 2: Lower bounds on the number of replicas in each model.

$n_{CAM_{LB}}$	$[2Max\tilde{B}(t,t+T_r) + MaxSil(t,t+T_r) - minC\tilde{B}C(t,t+T_r)]f$
$n_{CUM_{LB}}$	$[2(Max\tilde{B}(t,t+T_r) + MaxCu(t,t+T_r)) - minC\tilde{B}C(t,t+T_r)]f$

Corollary 4 *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 $MaxReplies_NCo(t, t+T_r)_k$ contains $x \mod n$ messages m_{ij} whose occurrences is $\lfloor \frac{x}{n} \rfloor + 1$ and $(n - x \ (mod\ n))$ messages whose occurrences is $\lfloor \frac{x}{n} \rfloor$.*

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 in order to distinguish correct and incorrect replies needs to get $minReplies_Co(t, t + T_r)_k > MaxReplies_NCo(t, t + T_r)_k$. It follows that the number of correct servers has to be enough to guarantee this condition. Table 2 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.

Theorem 1 *If $n < n_{CAM_{LB}}$ ($n < n_{CUM_{LB}}$) as defined in Table 2, then there not exists a protocol $\mathcal{P}_{reg}$ solving the safe register specification in $(\Delta S, CAM)$ model ($(\Delta S, CUM)$ model respectively).*

Proof Let us suppose that $n < n_{CAM_{LB}}$ ($n < n_{CUM_{LB}}$) and that protocol $\mathcal{P}_{reg}$ 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 severs involved in such operation reply once with 1. From Corollaries 1 and 2, c_i collects $MaxReplies_NCo(t, t + T_r)_i$ occurrences of 1 and $minReplies_Co(t, t + T_r)_i$ occurrences of 0. Since $\mathcal{P}_{reg}$ 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 severs involved in such operation replies once with 0. From Corollaries 1 and 2 and Corollary 3 and Corollary 4, c_i collects $MaxReplies_NCo(t, t+T_r)_i$ occurrences of 0 and $minReplies_Co(t, t+T_r)_i$ occurrences of 1. Since $\mathcal{P}_{reg}$ exists and no write operations occur, then c_i returns 1.

From Lemma 1 and using values in Table 2 we obtain following equations for both models:

- $(\Delta S, CAM)$:
  - $MaxReplies_NCo(t, t + T_r)_i = Max\tilde{B}(t, t + T_r) = (\lceil \frac{T_r}{\Delta} \rceil + 1)f$
  - $minReplies_Co(t, t+T_r)_i = n - [Max\tilde{B}(t, t+T_r) + MaxSil(t, t+T_r)] + minC\tilde{B}C(t, t+T_r) =$

$$[2(Max\tilde{B}(t, t + T_r)) + MaxSil(t, t + T_r)$$
$$-minC\tilde{B}C(t, t + T_r)]$$
$$-[(Max\tilde{B}(t, t + T_r) + MaxSil(t, t + T_r))$$
$$+minC\tilde{B}C(t, t + T_r)] =$$
$$Max\tilde{B}(t, t + T_r) = (\lceil \frac{T_r}{\Delta} \rceil + 1)f$$

- $(\Delta S, CUM)$:
  - $MaxReplies_NCo(t, t+T_r)_i = Max\tilde{B}(t, t+T_r) + MaxCu(t) = (\lceil \frac{T_r}{\Delta} \rceil + 1)f + MaxCu(t)$
  - $minReplies_Co(t, t+T_r)_i = n - [Max\tilde{B}(t, t+T_r) + MaxCu(t)] + minC\tilde{B}C(t, t+T_r) =$

$$[2Max\tilde{B}(t, t+T_r) + 2MaxCu(t)) - minC\tilde{B}C(t, t+T_r)] +$$
$$-[Max\tilde{B}(t, t+T_r) + MaxCu(t)] + minC\tilde{B}C(t, t+T_r) =$$
$$Max\tilde{B}(t, t+T_r) + MaxCu(t) = (\lceil \frac{T_r}{\Delta} \rceil + 1)f + MaxCu(t)$$

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.

□

$MaxReplies_NCo(t, t+T_r)_i$ and $minReplies_Co(t, t+T_r)_i$ 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 5 *For each $T_r \geq 2\delta$ if $n > n_{CAM_{LB}}$ ($n > n_{CUM_{LB}}$) then $MaxReplies_NCo(t, t+T_r)_i < minReplies_Co(t, t+T_r)_i$.*

At this point we compute $minCu(t)$, $MaxSil(t, t+T_r)$ and $minC\tilde{B}C(t, t+T_r)$ to finally state exact lower bounds depending on the system parameters, in particular depending on Δ, γ and the servers awareness, i.e., $(\Delta S, CAM)$ and $(\Delta S, CUM)$.

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 ma_k. Let $\{s_{-1}, s_{-2}, \ldots, s_{-c}\} \in Cu(t)$ 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_c the first server that became cured. Let $t_BB(s_i)$ and $t_EB(s_i)$ 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 become cured and the time instant in which it became correct. Considering that ma_k moves each Δ time then we have that $t_BB(s_{i-1}) - t_BB(s_i) = \Delta$ and $t_BCu(s_{-j}) - t_BCu(s_{-j+1}) = \Delta$. The same holds for the t_E of such states. Moreover $t_BB(s_1) = t_BCu(s_{-1})$. Now we are ready to build the read scenario with respect to S^*. In particular we build a scenario for the $(\Delta S, CAM)$ model and one for the $(\Delta S, CUM)$ model. Intuitively, the presence of cured servers do not have the same impact in the two models, thus in the $(\Delta S, CUM)$ model we maximize such number. Let $[t, t+2\delta]$ be the considered time interval and let ϵ be a positive number arbitrarily smaller, then we consider in the $(\Delta S, CAM)$ scenarios $t = t_EB(s_1) - \epsilon$ (cf. Figure 6) and in the $(\Delta S, CUM)$ scenarios $t_BB(s_b) = t + 2\delta - \epsilon$ (cf. Figure 7).

In the sequel we use the notion of Ramp Function:

$$\mathcal{R}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Lemma 3 *Let us consider a time interval $[t, t+T_r], T_r \geq 2\delta$ and an arbitrarily small number $\epsilon > 0$, then in fthe $(\Delta S, CAM)$ model $MaxCu(t) = \mathcal{R}(\lceil \frac{\gamma - \Delta + \epsilon}{\Delta} \rceil)$.*

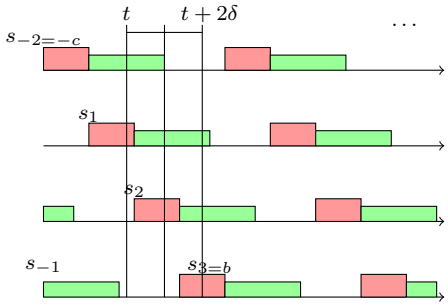

Figure 6: Representation of S^* when we consider a $(\Delta S, CAM)$ model, in particular $t_E B(s_1) = t + \epsilon$, for $\epsilon > 0$ and arbitrarily small.

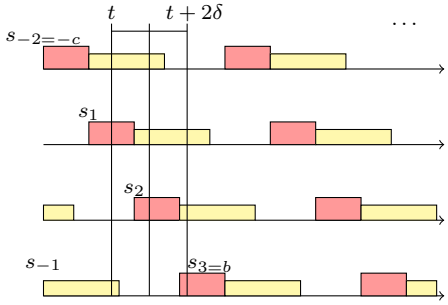

Figure 7: Representation of S^* when we consider a $(\Delta S, CUM)$ model, in particular $t_B B(s_c) = t + 2\delta - \epsilon$, for $\epsilon > 0$ and arbitrarily small.

Proof As we defined, s_{-1} is the most recent server that entered in a cured state, with respect to the considered time interval. Intuitively each s_{-j} is in $Cu(t)$ if $t_E Cu(s_{-j}) > t$. Considering that $t_E Cu(s_{-j}) - t_E Cu(s_{-j-1}) = \Delta$ then the number of servers in a cured state at t is $MaxCu(t) = \lceil \frac{t_E Cu(s_1) - t}{\Delta} \rceil$. [3] As we stated, for $(*, CAM)$ 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 $t_B B(s_1) = t_E B(s_1) - \Delta = t_B Cu(s_{-1})$. Substituting $t_B Cu(s_{-1}) = t + \epsilon - \Delta$, since we consider γ the upper bound for the curing time, then $t_E Cu(s_{-1}) = t + \epsilon - \Delta + \gamma$. So finally, $MaxCu(t) = \lceil \frac{t_E Cu(s_1) - t}{\Delta} \rceil = \lceil \frac{\gamma - \Delta + \epsilon}{\Delta} \rceil$ and since there can no be a negative result then $MaxCu(t) = \mathcal{R}(\lceil \frac{\gamma - \Delta + \epsilon}{\Delta} \rceil)$. This concludes the proof. □

Lemma 4 *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 $MaxCu(t) = \mathcal{R}(\lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil \Delta + \gamma}{\Delta} \rceil)$.*

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 $Cu(t)$ if $t_E Cu(s_{-j}) > t$. Considering that $t_E Cu(s_{-j}) - t_E Cu(s_{-j-1}) = \Delta$ then the number of servers in a cured state at t is $MaxCu(t) = \lceil \frac{t_E Cu(s_1) - t}{\Delta} \rceil$. As we state, for $(*, CUM)$ models we consider scenarios in which the end of the considered time interval, is just after $t_B B(s_b)$. Thus

[3] Consider Figure 6, s_2 is the most recent server that entered in the cured state. This is the server that spend more time in such state with respect to the others. It follows that other servers are in a cured state if during this time interval there is enough time for a "jump"

given an arbitrarily small number $\epsilon > 0$, let $t_B B(s_b) = t + T_r - \epsilon$. By construction we know that $t_B B(s_1) = t_E B(s_1) - \Delta = t_B Cu(s_{-1})$ and $t_B B(s_1) = t_B B(s_b) - \lceil \frac{T_r}{\Delta} \rceil \Delta$ (cf. Lemma 1). Substituting and considering that $t_E Cu(s_{-1}) = t_B Cu(s_{-1}) + \gamma$) we get the following: $t_E Cu(s_{-1}) = t + T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil + \gamma$. Finally $MaxCu(t) = \lceil \frac{t_E Cu(s_1) - t}{\Delta} \rceil = \lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil + \gamma}{\Delta} \rceil$ and since there can not be a negative result then $MaxCu(t) = \mathcal{R}(\lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil \Delta + \gamma}{\Delta} \rceil)$. This concludes the proof. □

Lemma 5 *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 $(\Delta S, CAM)$ model $MaxSil(t, t + T_r) = \mathcal{R}(\lceil \frac{\gamma - \Delta + \epsilon - T_r + \delta}{\Delta} \rceil)$.*

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 $Sil(t, t + 2\delta)$ if $t_E Cu(s_{-j}) > T_r - \delta$. Considering that $t_E Cu(s_{-j}) - t_E Cu(s_{-j-1}) = \Delta$ then the number of servers in a silent state at t is $MaxSil(t, t+2\delta) = \lceil \frac{t_E Cu(s_1) - T_r + \delta}{\Delta} \rceil$. 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 $t_B B(s_1) = t_E B(s_1) - \Delta = t_B Cu(s_{-1})$. Substituting $t_B Cu(s_{-1}) = t + \epsilon - \Delta$, since we consider γ the upper bound for curing time, then $t_E Cu(s_{-1}) = t + \epsilon - \Delta + \gamma$. So finally, $MaxSil(t, t + T_r) = \lceil \frac{t_E Cu(s_1) - T_r + \delta}{\Delta} \rceil = \lceil \frac{\gamma - \Delta + \epsilon - T_r + \delta}{\Delta} \rceil$, then since there can not be a negative result $MaxSil(t, t + 2\delta) = \mathcal{R}(\lceil \frac{\gamma - \Delta + \epsilon - T_r + \delta}{\Delta} \rceil)$. □

Lemma 6 *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 $MaxSil(t, t + T_r) = \lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil \Delta + \gamma - \delta}{\Delta} \rceil$.*

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 $Sil(t, t + T_r)$ if $t_E Cu(s_{-j}) > T_r - \delta$. Considering that $t_E Cu(s_{-j}) - t_E Cu(s_{-j-1}) = \Delta$ then the number of servers in a silent state at t is $MaxSil(t, t+T_r) = \lceil \frac{t_E Cu(s_1) - T_r + \delta}{\Delta} \rceil$. 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 $t_B B(s_b)$. Thus given an arbitrarily small number $\epsilon > 0$, let $t_B B(s_b) = t + T_r - \epsilon$. By construction we know that $t_B B(s_1) = t_E B(s_1) - \Delta = t_B Cu(s_{-1})$ and $t_B B(s_1) = t_B B(s_b) - \lceil \frac{T_r}{\Delta} \rceil \Delta$ (cf. Lemma 1). Substituting and considering that $t_E Cu(s_{-1} = t_B Cu(s_{-1}) + \gamma)$ we get the following: $t_E Cu(s_{-1} = t + T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil + \gamma$. Finally $MaxSil(t, t+T_r) = \lceil \frac{t_E Cu(s_1) - T_r + \delta}{\Delta} \rceil = \lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil + \gamma - T_r + \delta}{\Delta} \rceil$, then since there can not be a negative result, $MaxSil(t, t + T_r) = \lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil \Delta + \gamma - T_r + \delta}{\Delta} \rceil$. □

Lemma 7 *Let us consider a time interval $[t, t + T_r], T_r \geq 2\delta$ then in the $(\Delta S, CAM)$ model. $minC\tilde{B}C = \mathcal{R}(\lceil \frac{T_r}{\Delta} \rceil - \lceil \frac{\delta}{\Delta} \rceil) + \mathcal{R}(\lceil \frac{T_r - \gamma - T_r + \delta}{\Delta} \rceil)$.*

Proof By definition $minC\tilde{B}C(t, t + T_r) = minC\tilde{B}(t, t + T_r) + minB\tilde{C}(t, t + T_r)$.
- $minC\tilde{B}(t, t + T_r)$ 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 if belongs to $Co(t, t + \delta)$, it follows that servers in $\tilde{B}(t, t + \delta)$ do not correctly reply. Thus, $minC\tilde{B}(t, t + T_r) = Max\tilde{B}(t, t + T_r) - Max\tilde{B}(t, t + \delta)$. It may happen that $Max\tilde{B}(t, t + T_r) < Max\tilde{B}(t, t + T_r - \delta)$, but obviously there can no be negative servers, so we consider only non negative values, $minC\tilde{B}(t, t + T_r) = \mathcal{R}(Max\tilde{B}(t, t + T_r) - Max\tilde{B}(t, t + \delta))$.

- $min\tilde{BC}(t, t + 2\delta)$ is the minimum number of servers that incorrectly reply and then become correct in time that the correct reply is delivered. A server is able to correctly reply if it is correct before $t + T_r - \delta$ (the reply message needs at most δ time to be delivered). Thus we are interested in servers that are affected by a mobile agent up to $t + T_r - \gamma - \delta$. For (Δ, 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 $t = t_E B(s_1) - \epsilon$. In the time interval $[t, t + T_r - \gamma - \delta]$ the number of the mobile agent "jumps" is given by $\lceil \frac{T_r - \gamma - \delta}{\Delta} \rceil$ Trivially, we can not have a negative number, so it becomes $\mathcal{R}(\lceil \frac{T_r - \gamma - \delta}{\Delta} \rceil)$. Summing up $minC\tilde{B}C = \mathcal{R}(\lceil \frac{T_r}{\Delta} \rceil - \lceil \frac{\delta}{\Delta} \rceil) + \mathcal{R}(\lceil \frac{T_r - \gamma - \delta}{\Delta} \rceil)$, which concludes the proof. □

Lemma 8 *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 $min\tilde{CB} = \lceil \frac{T_r - \epsilon - \delta}{\Delta} \rceil$ otherwise $min\tilde{CB} = \mathcal{R}(Max\tilde{B}(t, t + T_r) - Max\tilde{B}(t, t + T_r - \delta))$.*

Proof $min\tilde{CB}(t, t + T_r)$ 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 $B(t + T_r - \delta, t + T_r)$, which means that servers in $B(t + T_r - \delta, t + T_r)$ are in $Co(t, t + \delta)$ (considering the scenario S^*). Thus, considering that in the (Δ, CUM) model we consider $t_B B(s_b) = t + T_r - \epsilon$ ($\epsilon > 0$ and arbitrarily small) then we consider the maximum number of "jumps" there could be in the time interval $[t + \delta, t + T_r - \epsilon]$. Thus $min\tilde{CB}(t, t + T_r) = \lceil \frac{t + T_r - \epsilon - t - \delta}{\Delta} \rceil = \lceil \frac{T_r - \epsilon - \delta}{\Delta} \rceil$. 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, $min\tilde{CB} = \mathcal{R}(Max\tilde{B}(t, t + T_r) - Max\tilde{B}(t, t + T_r - \delta))$, concluding the proof. □

Lemma 9 *Let us consider a time interval $[t, t + T_r], T_r \geq 2\delta$ then in the $(\Delta S, CUM)$ model then if $maxCu(t) > 0$ $minC\tilde{B}C = \lceil \frac{T_r - \epsilon - \delta}{\Delta} \rceil + \mathcal{R}(\lceil \frac{T_r}{\Delta} \rceil - \lceil \frac{\gamma - \delta}{\Delta} \rceil) + (MaxCu(t) - MaxSil(t, t + T_r))$, otherwise $minC\tilde{B}C$ assumes the same values as in the $(\Delta S, CAM)$ case.*

Proof By definition $minC\tilde{B}C(t, t + T_r) = min\tilde{CB}(t, t + T_r) + min\tilde{BC}(t, t + T_r)$. From Lemma 8, if $maxCu(t) > 0$ or $\Delta > \gamma$ then in the $(\Delta S, CUM)$ model $min\tilde{CB} = \lceil \frac{T_r - \epsilon - \delta}{\Delta} \rceil$ otherwise $min\tilde{CB} = \mathcal{R}(Max\tilde{B}(t, t + T_r) - Max\tilde{B}(t, t + T_r - \delta))$.
$min\tilde{BC}(t, t + T_r)$ 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 δ 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 $\lceil \frac{\gamma + \delta}{\Delta} \rceil$. Thus, among the Byzantine servers (i.e., $Max\tilde{B}(t, t + T)$) we consider servers not affected in the time interval $[t + T_r - \gamma + \delta, t + T_r]$. In other words such servers have γ time to became correct and δ time to reply before the end of the operation. Thus $Max\tilde{B}(t, t + T_r) - Max(t + T_r - \gamma + \delta, t + T_r)$. Again we can not have a negative number, so it becomes $\mathcal{R}(\lceil \frac{T_r}{\Delta} - \frac{\gamma - \delta}{\Delta} \rceil)$. 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 $min\tilde{BC}(t, t + 2\delta) = (MaxCu(t) - MaxSil(t, t + T_r))$. Summing up if $maxCu(t) > 0$ or $\Delta > \gamma$, then $minC\tilde{B}C = \lceil \frac{T_r - \epsilon - \delta}{\Delta} \rceil +$

Table 3: Values for a general read() operation that terminates after T_r time.

	$Max\tilde{B}(t, t+T_r)$	$MaxCu(t)$	$MaxSil(t, t+T_r)$
$(\Delta S, CAM)$	$\lceil \frac{T_r}{\Delta} \rceil + 1$	$\mathcal{R}(\lceil \frac{\gamma - \Delta + \epsilon}{\Delta} \rceil)$	$\mathcal{R}(\lceil \frac{\gamma - \Delta + \epsilon - T_r + \delta}{\Delta} \rceil)$
$(\Delta S, CUM)$	$\lceil \frac{T_r}{\Delta} \rceil + 1$	$\mathcal{R}(\lceil \frac{T_r - \epsilon - \lceil \frac{T_r}{\Delta} \rceil \Delta + \gamma}{\Delta} \rceil)$	$\lceil \frac{\gamma + \delta - \epsilon - \lceil \frac{T_r}{\Delta} \rceil \Delta}{\Delta} \rceil$

	$minC\tilde{B}C(t, t+T_r)$
$(\Delta S, CAM)$	$\mathcal{R}(\lceil \frac{T_r}{\Delta} \rceil - \lceil \frac{\delta}{\Delta} \rceil) + \mathcal{R}(\lceil \frac{T_r - \gamma - \delta}{\Delta} \rceil)$
$(\Delta S, CUM)$	$\lceil \frac{T_r - \epsilon - \delta}{\Delta} \rceil^a + \mathcal{R}(\lceil \frac{T_r}{\Delta} \rceil - \lceil \frac{\gamma + \delta}{\Delta} \rceil) + (MaxCu(t) - MaxSil(t, t+T_r))$

[a] if $maxCu(t) > 0$ otherwise is the same value of $minC\tilde{B}C(t, t+T_r)$ in the $(*, CAM)$ model

$\mathcal{R}(\lceil \frac{T_r}{\Delta} \rceil - \lceil \frac{\gamma - \delta}{\Delta} \rceil) + (MaxCu(t) - MaxSil(t, t+2\delta))$, otherwise $minC\tilde{B}C$ assumes the same values as in the $(\Delta S, CAM)$ model, which concludes the proof. □

In Table 3 are reported all the results found so far for $(\Delta S, *)$ models.

Such results have been proved considering $f = 1$. Extending such results to scenario for $f > 1$ is straightforward in the $(\Delta S, *)$ model. The extension to $f > 1$ in the $(ITB, *)$ and $(ITU, *)$ 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 $*$ to indicate that the result holds for $*$ assuming consistently the value CAM or CUM.

Lemma 10 *Let $n_{*_{LB}} \leq \alpha_*(\Delta, \delta, \gamma)f$ be the impossibility results holding in the $(\Delta S, *)$ model for $f = 1$. If there exists a tight protocol $\mathcal{P}_{reg}$ 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 $\mathcal{P}_{reg}$ solving the safe register for $n \geq \alpha_* f(\Delta, \delta, \gamma) + 1$ 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 $n = \alpha_*(\Delta, \delta, \gamma)f$ (meaning that $\mathcal{P}_{reg}$ does not exist). We can say that those worst scenarios are equivalent in terms of replicas, i.e., for each wsr_k is it possible to build an impossibility run if $n = \alpha_*(\Delta, \delta, \gamma)$ but $\mathcal{P}_{reg}$ works if $n = \alpha_*(\Delta, \delta, \gamma) + 1$ (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 is it possible to build an impossibility run if $n = \alpha_*(\Delta, \delta, \gamma)f$, i.e., $\alpha_*(\Delta, \delta, \gamma) - 1$ non Byzantine servers are not enough to cope with 1 Byzantine server, then it is straightforward that $\alpha_*(\Delta, \delta, \gamma) - f$ 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 $\mathcal{P}_{reg}$ 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. □

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

Table 4: Parameters for $\mathcal{P}_{Reg}$ Protocol in the (ITB, CAM) and (ITB, CUM) models, minimum number of replicas, and minimum expected occurrence of correct values.

(CAM, ITB)		
$2\delta \leq \Delta < 3\delta$	n_{CAM}	4f+1
	$\#reply_{CAM}$	2f+1
$\delta \leq \Delta < 2\delta$	n_{CAM}	6f+1
	$\#reply_{CAM}$	3f+1

(CUM, ITB)		
$2\delta \leq \Delta < 3\delta$	n_{CUM}	7f+1
	$\#reply_{CUM}$	4f+1
$\delta \leq \Delta < 2\delta$	n_{CUM}	12f+1
	$\#reply_{CUM}$	7f+1

Corollary 6 *Let $n_{*_{LB}} \leq \alpha_*(\Delta, \delta, \gamma)f$ 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.*

4 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.

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, δ and Δ (see Table 4). 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 Δ_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, it just replies to ECHO_REQ() messages. Otherwise, if a server s_i is in a cured state it first cleans its local variables and $broadcast$ to other servers a request. Then, after 2δ 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 δ time period. This is done invoking the awareAll function that broadcasts a default value $\perp$ after δ 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 2δ 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 choses a random number to associate

Table 5: Parameters for $\mathcal{P}_{Rreg}$ Protocol for the (ITB, CUM) model.

$k = \lceil \frac{2\delta}{\Delta} \rceil \geq 1$	$n_{CUM} \geq (5k+2)f+1$	$\#reply_{CUM} \geq (3k+1)f+1$	$\#echo_{CUM} \geq (3k)+1f$
$k=2$	$12f+1$	$7f+1$	$6f+1$
$k=1$	$7f+1$	$4f+1$	$4f+1$

to such particular maintenance() operation instance [4], broadcast the ECHO_REQ() message and waits 2δ before restart ing the operation. When there is a value whose occurrence overcomes the $\#echo_{CUM}$ 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 2δ 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 δ 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 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 2δ 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 (see $\#reply_{C*M}$ from Table 4) 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 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.

4.1 $\mathcal{P}_{reg}$ in the $(ITB, CAM)model$

The protocol $\mathcal{P}_{reg}$ for the (ITB, CAM) model is described in Figures 8 - 10, which present the maintenance(), write(), and read() operations, respectively.

Local variables at client c_i. Each client c_i maintains a set $reply_i$ 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

[4] 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.

insert$(V_i, \langle v_k, sn_k \rangle)$ 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.

- $pending_read_i$: set variable used to collect identifiers of the clients that are currently reading.
- $cured_i$: 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 $echo_read_i$: two sets used to collect information propagated through ECHO messages. The first one stores tuple $\langle j, \langle v, sn \rangle \rangle$ 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 read().
- $curing_i$: 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 $\langle v, sn \rangle$, 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 $\langle v, sn \rangle$ occurring at least $\#reply_{CAM}$ times (see Table **??**). 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 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 ECHO_REQ() messages. Otherwise, if a server s_i is in a cured state it first cleans its local variables and $broadcast$ to other servers an echo request then, after 2δ 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 $\langle v, sn \rangle$ 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 $\langle v, sn \rangle$. 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 δ time period. This is done invoking the awareAll function that broadcast a default value $\perp$ after δ 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 δ 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 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 $curing_i$).

```
function awareAll():
(01) broadcast ECHO(i, ⊥)
(02) wait(δ);
(03) broadcast ECHO(i, ⊥)
---------------------------------------------------------------
operation maintenance() executed while (TRUE) :
(04) cured_i ← report_cured_state();
(05) if (cured_i) then
(06)    cured_i ← false;
(07)    curing_state_i ← true;
(08)    V_i ← ∅; echo_vals_i ← ∅; pending_read_i ← ∅;curing_i ← ∅;
(09)    broadcast ECHO_REQ(i);
(10)    awareAll();
(11)    wait(2δ);
(12)    delete_cured_values(echo_vals);
(13)    insert(V_i, select_three_pairs_max_sn(echo_vals_i));
(14)    for each (j ∈ (curing_i)) do
(15)            send ECHO (i, V_i) to s_j;
(16)    endFor
(17)    curing_state_i ← false;
(18) endIf
---------------------------------------------------------------
when ECHO (j, V_j) is received:
(19)  for each (⟨v, sn⟩ ∈ V_j do
(20)          echo_vals_i ← echo_vals_i ∪ ⟨v, sn⟩_j;
(21)  endFor
---------------------------------------------------------------
when ECHO_REQ (j) is received:
(22)  curing_i ← curing_i ∪ j;
(23)  if (V_i ≠ ∅)
(24)    send ECHO(i, V_i);
(25)  endif
```

Figure 8: $\mathcal{A}_M$ algorithm implementing the maintenance() operation (code for server s_i) in the (ITB, CAM) model.

The read() **operation.** When a client wants to read, it broadcasts a READ() request to all servers and waits 2δ 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 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 $pending_read_i$ 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 REPLY() message carries the set V_i.

When a READ_ACK(j) message is delivered, c_j identifier is removed from both $pending_read_i$ set as it does not need anymore to receive updates for the current read() operation.

4.2 $\mathcal{P}_{reg}$ in the $(ITB, CUM)model$

$\mathcal{P}_{reg}$ **Detailed Description** The protocol $\mathcal{P}_{reg}$ for the (ITB, CUM) model is described in Figures 11 - 13, which present the maintenance(), write(), and read() operations, respectively. Table 5 reports the parameters for the protocol. In particular n_{CUM} is the bound on the number of servers, $\#reply_{CUM}$ is minimum number of occurrences from different servers of a value to be accepted as a reply during a read() operation and $\#echo_{CUM}$ is the minimum number of occurrences from different servers of a value to be accepted during the maintenance() operation.

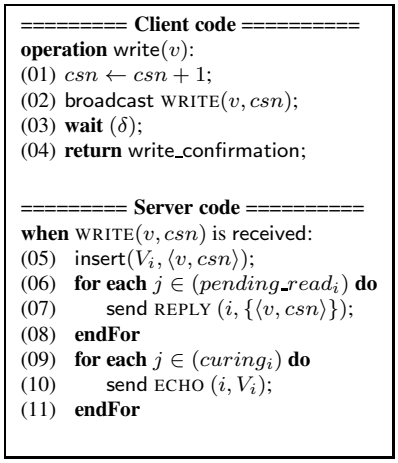

Figure 9: $\mathcal{A}_W$ algorithm implementing the write(v) operation in the (ITB, CAM) model.

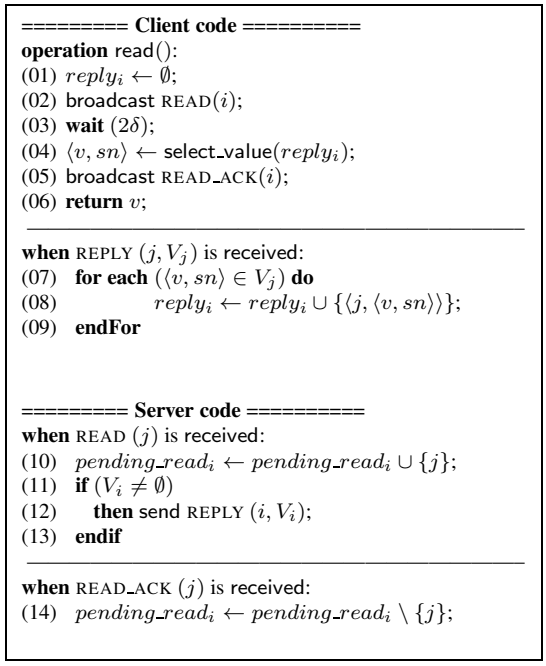

Figure 10: $\mathcal{A}_R$ algorithm implementing the read() operation in the (ITB, CAM) model.

Local variables at client c_i. Each client c_i maintains a set $reply_i$ 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 3 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.

- V_{safe_j}: this set has the same characteristic as V_j. The insert$(V_{safe_i}, \langle v_k, sn_k \rangle)$ function places the new value in V_{safe_i} according to the incremental order and if dimensions exceed 3 then it discards from V_{safe_i} the value associated to the lowest sn.

- W_i: is the set where servers store values coming directly from the writer, associating to it a timer, $\langle v, sn, timer \rangle$. Values from this set are deleted when the timer expires or has a value non compliant with the protocol.

- $pending_read_i$: set variable used to collect identifiers of the clients that are currently reading.

- $echo_vals_i$ and $echo_read_i$: two sets used to collect information propagated through ECHO messages. The first one stores tuple $\langle j, \langle v, sn \rangle \rangle$ 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 read().

- $curing_i$: 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_three_pairs_max_sn$(echo_vals_i)$: this function takes as input the set $echo_vals_i$ and returns, if they exist, three tuples $\langle v, sn \rangle$, such that there exist at least $\#echo_{CUM}$ occurrences in $echo_vals_i$ of such tuple. If more than three of such tuples 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 $\langle v, sn \rangle$ occurring occurring at least $\#reply_{CUM}$ times. If there are more pairs with the same occurrence, it returns the one with the highest sequence number.

- conCut(V_i, V_{safe_i}, 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 $V_{safe_i} \circ V_i \circ W_i$, 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 $V_{safe_i} = \{\langle v_b, 2 \rangle, \langle v_d, 4 \rangle, \langle v_f, 5 \rangle\}$ and $W_i = \emptyset$, then the returned set is $\{\langle v_c, 3 \rangle, \langle v_d, 4 \rangle, \langle v_f, 5 \rangle\}$.

The maintenance() **operation.** Such operation is executed by servers every 2δ times. Each time s_i resets its variables, except for W_i (that is continuously checked by the function timerCheck()) and the content of V_{safe_i}, which overrides the content of V_i, before to be reset. Then s_i choses a random number to associate to such particular maintenance() operation instance [5], broadcast the ECHO_REQ() message and waits 2δ before to restart the operation. In the meantime ECHO() messages are delivered and stored in the $echo_vals_i$ set. When there is value v whose occurrence overcomes the $\#echo_{CUM}$ threshold, such value is stored in V_{safe_i} and a REPLY() message with v is sent to current reader clients (if any).

[5] 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.

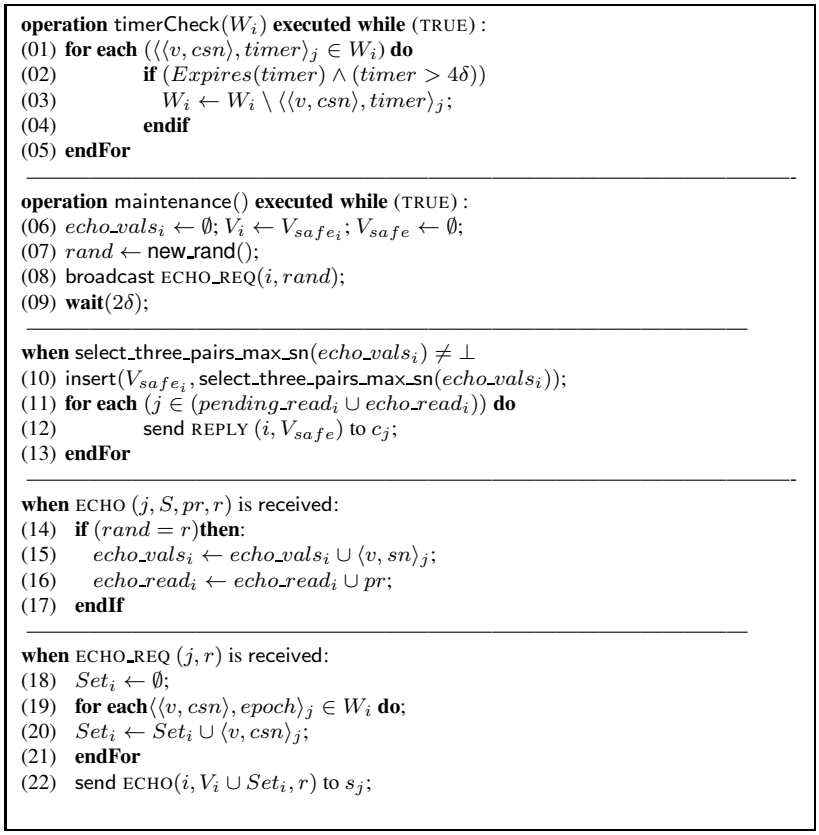

Figure 11: $\mathcal{A}_M$ algorithm implementing the maintenance() operation (code for server s_i) in the (ITB, CUM) model.

Notice that, contrarily to all the previous models, servers are not aware about their failure state and do not synchronize the maintenance() operation with each other. The first consequence is a 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 2δ 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.** 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 δ time units (the maximum message transfer delay) before returning.

When a server s_i delivers a WRITE message, it updates W_i, associating to such value a timer 4δ. 4δ it is a consequence of the double 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 4δ, which is enough for our purposes. On the other side a cured servers keeps a value (not necessarily coming from a write() operation) no more than the time it is in a cured state, 4δ, which is safe. After storing v in W_i, such value is inserted in REPLY() message to all clients that are currently reading (clients in $pending_read_i$) to notify them about the concurrent write() operation and to any 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 2δ 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 an acknowledgement message to servers to

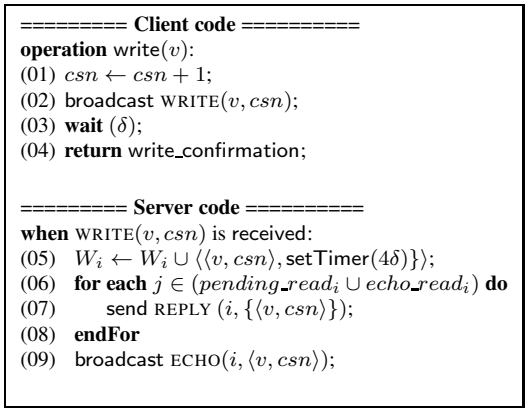

Figure 12: $\mathcal{A}_W$ algorithm implementing the write(v) operation in the (ITB, CUM) model.

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 $pending_read_i$ 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 REPLY() message is carried the result of conCut(V_i, V_{safe_i}, W_i). In this case, if the server is correct then V_i contains valid values, and V_{safe_i} contains valid values by construction, since it comes from values sent during the current maintenance(). If the server is cured, then V_i and W_i may contain any value. Finally, s_i forwards a READ_FW message to inform other servers about c_j read request. This is useful in case some server missed the READ(j) message as it was affected by mobile Byzantine agent when such message has been delivered.

When a READ_ACK(j) message is delivered, c_j identifier is removed from both $pending_read_i$ set as it does not need anymore to receive updates for the current read() operation.

5 Correctness

5.1 Correctness (ITB, CAM)

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

Lemma 11 *If a correct client c_i invokes* write(v) *operation at time t then this operation terminates at time $t + \delta$.*

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 9). □

Lemma 12 *If a correct client c_i invokes* read() *operation at time t then this operation terminates at time $t + 2\delta$.*

Proof The claim follows by considering that a read() returns a value to the client after 2δ time, independently of the behavior of the servers (see lines 03-06, Figure 10). □

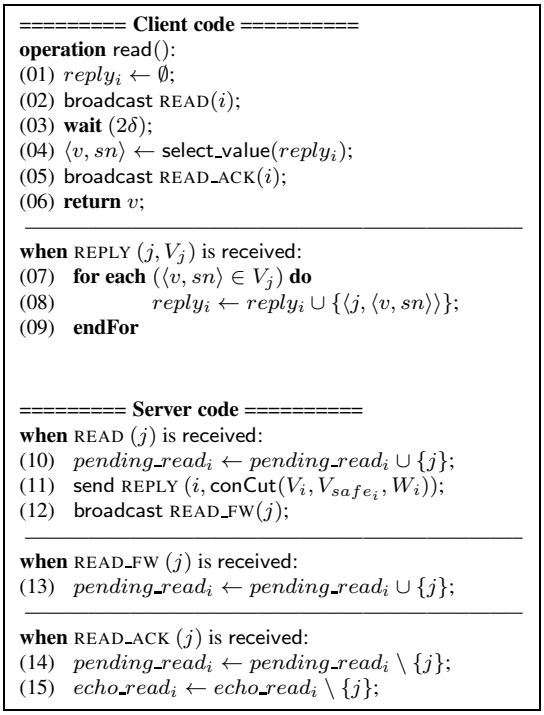

Figure 13: $\mathcal{A}_R$ algorithm implementing the read() operation in the (ITB, CUM) model.

Theorem 2 (Termination) *If a correct client c_i invokes an operation, c_i returns from that operation in finite time.*

Proof The proof follows from Lemma 11 and Lemma 12. □

Validity property is proved with the following steps:

- 1. 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 $\#reply - f = \#echo$ 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_i leaves it, then ma_i movement are aligned to such operation, this agent contribution is $\frac{2\delta}{\Delta} = k$. All the others $f-1$ mobile agent are not aligned, thus their contribution is $Max\tilde{B}(t, t+2\delta) = k+1$. Thus there are $k + (k+1) \times (f-1)$ Byzantine servers during the 2δ time maintenance() operation.

Lemma 13 (Step 1) *Let $T_i = t$ be the time at which mobile agent ma_i leave s_c. Let v be the value stored at $\#echo_{CAM}$ servers $s_j \notin B(t, t+\delta) \wedge s_j \in Co(t+\delta)$, $v \in V_j \forall s_j \in Co(t+\delta)$. At time $t + 2\delta$, at the end of the* maintenance(), *v is returned to s_c by the function* select_d_pairs_max_sn($echo_vals_c$).

Proof The proof follows considering that:

- the maintenance() employs a request-reply pattern and during such operation, by hypothesis, there are $\#echo_{CAM}$ 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 $\#echo_{CAM}$ 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_c 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 δ time after. Thus by time $t+2\delta$ server running the maintenance removes from $echo_vals$ 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 $echo_vals_c$ set.

$\#echo_{CAM} = (k+1)f > (k+1)f-1$ thus Byzantine servers can not force the select_d_pairs_max_sn($echo_vals_c$) function to return a not valid value and select_d_pairs_max_sn($echo_vals_c$) returns v that occurs $\#reply_{CAM}$ times, concluding the proof. □

Lemma 14 (Step 2.) *Let op_W be a* write(v) *operation invoked by a client c_k at time $t_B(op_W) = t$ then at time $t + \delta$ there are at least $\#reply_{CAM}$ 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 $n-(\frac{k}{2}+1)f = 2(k+1)f+1-(\frac{k}{2}+1)f = (k+\frac{k}{2}+1)f+1$ servers s_j that being either cured or correct, execute code in Figure 9, line 05, inserting v in V_j. Finally, $(k + \frac{k}{2} + 1)f + 1 > (k + 1)f + 1 = \#reply_{CAM}$ concluding the proof.

□

For simplicity, for now on, given a write() operation op_W we call $t_B(op_W) + \delta = t_{wC}$ the **completion time** of op_W, the time at which there are at least $\#reply_{CAM}$ servers storing the value written by op_W.

Lemma 15 (Step 3.) *Let op_W be a* write() *operation occurring at $t_B(op_W) = t$ and let v be the written value and let t_{wC} be its completion time. Then if there are no other* write() *operations after op_W, the value written by op_W is stored by all correct servers forever.*

Proof Following the same reasoning as Lemma 14, 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 9, line 05, inserting v in V_j. Now let us consider the following:

- Let $B_1 = \tilde{B}(t, t + \delta)$ be the set containing the $(\frac{k}{2} + 1)f$ Byzantine servers during $[t, t + \delta]$, so that there are $(2k + 1)f + 1 - \frac{k}{2} = (k + \frac{k}{2} + 1)f + 1 \geq \#reply_{CUM}$ 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 $\#reply_{CAM}$ non faulty servers storing v, being $\#reply_{CAM} > \#echo_{CAM}$, for Lemma 13 at the end of the maintenance() operation, by time $t + 3\delta$, those servers obtain v a result of select_d_pairs_max_sn($echo_vals$) invocation, whose is stored in V since there are no other write() operation and since v has the highest associated sequence number.

- Let $B_2 = \tilde{B}(t + \delta, t + 2\delta)$ be the set containing Byzantine servers in the next δ period. Those servers are $\frac{k}{2}f$ (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 $(k + \frac{k}{2} + 1)f + 1 - \frac{k}{2}f = (k + 1)f + 1 = \#reply_{CAM}$ non faulty servers storing v;

  – there are $(\frac{k}{2})f$ Byzantine servers in B_2 that begin the maintenance() operation during $[t+\delta, t+2\delta]$ time interval. There are $\#reply_{CAM}$ non faulty servers storing v, being $\#reply_{CAM} > \#echo_{CAM}$, for Lemma 13 at the end of the maintenance() operation, by time $t + 4\delta$, those servers, get v invoking select_d_pairs_max_sn($echo_vals$), whose is stored in V since there are no other write() operation and since v has the highest associated sequence number.

- Let $B_3 = \tilde{B}(t+2\delta, t+3\delta)$ be the set containing Byzantine servers in the next δ period. Those servers are $\frac{k}{2}f$. At $t + 3\delta$ there are $(k + 1)f + 1 - \frac{k}{2}f < \#reply_{CAM}$ 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 $(k + 1)f + 1 - \frac{k}{2}f + \frac{k}{2}f = \#reply_{CAM}$ 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 16 (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() *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 $t_E w_k$ be its completion time. Then the register stores v (there are at least $\#reply_{CAM}$ correct servers storing it) up to time at least $t_B W_{k+3}$.*

Proof The proof simply follows considering that:

- for Lemma 15 if there are no more write() operation then v, after t_{wC}, is in the register forever.
- any new written value is store in an ordered set V (cf. Figure 9 line 05) 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 3 (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 2δ, 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 $n - \frac{k}{2} - 1 \geq \#reply_{CAM}$ 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 $t_E(op_W) \leq t_B(op_R)$ and let v be the last written value. For Lemma 15 after the write completion time t_Cw there are $\#reply_{CAM}$ non faulty servers storing v. Since $t_B(op_R) + \delta \geq t_Cw$, then there are $\#reply_{CAM}$ non faulty servers replying with v (Figure 10, lines 11-12). 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 16 the last written value before $t_B(op_R)$ is still present in $\#reply_{CAM}$ non faulty servers. Thus at least the last written value is returned.
To conclude, for Lemma 1, during the read() operation there are at most $(k+1)f$ Byzantine servers, being $\#reply_{CAM} > (k+1)f$ then Byzantine servers may not force the reader to read another or older value and even if an older values has $\#reply_{CAM}$ occurrences the one with the highest sequence number is chosen. □

Theorem 4 *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 δ be the upper bound on the communication latencies in the synchronous system. If $n = n_{CAM}$ according to Table* **??** *then $\mathcal{P}_{reg}$ 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 2 and Theorem 3 and considering $\Delta = 1$ in the case of (ITU, CAM) model. □

Lemma 17 *Protocol $\mathcal{P}_{reg}$ for $\delta \leq \Delta < 3\delta$ is tight with respect to $\gamma \leq 2\delta$.*

Proof The proof follows from Theorem 4 and Theorem 1, i.e., upper bound and lower bound match. In particular Lower bounds are computed using the values in Table 3 to compute $n_{CAM_{LB}}$ as defined in Table 2 for $\gamma \leq 2\delta$ (cf. Lemma 13). □

5.2 Correctness (ITB, CUM)

To prove the correctness of $\mathcal{P}_{reg}$ we demonstrate that the termination property is satisfied i.e, that read() and write() operations terminates. For the validity property we follow te same four steps as defined in Section 5.1.

Lemma 18 *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 after δ time, independently of the behavior of the servers (see lines 03-04, Figure 12). □

Lemma 19 *If a correct client c_i invokes* read() *operation at time t then this operation terminates at time $t + 2\delta$.*

Proof The claim simply follows by considering that a read() returns a value to the client after 2δ time, independently of the behaviour of the servers (see lines 12-15, Figure 13). □

Theorem 5 (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 18 and Lemma 19. □

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

Lemma 20 *Let $[t, t + 2\delta]$ be a generic interval, then there are always at least $\#reply_{CUM}$ 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 2). Since does exist a tight protocol $\mathcal{P}$ solving a regular register in the $(\Delta S, CAM)$ model, then for Lemma 10, is it possible to apply values from Table 3 to compute the minimum number of correct replies during the considered time interval, substituting values in each case the result is always at least $\#reply_{CUM}$. □

Lemma 21 (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 $s_j \notin B(t, t + \delta)$, $v \in V_j \forall s_j \notin B(t, t + \delta)$. 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($echo_vals_c$).

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 can be 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 $(2k)f$ servers that were Byzantine in $[t - 4\delta, t]$ time period, thus they could be still in a cured state [6].
- 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.

$\#echo_{CUM} = (3k)f+1 > 3kf$ thus Byzantine servers can not force the select_three_pairs_max_sn($echo_vals_c$) function to return a not valid value so it returns v that occurs $\#reply_{CUM}$ 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 2δ time. Contrarily to all the previous model, the maintenance() operation is triggered each 2δ. Thus a mobile agent, just before to leave could leave s_c with the timer just reset and garbage in the $echo_set_c$

[6] We prove hereafter that $\gamma \leq 4\delta$, but to prove it we have first to prove that the maintenance() lasts 2δ time.

and V_c sets, which does not allow s_c to correctly terminate the operation. Thus s_c has to wait 2δ before to effectively starts 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 7 *Protocol $\mathcal{P}$ implements a* maintenance() *operation that implies $\gamma \leq 4\delta$.*

Lemma 22 (Step 2.) *Let op_W be a* write(v) *operation invoked by a client c_k at time $t_B(op_W) = t$ then at time $t + \delta$ there are at least $n - 2f > \#reply_{CUM}$ non faulty servers s_i such that $v \in W_i$ (so that when s_i invokes* conCut(V_i, V_{safe_i}, W_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 4δ, after that the value expires. For Lemma 1 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 12 line 05 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 op_W we call $t_B(op_W) + \delta = t_{wC}$ the **completion time** of op_W, the time at which there are at least $\#reply_{CUM}$ servers storing the value written by op_W.

Lemma 23 (Step 3.) *Let 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 op_W is stored by all correct servers forever (i.e., $v \in$ conCut(V_i, V_{safe_i}, W_i)).*

Proof From Lemma 22 at time t_{wC} there are at least $n - 2f > \#reply_{CUM}$ non faulty servers s_j such that $v \in W_i$. For sake of simplicity let us consider Figure 14. Let us consider that:

- for Lemma 22, 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($echo_vals_i$) 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_{safe_i} (cf. Figure 11 line 10), thus such value is present in the ECHO() message replies for the next 2δ 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 $n - (2k + 1) = (3k + 1)f + 1 = \#reply_{CUM} \geq \#echo_{CUM}$;
- 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)s_{10} and s_{11} respectively:
  - $t_B(op_M) \in [t', t' + \delta]$ (cf. s_{10} Figure 14): s_{10} starts op_{M_1} before $t' + \delta$, let us name it server type A. This means that $t_B(op_{M_{-1}}) + \delta < t' - \delta$, thus for Lemma 21, at the end of the operation $v \in V_{safe_{10}}$ and during op_{M_1} $v \in V_{10}$;
  - $t_B(op_M) \in [t' + \delta, t' + 2\delta]$ (cf. s_{11} Figure 14): s_{11} starts op_{M_1} after $t' + 2\delta$ let us name it server type B. This means that $t_B(op_{M_{-1}}) + \delta > t'$, thus at the end of the operation we can not say that $v \in V_{safe_{10}}$ but at least during $op_{M_{-1}}$ $v \in V_{11}$.

If all non faulty servers are type A, during op_{M_1} 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 $v \in V_{safe_b}$ and during op_{M_1} $v \in V_b$. During such operation both servers type A and type B have $vinV$. Again, if there are enough occurrences of v, the operation ends with $v \in V_{safe_b}$. It follows that servers type A, when run op_{M_1} 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, s_5 and s_6. At the same time there is server s_0, type A, that terminate its maintenanace() with $v \in V_{safe_0}$, and thus compensates s_5, allowing s_1, type B, to terminate the maintenanace() 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 24 (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 $\#reply_{CUM}$ correct servers that return it when invoke the function* conCut()*) up to time at least $t_B W_{k+3}$.*

Proof The proof simply follows considering that:

- for Lemma 23 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 11 line 06 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 6 (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 2δ, 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 20 there are at least $\#reply_{CUM}$ 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 23 after the write completion time t_{wC} there are at least $\#reply_{CUM}$ correct servers storing v (i.e., $v \in$ conCut(V_j, V_{safe_j})). Since $t_B(op_R)+2\delta \geq t_C w$, then there are $\#reply_{CUM}$ correct servers replying with v (cf. Lemma 20), 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.

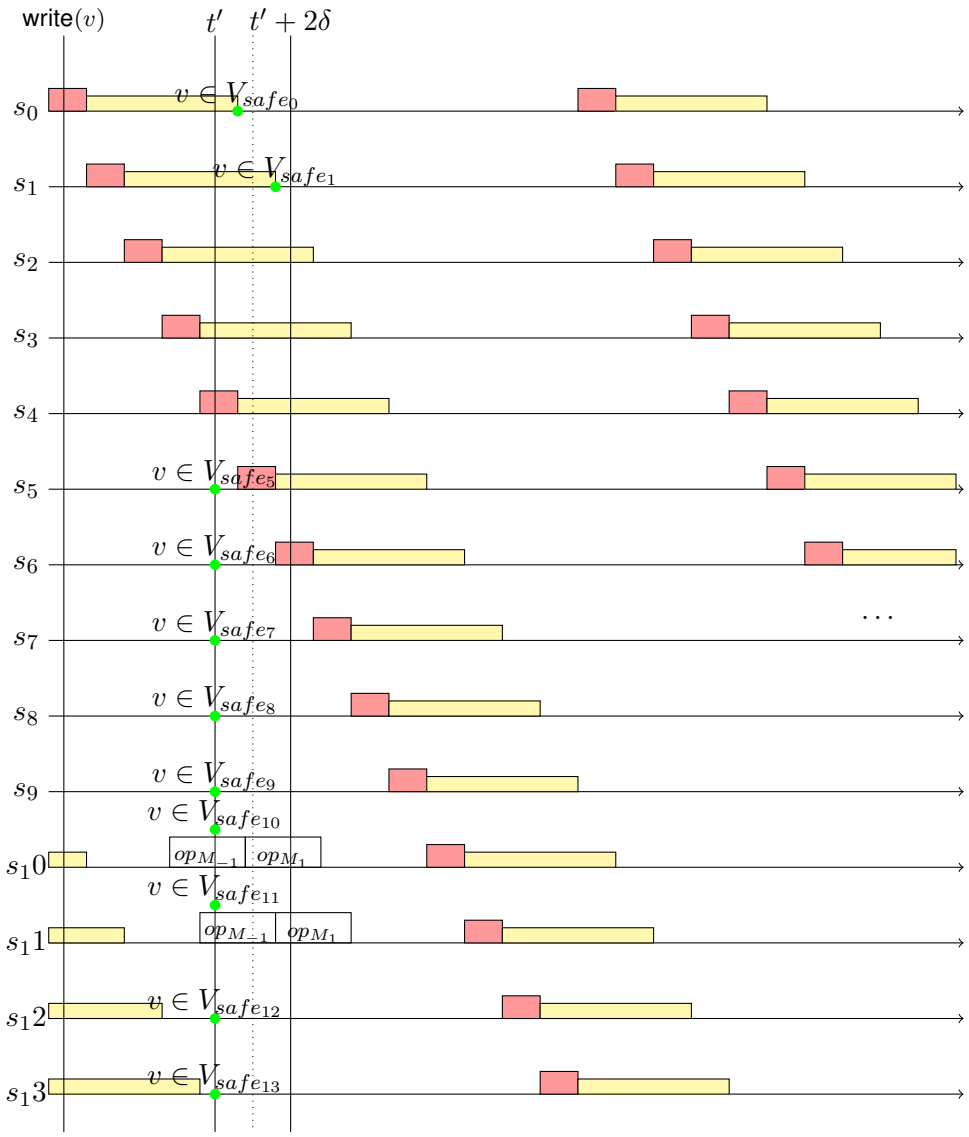

Figure 14: maintenance() operation op_{M_1} analysis after a write() operation, $t' = t + 4\delta$. White rectangles are maintenance() operation run by correct servers. In particular s_{10} runs such operation during the first δ period after t', while s_11 runs it during the second δ period.

- 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 24 the last written value before $t_B(op_R)$ is still present in $\#reply_{CUM}$ correct servers and all of them reply (cf. Lemma 20) thus at least the last written value is returned. To conclude, for Lemma 1, during the read() operation there are at most $(k+1)f$ Byzantine servers and $2k$ cured servers [7], being $\#reply_{CUM} = (3k+1)f + 1 > (3k+1)f$ then Byzantine servers may not force the reader to read another or older value and even if an older values has $\#reply_{CUM}$ occurrences the one with the highest sequence number is returned, concluding the proof. □

Theorem 7 *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 δ be the upper bound on the communication latencies in the synchronous system. If $n \geq (5k+2)f + 1$, then $\mathcal{P}_{reg}$ implements a SWMR Regular Register in the (ITB, CUM) round-free Mobile Byzantine Failure model.*

Proof The proof simply follows from Theorem 5 and Theorem 6. □

Lemma 25 *Protocol $\mathcal{P}_{reg}$ is tight in the (ITB, CUM) model with respect to $\gamma \leq 4\delta$.*

Proof The proof follows from Theorem 7 and Theorem 1, i.e., upper bound and lower bound match. In particular Lower bounds are computed using the values in Table 3 to compute $n_{CUM_{LB}}$ as defined in Table 2 for $\gamma \leq 4\delta$ (cf. Corollary 7). □

6 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 *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

[1] Noga Alon, Hagit Attiya, Shlomi Dolev, Swan Dubois, Maria Potop-Butucaru, and Sébastien Tixeuil. Practically stabilizing SWMR atomic memory in message-passing systems. *J. Comput. Syst. Sci.*, 81(4):692–701, 2015.

[2] N. Banu, S. Souissi, T. Izumi, and K. Wada. An improved byzantine agreement algorithm for synchronous systems with mobile faults. *International Journal of Computer Applications*, 43(22):1–7, April 2012.

[3] Rida A. Bazzi. Synchronous byzantine quorum systems. *Distributed Computing*, 13(1):45–52, January 2000.

[7] Servers where affected in the previous 4δ time period, thus they are still running the two maintenance() operations, that last at most 4δ.

[4] François Bonnet, Xavier Défago, Thanh Dang Nguyen, and Maria Potop-Butucaru. Tight bound on mobile byzantine agreement. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pages 76–90, 2014.

[5] Silvia Bonomi, Antonella del Pozzo, and Maria Potop-Butucaru. Tight self-stabilizing mobile byzantine-tolerant atomic register. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 6:1–6:10, New York, NY, USA, 2016. ACM.

[6] Silvia Bonomi, Shlomi Dolev, Maria Potop-Butucaru, and Michel Raynal. Stabilizing server-based storage in byzantine asynchronous message-passing systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC 2015)*, Donostia San-Sebastian, Spain, July 2015. ACM Press.

[7] Silvia Bonomi, Maria Potop-Butucaru, and Sébastien Tixeuil. Byzantine tolerant storage. In *Proceedings of the International Conference on Parallel and Distributed Processing Systems (IEEE IPDPS 2015)*, Hyderabad, India, May 2015. IEEE Press.

[8] Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Optimal mobile byzantine fault tolerant distributed storage. In *Proceedings of the ACM International Conference on Principles of Distributed Computing (ACM PODC 2016)*, Chicago, USA, July 2016. ACM Press.

[9] H. Buhrman, J. A. Garay, and J.-H. Hoepman. Optimal resiliency against mobile faults. In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS'95)*, pages 83–88, 1995.

[10] J. A. Garay. Reaching (and maintaining) agreement in the presence of mobile faults. In *Proceedings of the 8th International Workshop on Distributed Algorithms*, volume 857, pages 253–264, 1994.

[11] Leslie Lamport. On interprocess communication. part i: Basic formalism. *Distributed Computing*, 1(2):77–85, 1986.

[12] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, October 1998.

[13] Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Minimal byzantine storage. In *Proceedings of the 16th International Conference on Distributed Computing*, DISC '02, pages 311–325, London, UK, UK, 2002. Springer-Verlag.

[14] Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Small byzantine quorum systems. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 374–383. IEEE, 2002.

[15] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91)*, pages 51–59, 1991.

[16] R. Reischuk. A new solution for the byzantine generals problem. *Information and Control*, 64(1-3):23–42, January-March 1985.

[17] T. Sasaki, Y. Yamauchi, S. Kijima, and M. Yamashita. Mobile byzantine agreement on arbitrary network. In *Proceedings of the 17th International Conference on Principles of Distributed Systems (OPODIS'13)*, pages 236–250, December 2013.

[18] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[19] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel & Distributed Systems*, (4):452–465, 2009.