Distributed Computing manuscript No.
(will be inserted by the editor)

Tolerating Permanent and Transient Value Faults

Zarko Milosevic · Martin Hutle · André Schiper

Abstract Transmission faults allow us to reason about permanent and transient value faults in a uniform way. However, all existing solutions to consensus in this model are either in the synchronous system, or require strong conditions for termination, that exclude the case where all messages of a process can be corrupted. In this paper we introduce eventual consistency in order to overcome this limitation. Eventual consistency denotes the existence of rounds in which processes receive the same set of messages. We show how eventually consistent rounds can be simulated from eventually synchronous rounds, and how eventually consistent rounds can be used to solve consensus.

Depending on the nature and number of permanent and transient transmission faults, we obtain different conditions on n, the number of processes, in order to solve consensus in our weak model.

Keywords consensus · transmission faults · arbitrary faults · static and dynamic faults · transient and permanent faults · eventual consistency

1 Introduction

Consensus is probably the most fundamental problem in fault-tolerant distributed computing. It is related to the implementation of state machine replication, atomic broadcast, group membership, etc. The problem is defined over a set of processes Π, where each process $p \in \Pi$ has an initial value v_i, and requires that all processes agree on a common value.

Classical approach: Component fault model. Most research on consensus algorithms is considering *component fault models*, where faults are attached to a component that is either a process or a link. With respect to process/link faults, consensus can be considered with different fault assumptions. On the one end of the spectrum, processes/links can commit so called *benign* faults (processes fail only by crashing and links only loose messages); on the other end, faulty processes/links can exhibit an *arbitrary* behavior. Furthermore, in the context of a component fault model, faults are mainly *permanent* (as opposed to *transient* faults): if a process or link commits a fault, the process/link is considered to be faulty during whole execution. It follows that not all components can be faulty (at most f out of n per run), which is referred to as *static* faults (as opposed to dynamic faults that can affect any component).

Most research on consensus is about tolerating *permanent* and *static* process and/or link faults. While processes and links can be considered faulty, most of the literature considers only process faults. In the context of Byzantine faults, where at most f processes can behave arbitrarily, we can cite the early work of Lamport, Shostak and Pease [22,18] for a synchronous system. Consensus in a partially synchronous system with Byzantine faults is considered in [12,2,21,25]. Byzantine variants of Paxos [15] include [10,19,1,20,17]. Only few authors solve consensus in the synchronous system model where, in addition to Byzantine processes, a small number of links connecting correct processes may be arbitrary faulty during the entire execution of

Zarko Milosevic (corresponding author)
EPFL, 1015 Lausanne, Switzerland
Tel.: +41-21-6936746
Fax: +41-21-6936770
E-mail: zarko.milosevic@epfl.ch

Martin Hutle
Fraunhofer AISEC, Garching, Germany
E-mail: martin.hutle@aisec.fraunhofer.de

André Schiper
EPFL, 1015 Lausanne, Switzerland
E-mail: andre.schiper@epfl.ch

a consensus algorithm [24,28,30]. However, only a very limited number of links can be faulty.

There are two major problems of a priori blaming some component for the failure [26,27,11]. First, it may lead to undesirable consequences if faults are permanent: for example, in the classical Byzantine fault model, where a bounded number of processes can behave arbitrarily (even maliciously), the entire system will be considered faulty even if only one message from each process is received corrupted. Second, when solving consensus, faulty processes are typically not obliged to make a decision or they are allowed to decide differently than correct processes.

Some work in the component fault model has addressed transient and dynamic faults [5]. These papers solve consensus in the hybrid fault model for synchronous systems, where every process is allowed to commit up to f_l^{sa} arbitrary send link failures and experience up to f_l^{ra} arbitrary receive link failures without being considered as arbitrary faulty. Tolerating additional f_s send and f_r receive omissions (i.e., message loss) requires to increase the number of processes by small multiples of f_s and f_r.

Finally, note that when a process q receives a corrupted message from p, it makes no difference for q whether p is faulty and therefore sends a message that was not consistent with the protocol, or the message is corrupted by the link between p and q. Actually, for q these two cases are indistinguishable. Nevertheless, these two cases are not equivalent in the component fault model.

Alternative approach: Transmission fault model. These observations led to the definition of the *transmission fault model* that captures faults without blaming a specific component for the fault [26]. The transmission fault model is well-adapted to *dynamic* and *transient* faults.

Consensus under transmission faults in a synchronous system has been considered initially in [26]. In [11], this work combined with ideas from [14], is extended to non-synchronous systems with only benign transmission faults, leading to the *Heard-Of Model* (HO model). The paper gives several consensus algorithms under benign transmission faults.

In [4], the HO model for benign faults is extended to value faults. There, consensus under transmission faults (both benign and value faults) is solved the first time in a non-synchronous setting. For safety, only the number of corrupted messages is restricted, that is, in each round r of the round based model, every process p receives at most α corrupted messages.[1] However, for liveness, some additional assumptions are necessary, namely *rounds in which some subset of processes does not receive any corrupted messages.*[2] This means that, despite the possibility to handle dynamic and transient value faults in a non-synchronous system, [4] cannot tolerate *permanent faults located at a process p*, where all messages from p might be (always) corrupted.

This raises the following question: is it possible to design a consensus algorithm in the general transmission fault model, with non-synchronous assumptions, that does not require such a strong condition for liveness?

Transmission faults: Our contribution. We give a positive answer to the above question by presenting three consensus algorithms for transmission faults (both benign and value faults) that do not exclude permanent faults.[3] The key insight in achieving this goal is the introduction of the notion of *eventual consistency* that turns out to be fundamental building block for solving consensus under transmission faults. Informally speaking, for round-based algorithms, eventual consistency denotes *the existence of rounds in which processes receive the same set of messages.*

Our three algorithms are inspired by well-known consensus algorithms [12,10,20] for the classical Byzantine fault model [18], which we have adapted to the transmission fault model. All three algorithms require a round in which consistency eventually holds (processes receive the same set of messages). This round is used to bring the system in the univalent configuration, and later rounds are used to "detect" that the system entered a univalent configuration and allows processes to decide. So the key is to achieve eventually consistent rounds. This is the most important contribution. We show that eventually consistent rounds can be simulated from eventually synchronous rounds in the presence of both static and dynamic value faults. The benefits of our approach are the following:

– First, contrary to most of the related work on transmission faults and on the hybrid fault model (where both processes and links can be arbitrary faulty), which considers the synchronous system model, our consensus algorithms can also be used in systems, where synchrony assumptions hold only eventually.

[1] This assumption potentially allows corrupted messages on all links in a run; therefore it models dynamic faults.

[2] This assumption makes sense in the context of transient faults.

[3] We give three algorithms in order to show the generality of our approach.

- Second, contrary to the algorithms in [4], our algorithms can also be used in systems with permanent faults located at a process p, where all messages from p might be (always) corrupted.
- Third, by considering the transmission fault model, the algorithms can tolerate dynamic and transient value faults in addition to only permanent and static faults of the component fault model. As we explain in Section 10, considering (only) transmission faults allows a variety of interpretations, making it possible to apply our algorithms to a variety of system models: partially synchronous system with Byzantine processes, partially synchronous system with Byzantine processes eventually restricted to "symmetrical faults" [29], partially synchronous system with Byzantine processes, where, before stabilization time, in every round processes can receive some (bounded) number of corrupted messages from correct processes, etc.

Remark. Note that despite the similarity in title, [3] addresses a different topic. The paper investigates the possibility of designing protocols that are both self-stabilizing and fault-tolerant in an asynchronous system. A self-stabilizing distributed algorithm is an algorithm that, when started in an arbitrary state, guarantees to converge to a legitimate state and then forever remains in a legitimate state. Solving one-shot consensus, which is the subject of our paper, is impossible in the context of self-stabilization, because a process can start in any state, i.e., its first step can be $decide(v)$, where v is an arbitrary value.

In the model considered in our paper, (transmission) faults do not corrupt the initial configuration (the system starts in a pre-defined state) but may disturb the execution of the protocol. Therefore, the protocols presented in this paper cannot deal with an arbitrary initial configuration.

Organization of the paper. The rest of the paper is structured as follows. We describe the transmission fault model we consider in Section 2. The consensus problem is defined in Section 3. In Section 4 and Section 5 we introduce the communication predicates that we consider in the paper, including eventual consistency. Section 6 shows how to simulate eventual consistency under weak communication predicates, while Section 7 shows how to solve consensus with eventual consistency. In Section 8 we discuss in detail the combination of one of the consensus algorithms and the eventual consistency simulation. As we show in Section 9, eventual consistency can be achieved also directly with authentication. In Section 10 we argue that Byzantine faults and permanent value faults located at a process are indistinguishable, and thus our algorithms also work (but not only) in a partial synchronous model with Byzantine processes. We conclude the paper in Section 11.

2 Model

We use a slightly extended version of the round-based model of [4]. In this model, we reason about faults only as *transmission faults*, without looking for a "culprit" for the fault [4]. Therefore there are no "faulty" processes and no state corruption in our model, but messages can be arbitrarily corrupted (or lost) before reception. Nevertheless, as we explain in Section 10, the model can be used to reason about classical Byzantine faults.

Computations in this model are structured in rounds, which are communication-closed layers in the sense that any message sent in a round can be received only in that round. As messages can be lost, this does not imply that the systems is synchronous.An algorithm $\mathcal{A}$ is specified by sending function S_p^r and transition function T_p^r for each round r and process p. We now give a formal definition of the round-based model considered, and introduce the notions of (i) the *heard-of set* $HO(p,r)$, which captures synchrony and benign faults, (ii) the *safe heard-of set* $SHO(p,r)$, which handles corruptions, i.e., captures communication safety properties, and (iii) *consistency* $CONS(r)$, which is true in round r, if all processes receive the same set of messages at round r.

2.1 Heard-Of Sets and Consistent Rounds

Let Π be a finite non-empty set of cardinality n, and let $\mathcal{M}$ be a set of messages (optionally including a *null* placeholder indicating the empty message). To each p in Π, we associate a *process*, which consists of the following components: A set of states denoted by $states_p$, a subset $init_p$ of initial states, and for each positive integer r called *round number*, a message-sending function S_p^r mapping $states_p$ to a unique message from $\mathcal{M}$, and a state-transition function T_p^r mapping $states_p$ and partial vectors (indexed by Π) of elements of $\mathcal{M}$ to $states_p$. The collection of processes is called an *algorithm on* Π. In each round r, a process p:

1. applies S_p^r to the current state and sends the message returned to each process,[4]

[4] W.l.o.g., the same message is sent to all. Because of transmission faults, this does not prevent two processes p and q from receiving different messages from some process s.

2. determines the partial vector $\boldsymbol{\mu}_p^r$, formed by the messages that p receives at round r, and
3. applies T_p^r to its current state and $\boldsymbol{\mu}_p^r$.

The partial vector $\boldsymbol{\mu}_p^r$ is called the *reception vector of p at round r*.

Computation evolves in an infinite sequence of rounds. For each process p and each round r, we introduce two subsets of Π. The first subset is the *heard-of* set, denoted $HO(p,r)$, which is the support of $\boldsymbol{\mu}_p^r$, i.e.,

$$HO(p,r) = \left\{q \in \Pi \ : \ \boldsymbol{\mu}_p^r[q] \text{ is defined}\right\}.$$

A process q is in the set $HO(p,r)$ if p receives a message from process p in round r. Note that the message received may be corrupted. The second subset is the *safe heard-of* set, denoted $SHO(p,r)$, and defined by

$$SHO(p,r) = \left\{q \in \Pi \ : \ \boldsymbol{\mu}_p^r[q] = S_q^r(s_q)\right\},$$

where s_q is q's state at the beginning of round r. A process q is in the set $SHO(p,r)$ if the message received by p is not corrupted. In addition, for each round r, we define the *consistency* flag, denoted $CONS(r)$, which is true if all processes receive the same set of messages in round r, i.e.,

$$CONS(r) = (\forall p,q \in \Pi^2 : \ \boldsymbol{\mu}_p^r = \boldsymbol{\mu}_q^r).$$

From the sets $HO(p,r)$ and $SHO(p,r)$, we form the *altered heard-of set* denoted $AHO(p,r)$ as follows:

$$AHO(p,r) = HO(p,r) \setminus SHO(p,r).$$

For any round r, and for any set of rounds Φ, we further define the *safe kernel* of r resp. Φ:

$$SK(r) = \bigcap_{p\in\Pi} SHO(p,r) \qquad SK(\Phi) = \bigcap_{r\in\Phi} SK(r)$$

The safe kernel consists of all processes whose messages were received correctly by all processes. We use also $SK = SK(\mathbb{N})$. Similarly, the *altered span* (of round r) denotes the set of processes from which at least one process received a corrupted message (at round r):

$$AS(r) = \bigcup_{p\in\Pi} AHO(p,r) \qquad AS = \bigcup_{r>0} AS(r)$$

We also extend the notion of $CONS$ in a natural way to a set Φ of rounds, i.e., $CONS(\Phi) = \bigwedge_{r\in\Phi} CONS(r)$.

2.2 HO Machines

A *heard-of machine* for a set of processes Π is a pair $(\mathcal{A},\mathcal{P})$, where $\mathcal{A}$ is an algorithm on Π, and $\mathcal{P}$ is a *communication predicate*, i.e., a predicate over the collection

$$((HO(p,r), SHO(p,r))_{p\in\Pi}\, , CONS(r))_{r>0}$$

A *run* of an HO machine M is entirely determined by the initial configuration (i.e., the collection of process initial states), and the collection of the reception vectors $\left(\boldsymbol{\mu}_p^r\right)_{p\in\Pi,\ r>0}$.

2.3 Simulation of communication predicates

In the paper we will need to simulate[5] communication predicates $\mathcal{P}'$ using some HO machine $M = (\mathcal{A},\mathcal{P})$. Intuitively, in such a simulation, several rounds of M will be used to simulate one round in which predicate $\mathcal{P}'$ holds. If the run of M consists of k rounds, then algorithm $\mathcal{A}$ is a k round simulation of $\mathcal{P}'$ from $\mathcal{P}$.

Formally, let k be any positive integer, and let $\mathcal{A}$ be an algorithm that maintains a variable $m_p \in \mathcal{M}$ and $Msg_p \in \mathcal{M}^n$ at every process p. We call *macro-round* ρ the sequence of the k consecutive round $k(\rho-1)+1,\ldots,k\rho$. The variable m_p is an input variable that can be set externally in every macro-round.[6] The value of m_p at the beginning of macro-round ρ is denoted $m_p^{(\rho)}$, and the value of Msg_p at the end of macro-round ρ is denoted $Msg_p^{(\rho)}$. For the macro-round ρ, we define in analogy to the definitions of Section 2.1:

$$HO(p,\rho) = \left\{q \in \Pi \ : \ Msg_p^{(\rho)}[q] \text{ is defined}\right\}$$
$$SHO(p,\rho) = \left\{q \in \Pi \ : \ Msg_p^{(\rho)}[q] = m_q^{(\rho)}\right\}$$
$$CONS(\rho) = (\forall p,q \in \Pi^2 : \ Msg_p^{(\rho)} = Msg_q^{(\rho)})$$

We say that the HO machine $M = (\mathcal{A},\mathcal{P})$ simulates the communication predicate $\mathcal{P}'$ in k rounds if for any run of M, the collection $(HO(p,\rho), SHO(p,\rho))_{p\in\Pi}$, $CONS(\rho))_{\rho>0}$ satisfies predicate $\mathcal{P}'$.

Given a simulation $\mathcal{A}$ of $\mathcal{P}'$ from $\mathcal{P}$, any problem that can be solved with $\mathcal{P}'$ by algorithm $\mathcal{A}'$ can be

[5] The notion of a simulation differs from the notion of a *translation* of the HO model for benign faults. A translation establishes a relation purely based on connectivity, while with value faults, also some computation is involved. Because of this, we decided thus to use the term simulation instead.

[6] The sending function in a simulation algorithm is thus a function that maps $states_p$ and the input from $\mathcal{M}$ to a unique message from $\mathcal{M}$; while the state-transition function T_p^r is a function that maps $states_p$, the input from $\mathcal{M}$, and a partial vector (indexed by Π) of elements of $\mathcal{M}$ to $states_p$.

solved with $\mathcal{P}$ instead by simply simulating rounds of the algorithm $\mathcal{A}'$ using algorithm $\mathcal{A}$. In such a composed algorithm, the input variable $m_p^{(\rho)}$ of algorithm $\mathcal{A}$ is set at each macro-round ρ to the value returned by the sending function of $\mathcal{A}'$, and the transition function of $\mathcal{A}'$ is applied to the output $Msg_p^{(\rho)}$ of algorithm $\mathcal{A}$.

3 Consensus

Let V be (non-empty) totally ordered set. In the *consensus* problem every process p has an initial value $init_p \in V$ and decides irrevocably on a decision value, fulfilling:

Integrity: If all processes have the same initial value this is the only possible decision value.
Agreement: No two processes may decide differently.
Termination: All processes eventually decide.

Since, contrary to classical approaches, there is no deviation according to T_p^r, and thus we do not have the notion of a faulty process, the upper specification makes no exemption: all processes must decide the initial value in the Integrity clause, and all processes must make a decision by the Termination clause.

Formally, an HO machine $(\mathcal{A}, \mathcal{P})$ solves consensus, if any run for which $\mathcal{P}$ holds, satisfies Integrity, Agreement, and Termination. To make this definition non-trivial, we assume that the set of *HO* and *SHO* collections for which $\mathcal{P}$ holds is non-empty.

4 Communication predicates

In this section we introduce the communication predicates that will be used in the paper. As already mentioned, we reason about faults only as transmission faults. This allows us to deal with both permanent and transient faults, but also with static and dynamic faults.

4.1 Predicates that capture static and dynamic value faults

A *dynamic* fault is a fault that can affect any link in the system — as opposed to *static* faults that affect the links of at most f out of n processes per run [4]. We start with static faults:

$$\mathcal{P}_{stat}^f \ :: \ |AS| \leq f \tag{1}$$

with $f \in N$ and $N = \{0, \dots, n\}$. $\mathcal{P}_{stat}$ is the name of the predicate, and f is a free parameter. $\mathcal{P}_{stat}^f$ is a safety predicate that models static faults, where corrupted messages are received only from a set of f processes. In Section 10 we will argue that such an assumption corresponds to a system with at most f Byzantine processes.

For our algorithms we will also consider the weaker safety predicate $\mathcal{P}_{dyn}^f$ ($\forall f \in N$, $\mathcal{P}_{stat}^f$ implies $\mathcal{P}_{dyn}^f$) that restricts the number of corrupted messages only per round and per process:

$$\mathcal{P}_{dyn}^f \ :: \ \forall r > 0, \forall p \in \Pi : |AHO(p,r)| \leq f$$

with $f \in N$ and $0 \leq f \leq n$. Predicate $\mathcal{P}_{dyn}^f$ potentially allows corrupted messages on all links in a run, it therefore models dynamic value faults.

4.2 Predicates that restrict asynchrony of communication and dynamism of faults

Predicates $\mathcal{P}_{stat}$ and $\mathcal{P}_{dyn}$ only restrict the number of value faults; however, it does not tell us anything about liveness of communication. From [13] we know that we cannot solve consensus in an asynchronous system if all messages sent by one process may be lost. On the other hand, Santoro and Widmayer [26] showed that consensus is impossible to solve in a synchronous system if, at each time unit, there is one process whose messages may be lost. Therefore, in order to solve consensus we need to restrict asynchrony of communication and dynamism of faults.

A synchronous system could be modeled as follows:

$$\mathcal{P}_{SK}^f :: |SK| \geq n - f$$

$\mathcal{P}_{SK}^f$ requires that there is a set of processes (safe kernel) of size $n-f$ whose messages are correctly received in every round. From

$$\forall f \in N, \ \mathcal{P}_{SK}^f \Rightarrow \mathcal{P}_{stat}^f$$

it follows that $\mathcal{P}_{SK}^f$ implies static faults only. However, we want to study consensus with dynamic faults. We consider therefore the following predicate:

$$\mathcal{P}_{\diamond SK}^{f,k} :: \forall r > 0 \ \exists r_o > r, \ \Phi = \{r_0, \dots, r_0 + k - 1\} : \\ |SK(\Phi)| \geq n - f$$

with $f \in N$ and $k > 0$. This predicate (repeatedly) requires a safe kernel of size $n-f$ only eventually and only for k rounds. It also restrict the dynamism of value faults during these k round; i.e., corrupted messages can only be received from at most f processes.

In the paper we will consider $\mathcal{P}_{\diamond SK}$ always in conjunction, either with $\mathcal{P}_{stat}$ or $\mathcal{P}_{dyn}$. When we assume $\mathcal{P}_{\diamond SK}$ with $\mathcal{P}_{stat}$, i.e., $\mathcal{P}^{f,k}_{\diamond SK} \wedge \mathcal{P}^{f}_{stat}$, transmission value faults are static (benign transmission faults are not restricted, so they can be dynamic). On the other hand, when we assume $\mathcal{P}_{\diamond SK}$ with $\mathcal{P}_{dyn}$, i.e., $\mathcal{P}^{f,k}_{\diamond SK} \wedge \mathcal{P}^{\alpha}_{dyn}$ with $f \leq \alpha$, transmission value faults are no more static: $\mathcal{P}^{f,k}_{\diamond SK}$ alone does not imply $\mathcal{P}^{f'}_{stat}$ for any $f' < n$.

The implementation of the predicate $\mathcal{P}_{\diamond SK}$ in a partially synchronous system (in conjunction, either with $\mathcal{P}_{stat}$ or $\mathcal{P}_{dyn}$) is not discussed in this paper. The reader is referred to [12,6].

4.3 Permanent versus Transient Faults

Both predicates, $\mathcal{P}_{stat} \wedge \mathcal{P}_{\diamond SK}$ and $\mathcal{P}_{dyn} \wedge \mathcal{P}_{\diamond SK}$ allow *permanent faults*. Consider for example a run and a process p, where every process receives a corrupted message from p in every round:

$$\forall q \in \Pi, r > 0 : \; p \notin SHO(q, r)$$

and all other messages are received correctly. Such a run is included in the set of runs given by $\mathcal{P}_{stat} \wedge \mathcal{P}_{\diamond SK}$ and $\mathcal{P}_{dyn} \wedge \mathcal{P}_{\diamond SK}$, and thus the algorithms given later in the paper can solve consensus in such a run. More precisely, $\mathcal{P}^{f}_{stat} \wedge \mathcal{P}^{f}_{\diamond SK}$ and $\mathcal{P}^{f}_{dyn} \wedge \mathcal{P}^{f}_{\diamond SK}$ permits the existence of up to f such processes. As pointed out in Section 10, this allows our algorithms to solve consensus also, e.g., in classical models with Byzantine faults, an addresses the question raised in the introduction. Indeed, this contrasts with [4], where, although also $\mathcal{P}_{dyn}$ is considered (named $\mathcal{P}_{\alpha}$ there), eventually there has to be a round, where a sufficiently large subset of processes do not receive any corrupted messages. There, (most) faults have to be *transient*.

5 Eventual Consistency

In this section we introduce the notion of *eventual consistency* that turns out to be a fundamental building block for solving consensus under transmission value faults. Eventual consistency abstracts the major complexity present when solving consensus under the weak communication predicates presented above. Therefore eventual consistency allows us to express consensus algorithms in a very concise and elegant way.

Informally speaking, eventual consistency combines the requirement of a consistent round ($CONS(r)$ in our model) with some requirements on liveness and safety of communication. It can be seen as an *eventual* version of *interactive consistency* [22]. In a component fault model, an algorithm that solves interactive consistency allows correct processes to agree on a vector, where at least $n - f$ entries correspond to the initial values of the corresponding correct processes (f is the maximum number of faulty processes).

Interactive consistency, when seen as a communication primitive, can be captured by the following predicate:

$$\mathcal{P}^{f}_{IC} :: \; |SK| \geq n - f \wedge \forall r > 0 : CONS(r)$$

When we express the result of [22] in our model, their algorithm allows a $f + 1$ round simulation of $\mathcal{P}^{f}_{IC}$ from $\mathcal{P}^{f}_{SK}$ if $n > 3f$. Note that $\forall f \in N$, $\mathcal{P}^{f}_{IC} \Rightarrow \mathcal{P}^{f}_{stat}$.

Instead of $\mathcal{P}_{IC}$, we introduce a weaker predicate. We call the predicate *eventual consistency* and define it as follows:

$$\mathcal{P}^{f}_{\diamond cons} :: \; \forall r > 0 \; \exists r_o > r : \; |SK(r_0)| \geq n - f \wedge CONS(r_0)$$

This predicate requires that there is always eventually a consistent round with a safe kernel of size $n - f$. In contrast to $\mathcal{P}^{f}_{SK}$ and $\mathcal{P}_{IC}$, this predicate requires these safe kernels only eventually and then only for a single round. Also faults are no more static: $\mathcal{P}^{f}_{\diamond cons}$ alone does not imply $\mathcal{P}^{f'}_{stat}$ for any $f' < n$. Note that $\mathcal{P}^{f}_{\diamond cons}$ is a stronger predicate than $\mathcal{P}^{f,1}_{\diamond SK}$: although both predicates require a safe kernel of size $n - f$ and both restrict the dynamism of value faults for a single round, $\mathcal{P}^{f}_{\diamond cons}$ in addition requires that consistency holds during this round, i.e., for any two processes p and q we have $\boldsymbol{\mu}_p = \boldsymbol{\mu}_q$.

However, $\mathcal{P}_{\diamond cons}$ can be simulated from $\mathcal{P}_{\diamond SK}$. In the next section, we give two such simulations, and then establish the link to solving consensus.

6 Simulating eventual consistency $\mathcal{P}_{\diamond cons}$ from eventually safe kernels $\mathcal{P}_{\diamond SK}$

In this section we give two simulations of $\mathcal{P}_{\diamond cons}$ from $\mathcal{P}_{\diamond SK}$, one in the presence of only static value faults ($\mathcal{P}_{stat}$), and the other in the presence of dynamic (and static) value faults ($\mathcal{P}_{dyn}$). As we show, the first simulation requires a smaller number of processes in order to tolerate a given number of transmission value faults. Then we introduce a *generic predicate* $\mathcal{P}_{\diamond cons \oplus SK}$ that can be simulated from $\mathcal{P}_{\diamond SK}$. The predicate $\mathcal{P}_{\diamond cons \oplus SK}$, in conjunction with $\mathcal{P}_{dyn}$ or $\mathcal{P}_{stat}$, is later used in Section 7 to solve consensus.

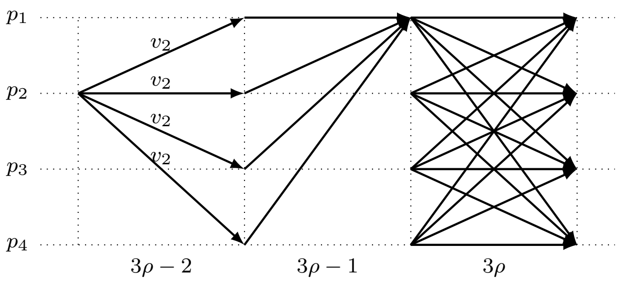


Fig. 1 Algorithm 1 from the point of view of v_2 sent by p_2 (p_1 is the coordinator, $n = 4$, $f = 1$).

6.1 Simulation in the presence of only static value faults

Algorithm 1 is a 3-round simulation of $\mathcal{P}^f_{\diamond cons} \wedge \mathcal{P}^f_{stat}$ from $\mathcal{P}^{3(f+1)}_{\diamond SK} \wedge \mathcal{P}^f_{stat}$ inspired by [10]. It ensures consistency during a sequence of rounds where the size of the kernel is at least $n - f$ (the corrupted messages can be received only from at most f processes). Moreover, it preserves $\mathcal{P}^f_{stat}$, i.e., if $\mathcal{P}^f_{stat}$ holds for basic rounds, then $\mathcal{P}^f_{stat}$ holds also for the macro-rounds obtained by the 3-round simulation using Algorithm 1. It requires $n > 3f$. As already mentioned in Section 2.3, a simulation is an algorithm that maintains at each process p two variables: an input variable m_p that is set at the beginning of every macro-round ρ (line 7), and an output variable Msg_p whose value is considered at the end of every macro-round ρ (lines 24 and 26). The special value $\perp$ represents the case when a (reception) vector does not contain a message from the respective process.

Algorithm 1 is a coordinator-based algorithm, where the coordinator is chosen using a rotating coordinator strategy: the coordinator of macro-round ρ is process $\rho \bmod n + 1$; in Algorithm 1 the variable *coord* refers to this process. We describe Algorithm 1 from the point of view of the message v_2 that is sent by process p_2 using Figure 1. Assume that process p_1 is the coordinator. In round $3\rho - 2$, process p_2 sends the message v_2 to all. In rounds $3\rho - 1$ and 3ρ of Algorithm 1, the processes send messages that contain a vector of those messages received in round $3\rho - 2$. In this description we focus only on those elements of the vectors that are related to message v_2 that is sent by process p_2 in macro-round ρ. In round $3\rho - 1$, all processes send the value received from p_2 to all. [7] The coordinator then compares the value received from p_2, say v_2, in round $3\rho - 2$ with the value indirectly received from the other processes.

Algorithm 1 Simulation of $\mathcal{P}^f_{\diamond cons} \wedge \mathcal{P}^f_{stat}$ from $\mathcal{P}^{f,3(f+1)}_{\diamond SK} \wedge \mathcal{P}^f_{stat}$

```
1:  Initialization:
2:    $Msg_p \leftarrow (\perp, \ldots, \perp)$        /* $Msg_p$ is the output variable */
3:                            /* $\perp$ represents the absence of message */
4:    $coord_p = \rho \bmod n + 1$

5:  Round $r = 3\rho - 2$:
6:    $S^r_p$:
7:      send $m_p$ to all
8:    $T^r_p$:
9:      $received_p \leftarrow \mu^r_p$

10: Round $r = 3\rho - 1$:
11:   $S^r_p$:
12:     send $received_p$ to all
13:   $T^r_p$:
14:     if $p = coord$ then
15:       for all $q \in \Pi$ do
16:         if $|\{q' \in \Pi : \mu^r_p[q'][q] = received_p[q]\}| < 2f + 1$
            then
17:           $received_p[q] \leftarrow \perp$

18: Round $r = 3\rho$:
19:   $S^r_p$:
20:     send $\langle received_p \rangle$ to all
21:   $T^r_p$:
22:     for all $q \in \Pi$ do
23:       if $(\mu^r_p[coord_p][q] \neq \perp)$ $\wedge$
          $|\{i \in \Pi : \mu^r_p[i][q] = \mu^r_p[coord_p][q]\}| \geq f + 1$ then
24:         $Msg_p[q] \leftarrow \mu^r_p[coord_p][q]$
25:       else
26:         $Msg_p[q] \leftarrow \perp$
```

If at least $2f + 1$ values v_2 have been received by the coordinator p_1, then p_1 keeps v_2 as the message received from p_2. Otherwise p_1 sets the message received from p_2 to $\perp$ (line 17). This guarantees that if p_1 keeps v_2, then at least $f + 1$ processes have received v_2 from p_2 in round $3\rho - 2$. Finally, in round 3ρ every process sends the value received from p_2 in round $3\rho - 2$ to all. The final value adopted as message received from p_2 at the end of round 3ρ (and therefore at the end of macro-round ρ) is computed as follows at each process p_i. Let val_i be the value received from coordinator p_1 in round 3ρ. If $val_i = \perp$ then p_i receives $\perp$ from p_2. Process p_i receives $\perp$ from p_2 in another case: if p_i did not receive $f + 1$ values equal to val_i in round 3ρ. Otherwise, at least $f + 1$ values received by p_i in round 3ρ are equal to val_i, and p_i adopts val_i as message received from p_2 in macro-round ρ.

Algorithm 1 relies on a coordinator for ensuring $\mathcal{P}^f_{\diamond cons}$: all processes assign to Msg_p the value received from the coordinator in round 3ρ (see line 24). This is achieved during a macro-round in which the size of the safe kernel is at least $n - f$, with the coordinator in the safe kernel. The rotating coordinator strategy ensures the existence of such a macro-round. Consider Figure 2 that illustrates the mechanism for ensuring consistency

[7] At line 16, the reception vector μ^r_p is a vector of vectors: $\mu^r_p[q']$ is the vector p has received from q', and $\mu^r_p[q'][q]$ is element q of this vector.

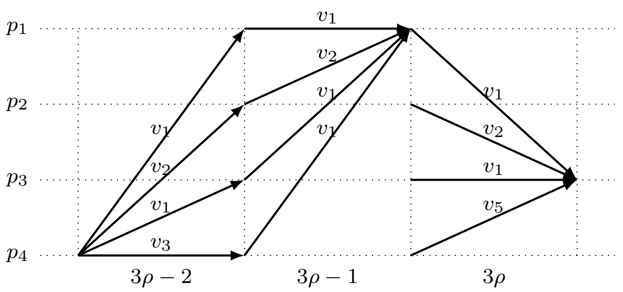


Fig. 2 How Algorithm 1 ensures $\mathcal{P}^{f}_{\diamond cons}$: point of view of message sent by p_4 and received by p_3. Process p_1 is the coordinator, $n = 4$, $f = 1$, only messages received from p_4 can be corrupted.

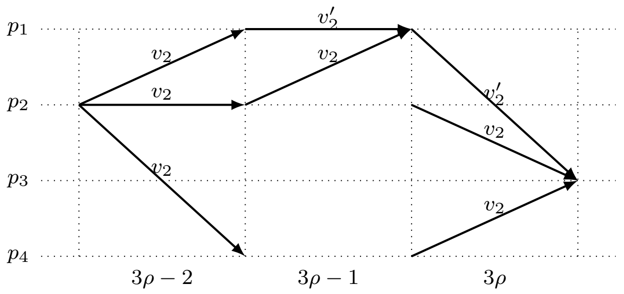


Fig. 3 How Algorithm 1 preserves $\mathcal{P}^{f}_{stat}$: point of view of v_2 sent by p_2 and received by p_3. Process p_1 is the coordinator, $n = 4$, $f = 1$, only messages received from p_1 can be corrupted. Absence of arrows represents message loss. Process p_3 can only receive v_2 or $\perp$ from p_2 (here $\perp$).

from the point of view of the message sent by process p_4 and received by process p_3. The coordinator adopts value v_1 as the message sent by process p_4 in round $3\rho-1$ (line 16) since it is forwarded by $2f+1$ processes. This ensures that the value v_1 sent by the coordinator in round 3ρ is also sent by at least f more processes from the safe kernel in round 3ρ. Therefore, the value sent by the coordinator satisfies the condition of line 23 at all processes and is therefore assigned to $Msg_p[p_4]$ by all processes at line 24.

Using Figure 3, we now explain how Algorithm 1 preserves $\mathcal{P}^{f}_{stat}$. Figure 3 considers message v_2 sent by p_2 and received by p_3; again, process p_1 is the coordinator. Messages received from p_2 in round $3\rho-2$ are not corrupted, and we show that the message received by p_3 from p_2 in macro-round ρ can only be v_2 or $\perp$. In round 3ρ, process p_3 does not "blindly" adopt the value received from the coordinator (the message received can be corrupted). The value received in round 3ρ from the coordinator is adopted by p_3 only if the same value is received from at least f additional processes (line 23). This ensures that at least one such message is not corrupted. In Figure 3, process p_3 adopts $\perp$ as message received from p_2 in macro-round ρ, since it did not receive $f+1$ messages equal to value v_2' received from the coordinator.

Lemma 1 *If $n > 3f$ then Algorithm 1 preserves $\mathcal{P}^{f}_{stat}$.*

Proof To avoid ambiguities, let in this proof $AS_\rho = \bigcup_{\rho>0} AS(\rho)$ denote the altered span with respect to macro-rounds implemented by Algorithm 1, while $AS = \bigcup_{r>0} AS(r)$ denotes the altered span with respect to the rounds of Algorithm 1.

We need to show that $|AS_\rho| \le f$ given that $|AS| \le f$. It is thus sufficient to show $AS_\rho \subseteq AS$. Assume by contradiction that there is a process $p \in \Pi$, a process $s \notin AS$, and a macro-round ρ so that $s \in AHO(p,\rho)$, i.e., s sends message m in macro-round ρ and p receives $m' \neq m$.

Then, because of line 23, for $Q = \{q : \boldsymbol{\mu}_p^{3\rho}[q][s] = m'\}$ we have $|Q| \ge f+1$. Because of $|AS| \le f$, there is a $i \in Q$ that has $received_i[s] = m'$. Moreover, this implies that $\boldsymbol{\mu}_i^{3\rho-2}[s] = m'$. Since s sent m, this is a contradiction to $s \notin AS$. □

Lemma 2 *If $n > 3f$ then Algorithm 1 simulates $\mathcal{P}^{f}_{\diamond cons}$ from $\mathcal{P}^{f,3(f+1)}_{\diamond SK}$.*

Proof Let ρ denote a macro-round, let $\Phi = \{3\rho-2, 3\rho-1, 3\rho\}$ be the set of rounds of ρ, and let $c_0 = \rho \bmod n+1$ be the coordinator of ρ such that

$$c_0 \in SK(\Phi) \ \wedge |SK(\Phi)| \ge n-f.$$

Such a macro-round exists, because *(i)* $\mathcal{P}^{f,3(f+1)}_{\diamond SK}$ holds and *(ii)* the coordinator is chosen using a rotating coordinator scheme (the coordinator of macro-round ρ is process ρ mod $n+1$). We show that with Algorithm 1 (i) $CONS(\rho)$ and (ii) $|SK(\rho)| \ge n-f$.

(i) Assume by contradiction that for two processes p and q, $Msg_p^{(\rho)}$ and $Msg_q^{(\rho)}$ differ by the message of process $s \in \Pi$, that is $Msg_p^{(\rho)}[s] \neq Msg_q^{(\rho)}[s]$. By round 3ρ, every process adopts the value of c_0 or sets $Msg^{(\rho)}[s]$ to $\perp$; when $c_0 \in SK(\Phi)$ it follows that $Msg_p^{(\rho)}[s]$ or $Msg_q^{(\rho)}[s]$ is $\perp$. W.l.o.g. assume that $Msg_p^{(\rho)}[s] = v$ and $Msg_q^{(\rho)}[s] = \perp$. For rounds $r \in [3\rho-1, 3\rho]$, let

$$R_p^r(v,s) := \left\{i \in \Pi \ : \ \boldsymbol{\mu}_p^r[i][s] = v\right\}$$

represent the set of processes from which p receives v at position s in round r. Similarly, for rounds $r \in [3\rho-1, 3\rho]$, let

$$Q^r(v,s) := \{i \in \Pi \ : \ S_i^r(s_i^r)[s] = v\}$$

represent the set of processes that sent v at position s in round r.

By line 23, if $Msg_p^{(\rho)}[s] = v$, then $|R_p^{3\rho}(v,s)| \ge f+1$, and $c_0 \in R_p^{3\rho}(v,s)$. Since $c_0 \in SK(\Phi)$, we have $c_0 \in$

$Q^{3\rho}(v,s)$ and thus, by line 16, $|R^{3\rho-1}_{c_0}(v,s)| \geq 2f+1$. From this and $|SK(\Phi)| \geq n-f$, we have $|R^{3\rho-1}_{c_0}(v,s) \cap SK(\Phi)| \geq f+1$. Therefore, at least $f+1$ processes p' in $SK(\Phi)$, including c_0, have $received_{p'}[s] = v$. It follows that $|R^{3\rho}_q(v,s)| \geq f+1$, and $c_0 \in R^{3\rho}_q(v,s)$. This contradicts the assumption that the condition in line 23 is false for process q.

(ii) For every process $p \in \Pi$ and $q \in SK(\Phi)$, by definition we have $received_p[q] = m_q$ at the end of round $3\rho-2$. In round $3\rho-1$, c_0 receives $received_{q'}[q] = m_q$ from every process $q' \in SK(\Phi)$, and thus there is no $q \in SK(\Phi)$ s.t. c_0 sets $received_{c_0}[q]$ to $\perp$ (*). In round 3ρ, since $c_0 \in SK(\Phi)$, every process p receives the message from c_0. In addition, since $n > 3f$ and $|SK(\Phi)| \geq n-f$, every process receives the message from $n-f > f+1$ processes in $SK(\Phi)$. By (*) and line 23, for all processes p and all $q \in SK(\Phi)$, we have $Msg_p[q] = m_q$. Thus $SK(\Phi) \subseteq SK(\rho)$, which shows that $|SK(\rho)| \geq n-f$. □

Corollary 1 *If $n > 3f$, Algorithm 1 is a simulation of $\mathcal{P}^f_{\diamond cons} \wedge \mathcal{P}^f_{stat}$ from $\mathcal{P}^{f,3(f+1)}_{\diamond SK} \wedge \mathcal{P}^f_{stat}$.*

Remark 1. Algorithm 1 can easily be extended to preserve also the following predicate:

$$|HO(p,r)| \geq n-f$$

Intuitively, such an assumption is typical for algorithms that are designed to work in a system with reliable channels. The modified simulation algorithm then uses the reception vector of the first round as *Msg* in case there would be less than f elements in *Msg*. It is easy to show that this does not affect Corollary 1, while preserving the above predicate. Since our algorithms do not need this assumption, we do not detail this extension further.

Remark 2. Interestingly there is also decentralized (i.e., coordinator-free) solution to this simulation. The algorithm is presented in [8] in terms of Byzantine faults but can be easily adapted to our framework. Such a simulation requires $f+1$ rounds. In some cases this approach can be beneficial [7].

6.2 Simulation in the presence of dynamic value faults

In this section we show a simulation of $\mathcal{P}_{\diamond cons}$ from $\mathcal{P}_{\diamond SK}$ and the weaker predicate $\mathcal{P}_{dyn}$ that (partially) preserves $\mathcal{P}_{dyn}$. More precisely, we show a simulation from $\mathcal{P}^f_{\diamond SK} \wedge \mathcal{P}^{\alpha}_{dyn}$ into $\mathcal{P}^f_{\diamond cons} \wedge \mathcal{P}^{\beta}_{dyn}$ with $\beta \geq \alpha$: the simulation may only partially preserve $\mathcal{P}^{\alpha}_{dyn}$ in the sense that the number of corruptions in the simulated rounds may increase from α to $\beta \geq \alpha$, depending on n.

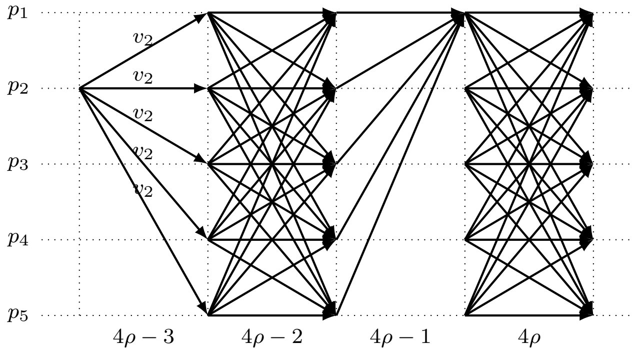


Fig. 4 Algorithm 2 from the point of view of v_2 sent by p_2; p_1 is the coordinator, $n=4$, $f=1$.

The simulation requires four rounds, as shown by Algorithm 2. As we can see, β is not a parameter of the algorithm. Fixing β leads to some requirement on n. More precisely, given f, $\alpha \geq f$, $\beta \geq \alpha$, Algorithm 2 requires $n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1}$. Similarly to Algorithm 1, it is coordinator-based.

The communication pattern of Algorithm 2 is very similar to Algorithm 1 with the addition of one "all-to-all" round (see Figure 4, to be compared with Figure 1). We explain Algorithm 2 from the point of view of the message sent by process p_2. In round $4\rho-3$, process p_2 sends message v_2 to all.[8] In round $4\rho-2$, all processes send to all the value received from p_2, and then compare the value v_2 received from p_2 in round $4\rho-3$ with the value indirectly received from the other processes in round $4\rho-2$. If at least $n-f$ values v_2 have been received by process p, then p keeps v_2 as the message received from p_2. Otherwise, the message received from p_2 is $\perp$ (line 9). As explained later, rounds $4\rho-3$ and $4\rho-2$ filter the values for rounds $4\rho-1$ and 4ρ in order to ensure $\mathcal{P}^{\beta}_{dyn}$ from $\mathcal{P}^{\alpha}_{dyn}$. Rounds $4\rho-1$ and 4ρ are very similar to rounds $3\rho-1$ and 3ρ in Algorithm 1.

Algorithm 2 relies on a coordinator for ensuring $\mathcal{P}^f_{\diamond cons}$: all processes assign to Msg_p the value received from the coordinator in round 4ρ (see line 31). This is achieved during a macro-round in which the size of the safe kernel is at least $n-f$, with the coordinator in the safe kernel. Since consistency is ensured under the same conditions as with Algorithm 1, we use exactly the same mechanism in Algorithm 2.

The additional complexity of Algorithm 2 comes from the part responsible for ensuring $\mathcal{P}^{\beta}_{dyn}$. We start by explaining on Figure 5 why Algorithm 1 does not preserve $\mathcal{P}^{\alpha}_{dyn}$ for the simplest case $f = \alpha = 1$, $n = 4$.

[8] Similar as in the description of Algorithm 1, in case of messages that contain a vector of messages, we focus only on those elements of the vectors that are related to the message sent by process p_2.

Algorithm 2 Simulation of $\mathcal{P}^{f}_{\diamond cons} \wedge \mathcal{P}^{\beta}_{dyn}$ from $\mathcal{P}^{f,4(f+1)}_{\diamond SK} \wedge \mathcal{P}^{\alpha}_{dyn}$

```
1:  Initialization:
2:    $Msg_p \leftarrow (\bot, \ldots, \bot)$ /* $Msg_p$ is the output variable */
3:    $coord_p = \rho \bmod n + 1$

4:  Round $r = 4\rho - 3$:
5:    $S^r_p$:
6:      send $m_p$ to all          /* $m_p$ is the input variable */
7:    $T^r_p$:
8:      $first_p \leftarrow \boldsymbol{\mu}^r_p$
9:      $conf_p \leftarrow (\bot, \ldots, \bot)$

10: Round $r = 4\rho - 2$:
11:   $S^r_p$:
12:     send $first_p$ to all
13:   $T^r_p$:
14:     for all $q \in \Pi$ do
15:       if $\left|\{i \in \Pi : \boldsymbol{\mu}^r_p[i][q] = first_p[q]\}\right| \geq n - f$ then
16:         $conf_p[q] \leftarrow first_p[q]$

17: Round $r = 4\rho - 1$:
18:   $S^r_p$:
19:     send $conf_p$ to all
20:   $T^r_p$:
21:     if $p = coord_p$ then
22:       for all $q \in \Pi$ do
23:         if $\left|\{i \in \Pi : \boldsymbol{\mu}^r_p[i][q] = conf_p[q]\}\right| < \alpha + f + 1$ then
24:           $conf_p[q] \leftarrow \bot$

25: Round $r = 4\rho$:
26:   $S^r_p$:
27:     send $conf_p$ to all
28:   $T^r_p$:
29:     for all $q \in \Pi$ do
30:       if $\left|\{i \in \Pi : \boldsymbol{\mu}^r_p[i][q] = \boldsymbol{\mu}^r_p[coord_p][q]\}\right| \geq \alpha + 1$ then
31:         $Msg_p[q] \leftarrow \boldsymbol{\mu}^r_p[coord_p][q]$
32:       else
33:         $Msg_p[q] \leftarrow \bot$
```

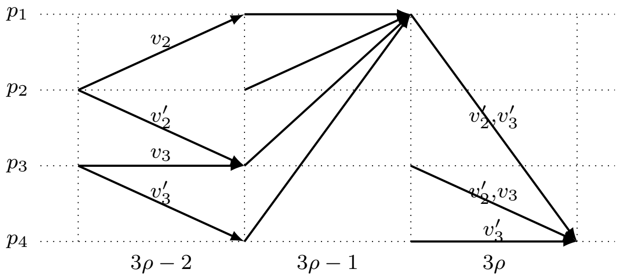


Fig. 5 Algorithm 1 does not preserves $\mathcal{P}^{\alpha}_{dyn}$; from the point of view of p_2 and p_3, that sends correspondingly v_2 and v_3 in round $3\rho - 2$, and reception of process p_4 of messages sent by p_2 and p_3. $n = 4$, $f = \alpha = \beta = 1$ and p_1 is coordinator. Message received by p_4 from coordinator in round 3ρ is corrupted; other messages are correctly received. Absence of arrows represents message loss.

$first_{p_1}$	×			
$first_{p_2}$		×		
$first_{p_3}$			×	
$first_{p_4}$				×

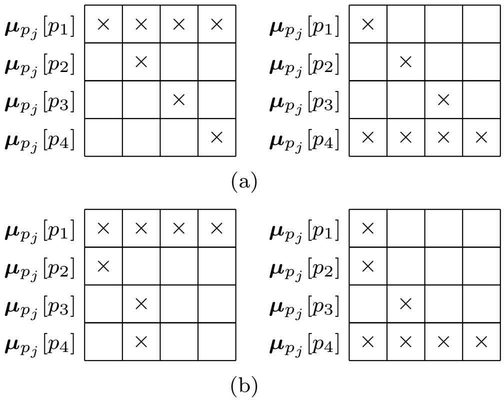
$first_{p_1}$	×			
$first_{p_2}$	×			
$first_{p_3}$		×		
$first_{p_4}$		×		

Fig. 6 After round $4\rho - 3$: Two examples (left and right) of corrupted values (represented by X).

According to $\mathcal{P}^{1}_{dyn}$, every process can receive at most one corrupted message per round. In round $3\rho - 2$, process p_3 receives the corrupted message v_2' from p_2, and p_4 receives the corrupted value v_3' from p_3. These values are sent to the coordinator p_1 in round $3\rho - 1$. Finally, in round 3ρ, process p_4 receives v_2', v_3' from p_1, v_2', v_3 from p_3, and v_3' from itself. Since there are $f + 1$ values equal to those sent by coordinator, p_4 considers v_2', respect. v_3', as messages received from p_2, respect. p_3, in macro-round ρ, violating $\mathcal{P}^{1}_{dyn}$. The problem comes from the fact that dynamic faults have a cumulative effect, i.e., messages that are corrupted in round $3\rho - 2$ add to corrupted messages from round 3ρ.

We now explain why the addition of round $4\rho - 2$ allows us to cope with this issue. Informally speaking, the role of round $4\rho - 2$ in Algorithm 2 is to transform dynamic faults into some maximum number of static faults, i.e., into some maximum number of faults localized at some fixed set of processes. Consider rounds $4\rho - 3$ and $4\rho - 2$, with $n = 4$, $\alpha = f = 1$. In round $4\rho - 3$, predicate $\mathcal{P}^{\alpha}_{dyn}$ ensures that, in total, at most $n \cdot \alpha = 4$ corrupted values are received. In other words, among the vectors $first_{p_1}$ to $first_{p_4}$ received (line 8), at most $n \cdot \alpha = 4$ elements can be corrupted (see Figure 6, where × represents possible corrupted values). In round $4\rho - 2$, each process p_i sends vector $first_{p_i}$ to all processes. Consider the reception of these four vectors by some process p_j. Since $\alpha = 1$, one of these vectors can be received corrupted at p_j. Figure 7 shows four examples, two starting from Figure 6 left, two starting from Figure 6 right.

To understand which value p adopts from q (lines 15 and 16) we need to look at column q in Figure 7. From line 16, p adopts a corrupted value from q only if column q contains at least $n - f = 3$ corrupted values. In the upper case, no column satisfies this condition, i.e., p adopts no corrupted value. In the lower case, columns 2 and 1 satisfy this condition, i.e., corrupted values can be adopted from p_2 or p_1. It is easy to see that in the case $n = 4$, $f = \alpha = 1$, corrupted values can be adopted from at most one process. In other words, rounds $4\rho - 3$ and $4\rho - 2$ has transformed $\alpha = 1$ dynamic fault into at most $\beta = 1$ static faults. However, in the case $n = 5$, $f = \alpha = 2$, rounds $4\rho - 3$ and $4\rho - 2$ transform $\alpha = 2$ dynamic fault into at most $\beta = 3$ static fault.



$\boldsymbol{\mu}_{p_j}[p_1]$
$\boldsymbol{\mu}_{p_j}[p_2]$
$\boldsymbol{\mu}_{p_j}[p_3]$
$\boldsymbol{\mu}_{p_j}[p_4]$
(a)
(b)

Fig. 7 After round $4\rho - 2$: (a) Two examples of vectors received by some p_j starting from Fig. 6 left; (b) Two examples of vectors received by some p_j starting from Fig. 6 right (corrupted values are represented by ×).

Transforming α dynamic faults into $\beta \geq \alpha$ static faults allows us to rely on the same mechanism as in Algorithm 1 for the last two rounds of the simulation. Note that in rounds $4\rho - 1$ and 4ρ of Algorithm 2 we have dynamic faults, while in rounds $3\rho - 1$ and 3ρ of Algorithm 1 faults were static. Nevertheless the same mechanisms can be used in both cases.

Theorem 1 *If $n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1}$, $n > \alpha + f$, $\alpha \geq f$, and $\beta \geq \alpha$, then Algorithm 2 simulates $\mathcal{P}^f_{\diamond cons} \wedge \mathcal{P}^\beta_{dyn}$ from $\mathcal{P}^{f,4(f+1)}_{\diamond SK} \wedge \mathcal{P}^\alpha_{dyn}$.*

The theorem follows directly from Lemmas 3 and 4: the first lemma considers $\mathcal{P}^\beta_{dyn}$ and $\mathcal{P}^\alpha_{dyn}$, the second $\mathcal{P}^f_{\diamond cons}$ and $\mathcal{P}^{f,4(f+1)}_{\diamond SK}$.

Lemma 3 *If $n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1}$ and $\beta \geq \alpha$, then Algorithm 2 simulates $\mathcal{P}^\beta_{dyn}$ from $\mathcal{P}^\alpha_{dyn}$.*

Proof We need to show that for every macro-round ρ, and every process p, we have $|AHO(p, \rho)| \leq \beta$, i.e., at most β messages are corrupted.

Assume by contradiction that there is a process p so that $|AHO(p, \rho)| > \beta$. That is, we have $|S| \geq \beta + 1$ for

$$S = \big\{s \in \Pi : Msg_p[s] \neq m_s \text{ and } Msg_p[s] \neq \bot\big\}$$

For all $s \in S$, let m'_s denote $Msg_p[s]$. The output $Msg_p[s]$ is set at line 31. Because of line 30, this implies that

$$\forall s \in S: \ \big|\big\{i \in \Pi : \ \boldsymbol{\mu}^{4\rho}_p[i][s] = m'_s\big\}\big| \geq \alpha + 1.$$

Because of $|AHO(p, 4\rho)| \leq \alpha$, at the end of round $4\rho-1$ we have

$$\forall s \in S, \exists i_s \in \Pi : conf_{i_s}[s] = m'_s.$$

Since in round $4\rho - 1$ the elements of *conf* can only be set to $\bot$, the same condition needs to holds also at the end of round $4\rho - 2$. Because of line 15, this implies

$$\forall s \in S, \exists i_s \in \Pi, \exists Q_s \subseteq \Pi, |Q_s| \geq n - f, \forall q \in Q_s :$$
$$\boldsymbol{\mu}^{4\rho-2}_{i_s}[q][s] = m'_s.$$

Because of $|AHO(p, 2)| \leq \alpha$, at the end of round $4\rho - 3$ we have

$$\forall s \in S, \exists Q'_s \subseteq \Pi, |Q'_s| \geq n - f - \alpha : \forall q \in Q'_s :$$
$$first_q[s] = m'_s.$$

Note that $first_q = \boldsymbol{\mu}^{4\rho-3}_q$. The number of tuples (q, s) such that $\boldsymbol{\mu}^{4\rho-3}_q[s] = m'_s$ is thus at least $(\beta + 1)(n - f - \alpha)$. From this it follows that there is at least one process q_0 where the number of corrupted messages in the first round is

$$\left\lceil \frac{(\beta + 1)(n - f - \alpha)}{n} \right\rceil = \alpha + \left\lceil \frac{(\beta + 1)(n - f - \alpha) - n\alpha}{n} \right\rceil > \alpha,$$

where the last inequation follows from $n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1}$ and $\beta \geq \alpha$, which ensures $(\beta + 1)(n - f - \alpha) - n\alpha > 0$. Therefore $AHO(q_0, 4\rho - 3) > \alpha$, which contradicts the assumption $AHO(q_0, 4\rho - 3) \leq \alpha$. □

Lemma 4 *If $n > \alpha + f$ and $\alpha \geq f$, then Algorithm 2 simulates $\mathcal{P}^f_{\diamond cons}$ from $\mathcal{P}^{f,4(f+1)}_{\diamond SK}$.*

Proof Let ρ denote a macro-round, let $\Phi = \{4\rho - 3, \ldots, 4\rho\}$ be the set of rounds of ρ, and let $c_0 = \rho \bmod n + 1$ be the coordinator of ρ such that

$$c_0 \in SK(\Phi) \ \wedge |SK(\Phi)| \geq n - f.$$

Such a macro-round exists, because *(i)* $\mathcal{P}^{f,4(f+1)}_{\diamond SK}$ holds and *(ii)* the coordinator is chosen using a rotating coordinator scheme (the coordinator of macro-round ρ is process $\rho \bmod n + 1$). We show that with Algorithm 2 (i) $CONS(\rho)$ and (ii) $|SK(\rho)| \geq n - f$.

(i) Assume by contradiction that for two processes p and q, $Msg^{(\rho)}_p$ and $Msg^{(\rho)}_q$ differ by the message of process $s \in \Pi$, that is $Msg^{(\rho)}_p[s] \neq Msg^{(\rho)}_q[s]$. By round 4ρ, every process adopts the value of c_0 or sets $Msg^{(\rho)}[s]$ to $\bot$; when $c_0 \in SK(\Phi)$ it follows that $Msg^{(\rho)}_p[s]$ or $Msg^{(\rho)}_q[s]$ is $\bot$. W.l.o.g. assume that $Msg^{(\rho)}_p[s] = v$ and $Msg^{(\rho)}_q[s] = \bot$. For rounds $r \in [4\rho-1, 4\rho]$, let $R^r_p(v, s) := \big\{q \in \Pi \ : \ \boldsymbol{\mu}^r_p[q][s] = v\big\}$ represent the set of processes from which p receives v at position s. Similarly, for rounds $r \in [4\rho - 1, 4\rho]$, let

$$Q^r(v, s) := \big\{q \in \Pi \ : \ S^r_q(s^r_q)[s] = v\big\}$$

represent the set of processes that sent v at position s.

By line 30, if $Msg^{(\rho)}_p[s] = v$, then $|R^{4\rho}_p(v, s)| \geq \alpha+1$, and $c_0 \in R^{4\rho}_p(v, s)$. Since $c_0 \in SK(\Phi)$, we have $c_0 \in$

$Q^{4\rho}(v,s)$ and thus, by line 23, $|R^{4\rho-1}_{c_0}(v,s)| \geq \alpha+f+1$. From this and $|SK(\Phi)| \geq n-f$, we have $|R^{4\rho-1}_{c_0}(v,s) \cap SK(\Phi)| \geq \alpha+1$. Therefore, at least $\alpha+1$ processes p' in $SK(\Phi)$, including c_0, have $conf_{p'}[s] = v$. It follows that $|R^{4\rho}_q(v,s)| \geq \alpha+1$, and $c_0 \in R^{4\rho}_q(v,s)$. This contradicts the assumption that the condition in line 30 is false for process q.

(ii) For every processes $p \in \Pi$ and $q \in SK(\Phi)$, by definition we have $first_p[q] = m_q$ at the end of round $4\rho-3$. In round $4\rho-2$, for every process $s \in SK(\Phi)$, $first_s$ is received. Therefore, by line 15 since $|SK(\Phi)| \geq n-f$, at every process $p \in \Pi$ we have $conf_p[q] = m_q$, for all $q \in SK(\Phi)$ (*). In round $4\rho-1$, c_0 receives $conf_{q'}[q] = m_q$ from every process $q' \in SK(\Phi)$, and thus there is no $q \in SK(\Phi)$ s.t. c_0 sets $conf_{c_0}[q]$ to $\perp$ (**). In round 4ρ, since $c_0 \in SK(\Phi)$, every process p receives the message from c_0. In addition, since $n \geq f+\alpha+1$ and $|SK(\Phi)| \geq n-f$, every process receives the message from $n-f \geq \alpha+1$ processes in $SK(\Phi)$. By (*), (**) and line 30, for all processes p and all $q \in SK(\Phi)$, we have $Msg_p[q] = m_q$. Thus $SK(\Phi) \subseteq SK(\rho)$, which shows that $SK(\rho) \geq n-f$. □

Corollaries 2 and 3 follow from Lemma 3.

Corollary 2 *If $n > (\alpha+1)(\alpha+f)$, then Algorithm 2 preserves $\mathcal{P}^{\alpha}_{dyn}$.*

By Corollary 2, preserving $\mathcal{P}^{\alpha}_{dyn}$ leads to a quadratic dependency between n and α. Corollary 3 shows the surprising result that, allowing more than α corruptions in the simulated round, leads instead to a linear dependency between n and α. Note that the simulation mentioned in Corollary 3 is not useful if $\left\lfloor \frac{\eta}{\eta-1}\alpha \right\rfloor \geq n$.

Corollary 3 *For any $\eta \in \mathbb{R}$, $\eta > 1$, if $n > \eta(\alpha+f)$, then Algorithm 2 simulates $\mathcal{P}^{\lfloor \frac{\eta}{\eta-1}\alpha \rfloor}_{dyn}$ from $\mathcal{P}^{\alpha}_{dyn}$.*

Proof Let $\xi = \frac{\eta}{\eta-1}$. From $\lfloor \xi\alpha \rfloor > \xi\alpha - 1 = \alpha\frac{\eta}{\eta-1} - 1 = \frac{\alpha\eta-\eta+1}{\eta-1}$ it follows that $\frac{\lfloor \xi\alpha \rfloor + 1}{\lfloor \xi\alpha \rfloor - \alpha + 1} < \eta$. The corollary follows from Lemma 3 by setting $\beta = \lfloor \xi\alpha \rfloor$. □

6.3 Generic predicate

In Section 7 we solve consensus using the following generic predicate, which combines $\mathcal{P}_{\diamond cons}$ and $\mathcal{P}_{\diamond SK}$:

$$\begin{aligned} \mathcal{P}^{f,b,k}_{\diamond cons \oplus SK} ::& \forall \phi > 0, \exists \phi_0 \geq \phi, \\ & CONS((\phi_0-1)k+1) \ \wedge |SK(\Phi)| \geq n-f, \\ & \text{where } \Phi = \{(\phi_0-1)k+1-b, \ldots, \phi_0 k\} \end{aligned}$$

It defines a phase with k rounds, where the first round of some phase ϕ_0 is consistent and all rounds of phase ϕ_0 plus the preceding b rounds have safe kernel of size at least equal to $n-f$.

Obviously, $\mathcal{P}_{\diamond cons \oplus SK}$ can be simulated from $\mathcal{P}_{\diamond SK}$ and $\mathcal{P}_{stat}$ using Algorithm 1, and from $\mathcal{P}_{\diamond SK}$ and $\mathcal{P}_{dyn}$ using Algorithm 2. In both cases, Algorithm 1 or Algorithm 2 simulate the first round of a phase, and a trivial simulation (where messages are just delivered as received) are used for the other rounds. Ensuring that the coordinator is in the safe kernel requires $f+1$ phases. In case of Algorithm 1, the first macro-round of a phase requires 3 rounds, and the others $k-1$ only 1 round. Therefore $f+1$ phases correspond to $(k+2)(f+1)$ rounds. This leads to:

Corollary 4 *If $n > 3f$, then $\mathcal{P}^{f,b,k}_{\diamond cons \oplus SK} \wedge \mathcal{P}^{f}_{stat}$ can be simulated from $\mathcal{P}^{f,K}_{\diamond SK} \wedge \mathcal{P}^{f}_{stat}$, where $K = (k+2)(f+1) + b + (k+1)$.*

Note that the additional term $k+1$ for K stems from the fact that the rounds with a safe kernel are not necessarily aligned to the phases of $\mathcal{P}^{f,b,k}_{\diamond cons \oplus SK}$. In case of Algorithm 2, since the first macro-round requires 4 rounds, we have:

Corollary 5 *If $n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1}$, $n > \alpha+f$, $\alpha \geq f$, and $\beta \geq \alpha$, then $\mathcal{P}^{f,b,k}_{\diamond cons \oplus SK} \wedge \mathcal{P}^{\beta}_{dyn}$ can be simulated from $\mathcal{P}^{f,K}_{\diamond SK} \wedge \mathcal{P}^{\alpha}_{dyn}$, where $K = (k+3)(f+1) + b + (k+2)$.*

Here the additional alignment term in K is $k+2$.

7 Solving consensus with eventual consistency

In this section we use the generic predicate $\mathcal{P}_{\diamond cons \oplus SK}$ to solve consensus. In all consensus algorithms below, the notation $\#(v)$ is used to denote the number of messages received with value v, *i.e.*,

$$\#(v) \equiv \left|\left\{q \in \Pi : \boldsymbol{\mu}^r_p[q] = v\right\}\right|.$$

7.1 The *BOTR* algorithm

We start with the simplest algorithm, namely the *BOTR* algorithm. The basic technique of this algorithm is that a value that is decided is locked in the sense that a sufficiently high quorum of processes retain this value as estimate. A similar algorithmic scheme can be found in algorithms for benign [9,23,16,11] and arbitrary [20,4] faults.

The code of *BOTR* is given as Algorithm 3. It consists of a sequence of phases, where each phase ϕ has two rounds $2\phi-1$ and 2ϕ. Every process p maintains

Algorithm 3 The *BOTR* algorithm

```
 1: Initialization:
 2:    $vote_p \leftarrow init_p \in V$
 3: Round $r = 2\phi - 1$ :
 4:    $S_p^r$:
 5:       send $vote_p$ to all
 6:    $T_p^r$:
 7:       if $|HO(p,r)| \geq T$ then
 8:          $vote_p \leftarrow \min\{v : \nexists v' \in V \; s.t. \; \#(v') > \#(v)\}$
 9: Round $r = 2\phi$ :
10:    $S_p^r$:
11:       send $vote_p$ to all
12:    $T_p^r$:
13:       if $\exists \bar{v} \neq \bot : \#(\bar{v}) \geq T$ then
14:          DECIDE $\bar{v}$
```

a single variable $vote_p$ initialized to p's initial value. In every round, every process p sends $vote_p$ to all. In round $2\phi - 1$, if a process p receives at least T messages then it updates $vote_p$, and sets $vote_p$ to *the smallest most often received value* of the current round. In round 2ϕ, if a process p receives at least T times the same value v then it decides on v.[9]

We will show that *BOTR* is safe (in the sense that it fulfills integrity and agreement) for appropriate choices of T when $\mathcal{P}^{\alpha}_{dyn}$ holds (or $\mathcal{P}^{\alpha}_{stat}$, since $\mathcal{P}^{\alpha}_{stat}$ implies $\mathcal{P}^{\alpha}_{dyn}$). The value of threshold T is chosen such that if some process decides v at line 14 of round r, then in any round $r' \geq r$, at all processes only v can be assigned to any *vote*, and hence only v can be decided. Termination is achieved in both cases if in addition the following predicate holds:

$$\begin{aligned}\mathcal{P}^{f}_{BOTR} :: \; & \forall \phi, \forall p, \exists \phi_0 > \phi : \\ & \mathit{CONS}(2\phi_0 - 1) \; \wedge \; |HO(p, 2\phi_0 - 1)| \geq n - f \\ & \wedge \; (\exists \phi_1 \geq \phi_0 : |\mathit{SHO}(p, 2\phi_1)| \geq n - f)\end{aligned}$$

The $\mathcal{P}^{f}_{BOTR}$ predicate ensures the existence of phases ϕ_0 and $\phi_1 \geq \phi_0$ such that: (i) in the first round of phase ϕ_0 processes receive the same set of at least $n - f$ messages, and (ii) in the second round of phase ϕ_1 processes receive at least $n - f$ uncorrupted messages.

Obviously, $\mathcal{P}^{f,0,2}_{\diamond cons \oplus SK}$ implies $\mathcal{P}^{f}_{BOTR}$. Eventual consistency ensures the first part of the predicate, namely the existence of a consistent round $2\phi_0 - 1$ where in addition every process receives enough messages. This guarantees that at the end of round $2\phi_0 - 1$ all processes adopt the same value for $vote_p$. The second part of the predicate forces every process to make a decision at the end of round $2\phi_1$.

7.1.1 Correctness of the BOTR algorithm

First we introduce some piece of notation. For any variable x local to process p, we denote $x_p^{(r)}$ the value of x_p at the end of round r. For any value $v \in V$ and any process p, at any round $r > 0$, we define the sets $R_p^r(v)$ and $Q_p^r(v)$ as follows:

$$\begin{aligned}R_p^r(v) &:= \left\{q \in \Pi \;:\; \boldsymbol{\mu}_p^r[q] = v\right\} \\ Q_p^r(v) &:= \left\{q \in \Pi \;:\; S_q^r(p, s_q) = v\right\}.\end{aligned}$$

where s_q denotes q's state at the beginning of round r. The set $R_p^r(v)$ (resp. $Q_p^r(v)$) represents the set of processes from which p receives v (resp. which ought to send v to p) at round r. Since at each round of the consensus algorithm, every process sends the same message to all, the sets $Q_p^r(v)$ do not depend on p, and so can be just denoted by $Q^r(v)$ without any ambiguity.

We start our correctness proof with a general basic lemma:

Lemma 5 *For any process p and any value v, at any round r, we have:*

$$|R_p^r(v)| \leq |Q^r(v)| + |AHO(p,r)|$$

Proof Suppose that process p receives a message with value v at round $r > 0$ from process q. Then, either the code of q prescribes it to send v to p at round r, i.e., q belongs to $Q^r(v)$ and thus q is also in $\mathit{SHO}(p,r)$, or the message has been corrupted and q is in $AHO(p,r)$. It follows that $R_p^r(v) \subseteq Q^r(v) \cup AHO(p,r)$, which implies $|R_p^r(v)| \leq |Q^r(v)| + |AHO(p,r)|$. □

As an intermediate step to argue agreement, our next lemma introduces a condition on T that ensures no two processes can decide differently *at the same round*:

Lemma 6 *If $T > \frac{n}{2} + \alpha$ then in any run of the HO machine $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$ there is at most one possible decision value per round.*

Proof Assume by contradiction that there exist two processes p and q that decide on different values v and v' in some round $r > 0$. From the code of *BOTR*, we deduce that $|R_p^r(v)| \geq T$ and $|R_q^r(v')| \geq T$. Then Lemma 5 ensures that $|Q^r(v)| \geq T - |AHO(p,r)|$ and $|Q^r(v')| \geq T - |AHO(q,r)|$.

Since each process sends the same value to all at each round r, the sets $Q^r(v)$ and $Q^r(v')$ are disjoint if v and v' are distinct values . Hence $|Q^r(v) \cup Q^r(v')| = |Q^r(v)| + |Q^r(v')|$. Then from $T > \frac{n}{2} + \alpha$ and since $\mathcal{P}^{\alpha}_{dyn}$ holds, we have $|AHO(p,r)| \leq \alpha$ and $|AHO(q,r)| \leq \alpha$. Therefore, we derive that $|Q^r(v) \cup Q^r(v')| > 2(T - \alpha) > n$, a contradiction. □

[9] The two rounds of *BOTR* algorithm can be merged in a single round in which the code of both state-transition functions is executed at once. We have split them in two rounds to emphasize on the different communication predicates required.

The next lemma shows that once a sufficently high number of processes have the same *vote*, no other value will be adopted in later rounds by any process:

Lemma 7 *If $T > \frac{2}{3}(n + 2\alpha)$, then in any run of the HO machine $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$, if $|\{p' \in \Pi : vote^{(r-1)}_{p'} = v\}| \geq T - \alpha$, every process q that updates its variable $vote_q$ at round r sets it to v.*

Proof Since *vote* is only updated in the first round of a phase, it sufficies to consider the case $r = 2\phi - 1$. Since q updates $vote_q$ in round r, because of line 13, $|HO(q, r)| \geq T$. Let $Q^r(\bar{v})$ denote the set of processes that, according to their sending functions, ought to send messages different from v at round r, and let $R^r_q(\bar{v})$ denote the set of processes from which q receives values different from v at round r. Since each process sends a message to all at each round, $Q^r(\bar{v}) = \Pi \setminus Q^r(v)$, and thus $|Q^r(\bar{v})| = n - |Q^r(v)|$. Similarly, we have $R^r_q(\bar{v}) = HO(q, r) \setminus R^r_q(v)$, and since $R^r_q(v) \subseteq HO(q, r)$, it follows that $|R^r_q(\bar{v})| \leq T - R^r_q(v)$.

Because of the assumption of the Lemma, and the fact that processes send their current value of *vote* in every round, we have $|Q^r(v)| \geq T - \alpha$. It follows that $|Q^r(\bar{v})| \leq n - (T - \alpha)$. With an argument similar to the one used in the proof of Lemma 5, we derive that $|R^r_q(\bar{v})| \leq |Q^r(\bar{v})| + |AHO(q, r)|$. Therefore, we obtain $|R^r_q(\bar{v})| \leq n - T + \alpha + |AHO(q, r)|$.

Since $\mathcal{P}^{\alpha}_{dyn}$ holds, it follows that $|R^r_q(\bar{v})| \leq n - T + 2\alpha$. It follows that because of $T > \frac{2}{3}(n + 2\alpha)$, we have $|R^r_q(\bar{v})| < \frac{1}{3}(n + 2\alpha)$, and therefore $|R^r_q(v)| > |R^r_q(\bar{v})|$. This implies that v is the most frequent value received by q at round r. Then the code entails q to set $vote_q$ to v. □

We now extend the statement of Lemma 7 to hold also for any phase $\phi > \phi_0$:

Lemma 8 *If $T > \frac{2}{3}(n + 2\alpha)$, then in any run of the HO machine $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$ such that process p decides some value v at some phase $\phi_0 > 0$, every process q that updates its variable $vote_q$ at some phase $\phi > \phi_0$ necessarily sets it to v.*

Proof Assume process p decides value v at round $r_0 = 2\phi_0$ of phase ϕ_0. We prove by induction on r that:

$$\forall r \geq r_0,\ |\{q \in \Pi : vote^{(r-1)}_q = v\}| \geq T - \alpha.$$

Then Lemma 7 ensures this Lemma.

Basic case: $r = r_0$. Since p decides v at round r_0, then $|R^{r_0}_p(v)| \geq T$. By Lemma 5, we have $|Q^{r_0}(v)| \geq T - \alpha$ when $\mathcal{P}^{\alpha}_{dyn}$ holds. From the code of $BOTR$, we have $Q^{r_0}(v) = \{q \in \Pi : vote^{(r_0-1)}_q = v\}$, and so the basic case follows.

Inductive step: $r > r_0$. Assume $|\{q \in \Pi : vote^{(r-1)}_q = v\}| \geq T - \alpha$. Lemma 7 ensures that for any process q, $vote_q$ is updated only to v. Thus $|\{q \in \Pi : vote^{(r)}_q = v\}| \geq |\{q \in \Pi : vote^{(r-1)}_q = v\}| \geq T - \alpha$. □

With the help of the previous lemmas, we can show the agreement clause of consensus:

Proposition 1 (Agreement) *If $T > \frac{2}{3}(n+2\alpha)$, then there is at most one possible decision value in any run of the HO machine $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$.*

Proof Let ϕ_0 be the first phase at which some process p makes a decision, and let v be p's decision value. Assume that process q decides v' at phase ϕ. By definition of ϕ_0, we have $\phi \geq \phi_0$.

We proceed by contradiction, and assume that $v \neq v'$. Since $T > \frac{n}{2} + \alpha$, by Lemma 6, we derive that $\phi > \phi_0$. Since p decides v at round $2\phi_0$ and q decides v' at round 2ϕ, Lemma 5 ensures that $|Q^{2\phi_0}(v)| \geq T - |AHO(p, 2\phi_0)|$ and $|Q^{2\phi}(v')| \geq T - |AHO(q, 2\phi)|$.

Since $T > \frac{2}{3}(n+2\alpha)$, Lemma 8 implies that $Q^{2\phi_0}(v)$ and $Q^{2\phi}(v')$ are disjoint sets. Therefore,

$$|Q^{2\phi_0}(v) \cup Q^{2\phi}(v')| = |Q^{2\phi_0}(v)| + |Q^{2\phi}(v')|.$$

Because of $T > \frac{2}{3}(n+2\alpha)$, we have $|Q^{2\phi_0}(v) \cup Q^{2\phi}(v')| > n$, which is a contradiction. □

Now we show that the HO machines $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$ satisfies the integrity clause of consensus for $T > 2\alpha$.

Proposition 2 (Integrity) *If $T > 2\alpha$, then in any run of the HO machine $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$ where all the initial values are equal to some value v_0, the only possible decision value is v_0.*

Proof Consider a run of the HO machine $\langle BOTR, \mathcal{P}^{\alpha}_{dyn}\rangle$ such that all the initial values are equal to v_0.

First, by induction on r, we show that:

$$\forall r > 0 :\ Q^r(v_0) = \Pi$$

Note that according to the code of $BOTR$, p belongs to $Q^r(v_0)$ if and only if $vote^{(r-1)}_p = v_0$, and so $Q^r(v_0) = \{p \in \Pi : vote^{(r-1)}_p = v_0\}$.

Basic case: $r = 1$. All the initial values are equal to v_0. Therefore, every process sends a message with value v_0 at round 1.

Inductive step: $r > 1$. Suppose that $Q^{r-1}(v_0) = \Pi$. Let p be a process that updates its variable x_p at round $r-1$. Since $AHO(p, r-1) \leq \alpha$, each process p receives at most α values distinct from v_0 at round $r-1$. Therefore, either p does not modify $vote_p$ at the end of round r which remains equal to v_0, or p receives at least T messages at round r, and thus at least $T - \alpha$ messages with value v_0 and at most α values different from v_0. In the latter case, p sets $vote_p$ to v_0 since $T > 2\alpha$. This shows that definitely, $vote_p^{(r-1)} = v_0$. Therefore, $Q^r(v_0) = \Pi$.

Let p be a process that makes a decision at some round $r_0 > 0$. We have just shown that $Q^{r_0}(v_0) = \Pi$. When $|AHO(p, r_0)| \leq \alpha$ holds, p receives at most α messages with value different to v_0. Since $T > \alpha$, the code entails p to decide v_0 at round r. □

For liveness, as already stated the communication predicate $\mathcal{P}^f_{BOTR}$ ensures that (i) $vote_q$ are eventually identical, and (ii) each process then hears of sufficiently many processes to make a decision.

Proposition 3 (Termination) *If $n > 4\alpha + 3f$ and $T > \frac{2}{3}(n + 2\alpha)$, then any run of the HO machine $\langle BOTR, \mathcal{P}^f_{BOTR} \wedge \mathcal{P}^\alpha_{dyn}\rangle$ satisfies the Termination clause of consensus.*

Proof Since $n > 4\alpha + 3f$, we have $n - f \geq T$. By $\mathcal{P}^f_{BOTR}$, there exists a phase ϕ_0 such that for all processes p $|HO(p, 2\phi_0 - 1)| \geq n - f \wedge CONS(2\phi_0 - 1)$. Therefore, in round $2\phi_0 - 1$, for any two processes p and q, we have $\boldsymbol{\mu}_p^{2\phi_0-1} = \boldsymbol{\mu}_q^{2\phi_0-1}$, and $|HO(p, 2\phi_0 - 1) \cap HO(q, 2\phi_0 - 1)| \geq n - f \geq T$. The code of $BOTR$ algorithm (see line 8) implies that for all processes p at the end of round $2\phi_0 - 1$ we have $vote_p$ set to the same value v_0.

Because of $T > \frac{2}{3}(n + 2\alpha)$, a similar argument as the one used in Lemma 8 shows that every process q that updates $vote_q$ at round $r' > r_0$ definitely sets it to v_0. Moreover, from $\mathcal{P}^f_{BOTR}$ we have $\forall p \in \Pi$, $\exists \phi_1 \geq \phi_0$ *s.t.* $|SHO(p, 2\phi_1)| \geq n - f \geq T$. Therefore, there exist a round $2\phi_1$ such that every process p in Π eventually receives at least T messages with value v_0 at round $2\phi_1$, and so decides v_0. □

Combining Propositions 1, 2, and 3, we get the following theorem:

Theorem 2 *If $n > 4\alpha + 3f$ and $T > \frac{2}{3}(n + 2\alpha)$, then the HO machine $\langle BOTR, \mathcal{P}^f_{BOTR} \wedge \mathcal{P}^\alpha_{dyn}\rangle$ solves consensus.*

Similar reasoning can be used to show:

Corollary 6 *If $\alpha = f$, $n > 5f$ and $T > \frac{2}{3}(n + f)$, then the HO machine $\langle BOTR, \mathcal{P}^f_{BOTR} \wedge \mathcal{P}^\alpha_{stat}\rangle$ solves consensus.*

7.2 The *BLV* algorithm

The next algorithm we present is called BLV. It is based on the *last voting* mechanism [11] that was first introduced in the seminal Paxos algorithm by Lamport [15] for benign faults. This mechanism is also at the core of the PBFT algorithm by Castro and Liskov [10], the Byzantine variant of the Paxos algorithm.

Algorithm 4 *BLV* algorithm

```
Initialization:
  $vote_p \leftarrow init_p \in V$
  $ts_p \leftarrow 0$
  $history_p \leftarrow \{(init_p, 0)\}$

Round $r = 3\phi - 2$ :
  $S_p^r$:
    send $\langle vote_p, ts_p, history_p\rangle$ to all
  $T_p^r$:
    $select_p \leftarrow \mathcal{FBLV}_{T,\alpha}(\boldsymbol{\mu}_p^r)$
    if $select_p \neq null$ then
      $history_p \leftarrow history_p \cup \{(select_p, \phi)\}$

Round $r = 3\phi - 1$ :
  $S_p^r$:
    if $\exists(v, \phi) \in history_p$ then
      send $\langle v\rangle$ to all
  $T_p^r$:
    if $\#(v) \geq T$ then
      $vote_p \leftarrow v$
      $ts_p \leftarrow \phi$

Round $r = 3\phi$ :
  $S_p^r$:
    if $ts_p = \phi$ then
      send $\langle vote_p\rangle$ to all
  $T_p^r$:
    if $\exists \bar{v} \neq \perp$ : $\#(\bar{v}) \geq T$ then
      DECIDE $\bar{v}$
```

BLV is designed to work both under $\mathcal{P}^\alpha_{stat}$ and $\mathcal{P}^\alpha_{dyn}$. It requires $n > 2(\alpha + f)$ and $T > \frac{n}{2} + \alpha$ in the presence of dynamic value faults ($\mathcal{P}^\alpha_{dyn}$), or $n > 3f$, $\alpha = f$, and $T > \frac{n+f}{2}$ if value faults are only static ($\mathcal{P}^\alpha_{stat}$). Termination is achieved with $\mathcal{P}^\alpha_{dyn}$ or $\mathcal{P}^\alpha_{stat}$ if in addition the following predicate holds:

$$
\begin{aligned}
\mathcal{P}^f_{BLV} :: &\forall\phi > 0, \exists\phi_0 > \phi : CONS(3\phi_0 - 2)\\
&\wedge \forall r \in \{3\phi_0 - 2, \dots, 3\phi_0\} : |SK(r)| \geq n - f
\end{aligned}
$$

The $\mathcal{P}^f_{BLV}$ predicate ensures the existence of a phase ϕ_0 such that: (i) in the first round of ϕ_0 processes receive the same set of messages, and (ii) in all three rounds of ϕ_0 processes receive at least $n - f$ uncorrupted messages.

Obviously, $\mathcal{P}^{f,0,3}_{\diamond cons \oplus SK}$ implies $\mathcal{P}^f_{BLV}$. Eventual consistency ensures that at the end of round $3\phi_0 - 2$, all processes select the same value. The condition that there exists a large enough safe kernel in phase ϕ_0 finally forces every process to make a decision at the end of round $3\phi_0$.

The code of BLV is given as Algorithm 4. It consists of a sequence of phases, where each phase ϕ has three

Algorithm 5 Function $\mathcal{FBLV}_{T,\alpha}(\mathbf{M})$

```
30: $possibleV \leftarrow \{(v, ts) : \exists i \in \Pi : (v, ts) = \mathbf{M}[i] \wedge |\{q : \mathbf{M}[q] = (v', ts', -) \wedge ((v = v' \wedge ts = ts') \vee ts > ts')\}| \geq T\}$
31: $confirmedV \leftarrow \{v : (v, ts) \in possibleV \wedge |\{q : \mathbf{M}[q] = (-, -, history) \wedge (v, ts) \in history\}| > \alpha\}$

32: if $|confirmedV| \geq 1$ then
33:     return $\min(confirmedV)$
34: else if $\{q : \mathbf{M}[q] = (-, 0, -)\} \geq T$ then
35:     return minimal $v$, such that $\exists (v, 0, -) \in \mathbf{M}$ and $\nexists (v', 0, -) \in \mathbf{M}$ $s.t.$ $\#((v', 0, -)) > \#((v, 0, -))$
36: else
37:     return $null$
```

rounds $3\phi - 2$, $3\phi - 1$ and 3ϕ. The *last voting* mechanism uses a timestamp variable ts in addition to to the variable $vote$. Whenever a process p updates $vote_p$ in round $3\phi - 1$, ts_p is set to ϕ (line 21 and 22). If enough processes update $vote$ in round $3\phi - 1$, then a decision is possible in phase 3ϕ. This is the same mechanism as in *Paxos*. Note the condition at line 20. It ensures that in round $3\phi - 1$, all processes that update $vote$, update it to the same value. As in Paxos, this ensures that in round 3ϕ, processes attempt to decide on one single value, which is necessary for agreement.

In order to deal with value faults, BLV maintains also a *history* variable, which stores pairs (v, ϕ). Having $(v, \phi) \in history_p$ means that p added (v, ϕ) to $history_p$ in phase ϕ (line 11). The *history* variable ensures that a corrupted message with invalid values for $vote$ and ts_p will not affect the safety properties of the algorithm. It is mainly used in round $3\phi - 2$, which has two roles, the first related to agreement and integrity, and the second related to termination:

1. *Safety role:*
   (a) *Agreement*: If a process p has decided v in some phase ϕ_0, then for any process q, only v can be assigned to $select_q$ at line 9 in phases $\phi > \phi_0$.
   (b) *Integrity*: If all process have the same initial value v, then only v can be assigned to $select_p$ at line 9.
2. *Termination role*: In a consistent round with safe kernel of size $n - f$, all processes must assign the same value to $select$ at line 9.

Line 9 refers to the selection function $\mathcal{FBLV}_{T,\alpha}$, which takes as input the messages received in round $3\phi - 2$. We explain now this function (Algorithm 5):

- Line 30 (together with line 31) ensures 1a. More precisely, it ensures selection of the most recent vote in the history of some process. This is basically the same mechanism as in Paxos, adapted to transmission value faults. Selecting the most recent vote among the set of majority processes can be expressed in Paxos as follows:

  $$mostRecentV \leftarrow \Big\{(v, ts) : \ (v, ts) \in \boldsymbol{\mu}_p^r \ \wedge$$
  $$|\{q : \boldsymbol{\mu}_p^r[q] = (v', ts') \wedge ts \geq ts')\}| > \frac{n}{2}\Big\}$$

  In Paxos, this selection rule ensures agreement since most recent vote is a single value. In BLV, a corrupted message can contain $(vote, ts)$ with ts equal to the highest timestamp of a process, but with a different $vote$. Therefore, the above selection rule does not ensure 1a, since several values can satisfy the condition of lines 30 and 31. The solution consists in transforming condition $ts \geq ts'$ into $(v = v' \wedge ts = ts') \vee ts > ts'$ and using a higher threshold ($T > \frac{n}{2} + f$ or $T > \frac{n+f}{2}$).
  With this, if a process has previously decided $\bar{v}$, then only $\bar{v}$ can be in *confirmedV*.[10]
- Line 31 prevents from returning a value v from a pair (v, ts) that is from a corrupted message: the pair must be in the history of at least one process. Therefore, a pair (v, ts) is considered only if it is part of the history in at least $\alpha + 1$ messages received. Together with line 30, it also ensures 1b: when all processes have the same initial value, no other value is in the $history_p$ variable of processes.

We consider now lines 32 and 33 of Algorithm 5. As we just explained, if a process has previously decided $\bar{v}$, then only $\bar{v}$ can be in *confirmedV*, that is, $|confirmedV| = 1$. In this case, by line 33, the function $\mathcal{FBLV}_{T,\alpha}$ returns $\bar{v}$. If no correct process has decided, we can have $|confirmedV| > 1$. In this case, if some round $3\phi - 2$ is a consistent round with safe kernel of size $n - f$, then all processes consider the same set *confirmedV*, which ensures 2. Lines 34 and 35 are for the case where not all processes have the same initial value. Termination would be violated without these lines.

[10] Consider two phases ϕ_0 and $\phi_0 + 1$, such that a process has decided $\bar{v}$ in phase ϕ_0. We consider the more general case in the presence of dynamic faults, and we assume that $n = 5$, $f = \alpha = 1$ and $T = 4$. This means that at least $T - \alpha = 3$ processes have $ts = \phi_0$ and $vote = \bar{v}$. Consider in phase $\phi_0 + 1$ that $(v, ts) \in possibleV_p$ at p with $v \neq \bar{v}$. This means that p, in round $3(\phi_0 + 1) - 2$, has received $T = 4$ messages with either $(v, ts, -)$, or $(-, ts', -)$ and $ts' < ts$. Since $n = 5$ and $T = 4$, at least one of these messages is from a process c such that $vote_c = \bar{v}$ and $ts_c = \phi_0$. Since $v \neq \bar{v}$, we must have $\phi_0 < ts$. However, in phase $\phi_0 + 1$, no process p can have (v, ts) with $ts > \phi_0$ in $history_p$. Therefore, by line 31, we will not have $v \in confirmedV$.

7.2.1 Correctness of the BLV algorithm

In this section we use the same definition of $R(v)$ and $Q(v)$ as in Section 7.1.1.

Definition 1 A value v is locked in a phase ϕ by process p if $vote_p = v$ and $ts_p = \phi$ at the end of round $3\phi - 1$.

Lemma 9 *If $T > \frac{n}{2} + \alpha$, then in any run of the HO machine $(BLV, \mathcal{P}^{\alpha}_{dyn})$ there is at most one locked value per phase.*

Proof Assume by contradiction that there exist two processes p and q that lock different values v and v' in some phase $\phi_0 > 0$. From line 20 we deduce that $|R^{3\phi_0}_p(v)| \geq T$ and $|R^{3\phi_0}_q(v')| \geq T$. Then Lemma 5 (note that this lemma holds also for BLV) ensures that $|Q^{3\phi_0}(v)| \geq T - \alpha$ and $|Q^{3\phi_0}(v')| \geq T - \alpha$ when $\mathcal{P}^{\alpha}_{dyn}$ holds.

Since each process sends the same value to all at each round, the sets $Q^{3\phi_0}(v)$ and $Q^{3\phi_0}(v')$ are disjoint if v and v' are distinct values. Hence,

$$|Q^{3\phi_0}(v) \cup Q^{3\phi_0}(v')| = |Q^{3\phi_0}(v)| + |Q^{3\phi_0}(v')| \geq 2T - 2\alpha.$$

Consequently, since $T > \frac{n}{2} + \alpha$, we derive that $|Q^{3\phi_0}(v) \cup Q^{3\phi_0}(v')| > n$, a contradiction. □

Lemma 10 *If $T > \alpha$, then in any run of the HO machine $(BLV, \mathcal{P}^{\alpha}_{dyn})$ there is at most one possible decision value per phase.*

Proof Assume by contradiction that there exist two processes p and q that decide on different values v and v' in some phase $\phi_0 > 0$. From line 28 we deduce that $|R^{3\phi_0}_p(v)| \geq T$ and $|R^{3\phi_0}_q(v')| \geq T$. Then Lemma 5 and $\mathcal{P}^{\alpha}_{dyn}$ ensure that $|Q^{3\phi_0}(v)| \geq T - \alpha$ and $|Q^{3\phi_0}(v')| \geq T - \alpha$.

Since each process sends the same value to all at each round, the sets $Q^{3\phi_0}(v)$ and $Q^{3\phi_0}(v')$ are disjoint since v and v' are distinct values. Hence, when $T > \alpha$, the sets $Q^{3\phi_0}(v)$ and $Q^{3\phi_0}(v')$ are not empty, and so by line 25, there exist two processes p' and q' that have $vote_{p'} = v$, $ts_{p'} = \phi_0$, $vote_{q'} = v'$ and $ts_{q'} = \phi_0$. A contradiction with Lemma 9. □

Lemma 11 *If $T > \frac{n}{2} + \alpha$, then in any run of the HO machine $(BLV, \mathcal{P}^{\alpha}_{dyn})$, if process p decides v in phase $\phi_0 > 0$, then for all later phases $\phi > \phi_0$ and all processes q, (v, ϕ) is the only pair that can be added to $history_q$.*

Proof Assume by contradiction that $\phi_1 > \phi_0$ is the first phase where a pair (v_1, ϕ_1) with $v_1 \neq v$ is added to the $history_q$ at process q. This implies that if *history* at some process contains a pair (v', ϕ') with $v' \neq v$, then $\phi' \leq \phi_0$ (*).

Since by our assumption q added $(v', -)$ to $history_q$ in phase ϕ_1, this implies that $\mathcal{FBLV}_{T,\alpha}$ returns v' at line 9 in phase ϕ_1. Therefore, either (i) line 33 or (ii) line 35 of Algorithm 5 was executed by q in phase ϕ_1.

In case (ii), the condition of line 34 has to be true. This implies that $|R^{3\phi_1-2}_q((-, 0, -))| \geq T$, and thus, by Lemma 5, $|Q^{3\phi_1-2}((-, 0, -))| \geq T - \alpha$.

> We prove an intermediate result: In phases ϕ such that $\phi_0 \leq \phi < \phi_1$, we have $|\{q \in \Pi : vote^{3\phi-1}_q = v \wedge ts^{3\phi-1}_q \geq \phi_0\}| \geq T - \alpha$. Since p decides v in phase ϕ_0, $|R^{3\phi_0}_p(v)| \geq T$, and thus by Lemma 5, we have $|Q^{3\phi_0}(v)| \geq T - \alpha$. From the code of the BLV algorithm, we have $Q^{3\phi_0}(v) = \{q : vote^{3\phi_0-1}_q = v \wedge ts^{3\phi_0-1}_q = \phi_0\}$, therefore the claim holds for phase ϕ_0.
> We now show that any process that locked value v in phase ϕ_0 (see Definition 1) and updates *vote* in phase ϕ such that $\phi_0 < \phi < \phi_1$, sets it to v. This ensures the claim. Assume by contradiction that one of these processes q' sets $vote_q$ to v' in round $3\phi - 1$. By line 20, $|R^{3\phi-1}_q(v')| \geq T$. Then Lemma 5 ensures that $|Q^{3\phi-1}(v')| \geq T - \alpha$. Since $T > \alpha$, we have $|Q^{3\phi-1}(v')| > 0$, i.e. at least one process sent v' at line 18. Therefore, by line 17 at least one process has $(v', \phi_0 + 1)$ in *history*, a contradiction with the assumption that ϕ_1 is the first phase where a pair $(v', -)$ is added to *history* at some process.

So we have also $|\bigcup_{ts \geq \phi_0} Q^{3\phi_1-2}((v, ts, -))| \geq T - \alpha$. Since in each round, every process sends the same value to all, and $\phi_0 > 0$, the sets $X(v) = \bigcup_{ts \geq \phi_0} Q^{3\phi_1-2}((v, ts, -))$ and $Q^{3\phi_1-2}((-, 0, -))$ are disjoint. Hence,

$$|X(v) \cup Q^{3\phi_1-2}((-, 0, -))| =$$
$$|X(v)| + |Q^{3\phi_1-2}((-, 0, -))| \geq 2T - 2\alpha.$$

Together with $T > \frac{n}{2} + \alpha$, we derive that $|\bigcup_{ts \geq \phi_0} Q^{3\phi_1-2}((v, ts, -)) \cup Q^{3\phi_1-2}((-, 0, -))| > n$, a contradiction.

In case (i), the condition at line 32, has to be true, i.e., v' need to be part of *confirmedV* set at line 31. Value v' can be part of the set *confirmedV* only if (v', ts') is part of the set *possibleV* at line 30. We show that if (v', ts') is part of the set *possibleV* at line 30, v' cannot be part of the set *confirmedV* at line 31, which establishes the contradiction.

If the pair (v', ts') is added to the set *possibleV* at line 30, then $HO(q, 3\phi_1 - 2) \geq T$. Since $2T - \alpha > n + \alpha$, $|HO(q, 3\phi_1 - 2) \cap \bigcup_{ts \geq \phi_0} Q^{3\phi_1-2}((v, ts, -))| > \alpha$. Therefore, since $\mathcal{P}^{\alpha}_{dyn}$ holds, any set of messages of size

T contains at least one message m with $m.vote = v$ and $m.ts \geq \phi_0$ (**). So we have $|\{m' \in \boldsymbol{\mu}_q^r : (m'.ts < ts')\}| \geq T$ and, because of (**), $ts' > \phi_0$.

The value v' is added to the set *confirmedV* at line 31 only if there are at least $\alpha + 1$ messages $\overline{m}$ in $\boldsymbol{\mu}_q^{3\phi-2}$ such that: $(\overline{v}, \overline{\phi}) \in \overline{m}.history$ and $\overline{\phi} \geq ts'$ and $\overline{v} = v'$. Since $ts' > \phi_0$, by (*), q receives at most α such messages, a contradiction. □

Proposition 4 (Agreement) *If $T > \frac{n}{2} + \alpha$, then no two processes can decide differently in any run of the HO machine $(BLV, \mathcal{P}_{dyn}^{\alpha})$.*

Proof Let a phase $\phi_0 > 0$ be the first phase at which some process p makes a decision, and let v be the p's decision value. Assume that process q decides v' at phase ϕ'. By definition of ϕ_0, we have $\phi' \geq \phi_0$.

We proceed by contradiction and assume that $v \neq v'$. By Lemma 10, we derive that $\phi' > \phi_0$. Since q decides at round $3\phi'$, by line 28 we have $|R_q^{3\phi'}(v')| \geq T$. By Lemma 5, we have $|Q^{3\phi'}(v')| \geq T - \alpha$. Since $T > \alpha$, there is at least one process p' that sends v' in round $3\phi'$. By line 25 and line 22 we have that process p' sends it's current *vote* in round $3\phi'$ only if *vote* is updated in round $3\phi' - 1$. Therefore, $|R_{p'}^{3\phi'-1}(v')| \geq T$, i.e. by Lemma 5, we have $|Q^{3\phi'-1}(v')| \geq T - \alpha$. Since $T > \alpha$, at least one process q' sends v' in round $3\phi' - 1$. By line 17, if q' sends v' in round $3\phi' - 1$, then $\exists (v', \phi') \in history_{q'}^{3\phi'-2}$, a contradiction with Lemma 11. □

Lemma 12 *If $T > 2\alpha$, then in any run of the HO machine $(BLV, \mathcal{P}_{dyn}^{\alpha})$ where all the initial values are equal to some value v, for all processes q, $history_q$ contains only pairs $(v, -)$.*

Proof Since all processes have v as their initial value, *history* at all processes is initialized to $(v, 0)$. Assume by contradiction that ϕ_0 is the first phase where a pair $(v', -)$ is added to $history_p$ at some process p (*). This implies that $\mathcal{FBLV}_{T,\alpha}$ returns v' at line 9. Therefore, either (i) line 33 or (ii) line 35 of Algorithm 5 was executed by p in phase ϕ_0.

For (i), the condition at line 32 has to be true, i.e., v' needs to be in *confirmedV* at line 31. This means that p received more than α messages $m = (-, -, history_m)$ with $(v', ts) \in history_m$ in round $3\phi_0 - 2$. By Lemma 5 and $\mathcal{P}_{dyn}^{\alpha}$, at least one process sends a message $m = \langle -, -, history_m \rangle$ with $(v', ts) \in history_m$ in round $3\phi_0 - 2$, a contradiction with (*).

For (ii), the condition of line 34 has to be true. If this condition is true, this implies that $|HO(p, 3\phi_0 - 2)| \geq T$. Since $T > 2\alpha$, $\mathcal{P}_{dyn}^{\alpha}$ holds, and all processes have the same initial value v, v is returned at line 35 and (v, ϕ_0) is added to the $history_p$. A contradiction. □

Proposition 5 (Integrity) *If $T > 2\alpha$, then in any run of the HO machine $(BLV, \mathcal{P}_{dyn}^{\alpha})$ where all the initial values are equal to some value v, the only possible decision value is v.*

Proof By contradiction, assume that phase $\phi_0 > 0$ is the first phase in which some process p decides $v' \neq v$.

Since p decides at round $3\phi_0$, by line 28 we have $|R_p^{3\phi_0}(v')| \geq T$. By Lemma 5 and $\mathcal{P}_{dyn}^{\alpha}$, we have $|Q^{3\phi_0}(v')| \geq T - \alpha$. Since $T > \alpha$, there is at least one process q that sends v' in round $3\phi_0$. By line 25 and line 22, we have that process q sends it's current *vote* in round $3\phi_0$ only if *vote* is updated in round $3\phi_0 - 1$. Therefore, $|R_q^{3\phi_0-1}(v')| \geq T$, i.e., by Lemma 5 and $\mathcal{P}_{dyn}^{\alpha}$, we have $|Q^{3\phi_0-1}(v')| \geq T - \alpha$. Since $T > \alpha$, at least one process q' sends v' in round $3\phi_0 - 1$. By line 17, if q' sends v' in round $3\phi_0 - 1$, then $\exists (v', \phi_0) \in history_{q'}^{3\phi_0-2}$, a contradiction with Lemma 12. □

Proposition 6 (Termination) *If $n > 2(f + \alpha)$, $T > \frac{n}{2} + \alpha$ and $f \leq \alpha$, then any run of of the HO machine $(BLV, \mathcal{P}_{dyn}^{\alpha} \wedge \mathcal{P}_{BLV}^{f})$ satisfies the Termination clause of consensus.*

Proof By $\mathcal{P}_{BLV}^{f}$, there exists a phase ϕ_0 such that

$$CONS(3\phi_0 - 2) \wedge \forall r \in \{3\phi_0 - 2, \ldots, 3\phi_0\} : SK(r) \geq n - f.$$

Therefore, in round $3\phi_0 - 2$, for any two processes p and q, we have $\boldsymbol{\mu}_p^r = \boldsymbol{\mu}_q^r$, and $|SHO(p, 3\phi_0 - 2) \cap SHO(q, 3\phi_0 - 2)| \geq n - f$.

Part A. We now prove that $select_p^{3\phi_0-2}$ will be the same at all processes p, i.e., that $\mathcal{FBLV}_{T,\alpha}$ returns the same value at all processes, and all processes add the same pair to *history* in round $3\phi_0 - 2$. There are two cases to consider: (i) some process $p \in SK(\phi_0)$ locked a value in some phase smaller than ϕ_0, or (ii) there is no such process in $SK(\phi_0)$.

Case (i): Let $\phi < \phi_0$ be the largest phase in which some process p locked some value v (line 21). By Lemma 9 and since $Q > \frac{n}{2} + \alpha$, all processes that lock a value in phase ϕ, lock the same value v. Since $n > 2(f + \alpha)$ and $T > \frac{n}{2} + \alpha$, $n - f \geq T$; therefore in case (i) at least a pair $(v, -)$ is added to the set *possibleV* at line 30 of Algorithm 5 (*).

We consider now line 31 of Algorithm 5. If p locked value v in phase ϕ, then $|R_p^{3\phi-1}(v)| \geq T$, i.e., by Lemma 5, we have $|Q^{3\phi-1}(v)| \geq T - \alpha$ when $\mathcal{P}_{dyn}^{\alpha}$ holds. Because of line 17 of Algorithm 4, at least $T - \alpha$ processes have (v, ϕ) in *history*. By assumption, $n > 2(f + \alpha)$ and $T > \frac{n}{2} + \alpha$, therefore $n - f + T > n + \alpha$. Therefore, because of $|SK(\phi_0)| \geq n - f$, any set of messages received in round $3\phi_0 - 2$ contains more than α messages m with $(v, \phi) \in m.history$. Since $n > 2(f + \alpha)$ and $T > \frac{n}{2} + \alpha$,

$n - f \geq T$ (***), and therefore v is added to the set *confirmedV* at line 31 of Algorithm 5 (**).

From (*) and (**), it follows that the condition of line 32 of Algorithm 5 is true at all processes in phase ϕ_0. Moreover, since function $\mathcal{FBLV}_{T,\alpha}$ is deterministic and $CONS(3\phi_0 - 2)$ holds, for any two processes p and q, we have $select_p = select_q$ at line 9. Therefore p and q add the same pair to *history* at line 11.

Case (ii): By hypothesis, for all processes $p \in SK(\phi_0)$, we have $ts_p = 0$. By (***) $n - f \geq T$ and therefore the condition at line 34 of Algorithm 5 is true at each process. Moreover, by $CONS(3\phi_0 - 2)$ we have for any two processes p and q $\boldsymbol{\mu}_p^r = \boldsymbol{\mu}_q^r$. Therefore, the value returned at line 35 of Algorithm 5 is the same at all processes, and they will add the same pair to *history* at line 11 of Algorithm 4.

Part B. From Part A, there exists a value v such that at all processes p we have $(v, \phi_0) \in history_p$ at the beginning of round $3\phi_0 - 1$. Therefore all processes send v to all at line 18. By $|SK(3\phi_0 - 1)| \geq n - f$ we have that all processes receive at least $n - f$ messages equal to v, and since by (***) $n - f \geq T$, they all set $vote_p$ to v (line 21) and send v to all at line 26. By $|SK(3\phi_0)| \geq n - f$ and the same reasoning we can show that all processes receive $n - f$ messages equal to v in round $3\phi_0$, and since by (***) $n - f \geq T$, decide v at line 29 in phase ϕ_0. □

Combining Propositions 4, 5, and 6, we get the following theorem:

Theorem 3 *If $n > 2(\alpha + f)$ and $T > \frac{n}{2} + \alpha$, then the HO machine $\langle BLV, \mathcal{P}_{BLV}^f \wedge \mathcal{P}_{dyn}^\alpha \rangle$ solves consensus.*

Similar reasoning can be used to show:

Theorem 4 *If $\alpha = f$, $n > 3f$ and $T > \frac{n+f}{2}$, then the HO machine $\langle BLV, \mathcal{P}_{BLV}^f \wedge \mathcal{P}_{stat}^\alpha \rangle$ solves consensus.*

Note that the BLV algorithm can also be used in the model considered in [4], where all faults are transient. By Theorem 3, the BLV algorithm solves consensus in this model if $n > 2\alpha$ ($f = 0$), in contrast to algorithm $\mathcal{A}_{T,E}$ in [4], which requires $n > 4\alpha$. Algorithm $\mathcal{U}_{T,E,\alpha}$ in [4] requires $n > 2\alpha$ but, contrary to BLV, requires for safety that in every round every process receives a sufficient number of correct messages. This is not required by BLV, which is still correct even if processes do not receive any correct message in some rounds.

7.3 The BLK algorithm

The third algorithm we present is called BLK. It is based on locking/unlocking mechanism that was first introduced in the seminal consensus algorithm for benign and arbitrary faults given by Dwork, Lynch and Stockmayer [12].

It requires $n > 2(\alpha + f)$ and $T > \frac{n}{2} + \alpha$ in the presence of dynamic value faults ($\mathcal{P}_{dyn}^\alpha$), or $n > 3f$, $\alpha = f$ and $T > \frac{n+f}{2}$ if value faults are only static ($\mathcal{P}_{stat}^\alpha$). The code of BLK is given as Algorithm 6. It consists of a sequence of phases, where each phase ϕ has three rounds $3\phi - 2$, $3\phi - 1$, and 3ϕ. In addition to the variable *vote*, and similarly to BLV, the algorithm maintains a timestamp *ts* and a *history* variable. In round $3\phi - 2$, every process p sends $\langle vote_p, init_p \rangle$ to all, where $init_p$ is p's initial value. It is maybe surprising to see that also the initial value $init_p$ is sent in the first round. The initial value is used only when $vote_p =$ NONE, as can be seen in the selection function $\mathcal{FBLK}_T$ (Algorithm 7). A value selected in round $3\phi - 2$ (lines 9 and 15) is sent to all in round $3\phi - 1$. If in round $3\phi - 1$, a process p receives at least T messages equal to some value v, it sets $vote_p$ to v and ts_p to ϕ (lines 18 and 19). Then we say that process p *locked* value v in phase ϕ. If $vote_p =$ NONE then process p has not locked any value. In round 3ϕ, a process p sends $\langle vote_p, ts_p, history_p \rangle$ to all processes. If some value v is locked in phase ϕ by sufficiently high quorum of processes, then a decision is possible in phase ϕ (line 24).

A value can be unlocked by process p in round 3ϕ, if p learns that some process q locked different value in higher phase ($ts_q > ts_p \wedge vote_q \neq vote_p$). In addition to *vote* and *ts*, BLK maintains the *history* variable, which stores pairs (v, ϕ). Having $(v, \phi) \in history_p$ means that p selected v in round $3\phi - 2$ and added (v, ϕ) to $history_p$ in phase ϕ (line 11). It is used to filter out corrupted pairs $(vote, ts)$ at round 3ϕ.

It can be shown, using similar technique as for BLV, that BLK is safe (it fulfills integrity and agreement) for appropriate choice of T when $\mathcal{P}_{dyn}^\alpha$ holds (or $\mathcal{P}_{stat}^\alpha$, since $\mathcal{P}_{stat}^\alpha$ implies $\mathcal{P}_{dyn}^\alpha$). Termination is achieved in both cases if in addition the following predicate holds:

$$\begin{aligned}\mathcal{P}_{BLK}^f ::&\forall \phi > 0, \exists \phi_0 > \phi : CONS(3\phi_0 - 2)\\ &\wedge \forall r \in \{3\phi_0 - 3, \dots, 3\phi_0\} : |SK(r)| \geq n - f\end{aligned}$$

The $\mathcal{P}_{BLK}^f$ predicate ensures the existence of a phase ϕ_0 such that: (i) in the first round of ϕ_0 processes receive the same set of messages, (ii) in all three rounds of ϕ_0 processes receive correctly messages from at least $n - f$ processes, and (iii) in the last round of phase $\phi_0 - 1$ processes receive at least $n - f$ uncorrupted messages.

Obviously, $\mathcal{P}_{gen}^{1,3}$ implies $\mathcal{P}_{BLK}^f$. Eventual consistency ensures that at the end of round $3\phi_0 - 2$, all processes set $select_p$ to the same value. $\mathcal{P}_{BLK}^f$ also ensures a large enough safe kernel in the last round of the *previous*

Algorithm 7 Function $\mathcal{FBLK}_T(\mathcal{M})$

```
1: $validV \leftarrow \{m.vote$ s.t. $m \in \mathcal{M}$ and $|m' \in \mathcal{M}$ s.t. $m'.vote = m.vote$ or $m'.vote = \text{None}| \geq T\}$
2: if $|validV| > 0$ then
3:    if None $\in validV$ then
4:       return minimal $v$, such that $\exists(-, v) \in \mathbf{M}$ and $\nexists(-, v') \in \mathbf{M}$ $s.t.$ $\#((-, v')) > \#((-, v))$
5:    else
6:       return $min(validV)$
7: else
8:    return NULL
```

Algorithm 6 *BLK* algorithm

```
1: Initialization:
2:    $vote_p \leftarrow init_p \in V$
3:    $ts_p \leftarrow 0$
4:    $history_p \leftarrow \emptyset$
5: Round $r = 3\phi - 2$ :
6:    $S_p^r$:
7:       send $\langle vote_p,\ init_p \rangle$ to all
8:    $T_p^r$:
9:       $select_p \leftarrow \mathcal{FBLK}_T(\boldsymbol{\mu}_p^r)$
10:      if $select_p \neq$ NULL then
11:         $history_p \leftarrow history_p \cup \{(select_p, \phi)\}$
12: Round $r = 3\phi - 1$ :
13:   $S_p^r$:
14:      if $\exists(v, \phi) \in history_p$ then
15:         send $\langle v \rangle$ to all
16:   $T_p^r$:
17:      if $\#(v) \geq T$ then
18:         $vote_p \leftarrow v$
19:         $ts_p \leftarrow \phi$
20: Round $r = 3\phi$ :
21:   $S_p^r$:
22:      send $\langle vote_p,\ ts_p,\ history_p \rangle$ to all
23:   $T_p^r$:
24:      if $\exists \bar{v} \neq \bot$ : $\#(\langle \bar{v}, \phi, - \rangle) \geq T$ then
25:         Decide $\bar{v}$
26:      if$((\exists \langle v',\ ts,\ - \rangle \in \boldsymbol{\mu}_p^r$ $s.t.$ $vote_p \neq v' \wedge ((ts >$
         $ts_p) \vee ts_p = 0)$ and
            $(|\{m : m \in \boldsymbol{\mu}_p^r \wedge (v', ts) \in m.history\}| > \alpha))$ or
            (received at least $T$ $\langle v,\ 0,\ - \rangle$ $s.t.$
               $\nexists \bar{v} : |\langle \bar{v}, 0, - \rangle| > \alpha)$ then
27:      $vote_p \leftarrow$ None
28:      $ts_p \leftarrow 0$
```

phase $\phi_0 - 1$. The role of this round is to ensure that all processes either lock the same value (those with the highest timestamp), or they do not lock any value. The condition that there exists a large enough safe kernel in phase ϕ_0 finally forces every process to make a decision at the end of round $3\phi_0$.

The proof of correctness follows a similar pattern as for *BLV* and is not repeated here.

BLK versus BLV. There are strong similarities between BLV and BLK: three rounds per phase, only round $3\phi-2$ must eventually be a consistent round, the *history* variable. However, the mechanisms for agreement differ: BLV uses a *last voting* mechanism, while BLK employs a *locking* mechanism. The two mechanisms are used in round $3\phi - 2$, when assigning a value to *select* (line 9):

– The last voting mechanism uses *vote* and *ts* (mechanism of *PBFT* and *Paxos*).
– The locking mechanism uses only *vote* (mechanism introduced in [12]).

This difference has consequences in the information sent in round $3\phi - 2$: in BLV, $\langle vote_p, ts_p, history_p \rangle$ is sent; in BLK, only $\langle vote_p, init_p \rangle$ is sent. The initial value is only needed when several correct processes do not have a locked value ($vote =$ None) as can be seen in Algorithm 7 (see line 3 and 4).

To illustrate the difference between the two mechanisms, consider the case with dynamic value faults where $n = 5$, $\alpha = f = 1$, $T = 4$ and some process p_1 has decided v_1 at the end of phase ϕ_1. A possible configuration of processes p_1 to p_5 for the two algorithms at the end of phase ϕ_1 is the following:

$$(v_1, \phi_1),\ (v_1, \phi_1),\ (v_1, \phi_1)\ (v_2, \phi_2), (v_2, \phi_2)$$

where each tuple represents the states $(vote, ts)$ and $\phi_2 < \phi_1$.[11] The history at $T - \alpha = 3$ processes contains the pair (v_1, ϕ_1). In round $3(\phi_1 + 1) - 2$ of the BLV algorithm, let a process p_2 receive, from processes p_1 to p_5 (the message received from process p_5 is corrupted):

$$(v_1, \phi_1, -),\ (v_1, \phi_1, -),\ (v_1, \phi_1, -),\ (v_2, \phi_2, -),\ (v_2, \phi_1, -).$$

With the last voting mechanism, we have $v_1 \in confirmedV$ (there are 4 messages with $vote = v_1$ or $ts < \phi_1$ and (v_1, ϕ_1) is in *history* of the message sent by three processes), and $select_p$ is set to v_1. Assume that similarly, in round $3(\phi_1 + 1) - 2$ of the BLKalgorithm, process p_2 receives, from processes p_1 to p_5 (all messages are correctly received):

$$(v_1, -),\ (v_1, -),\ (v_1, -),\ (v_2, -),\ (v_2, -).$$

With the locking mechanism, *validV* in Algorithm 7 is empty (there are no four messages with $vote = v_1$),

[11] Process p_1 decided v_1 by receiving correctly messages from processes p_1,p_2 and p_3 and the corrupted message $\langle v_1, \phi_1, - \rangle$ from p_4.

and NULL is returned. With the locking mechanism, processes p_1, p_2 and p_3 have "locked" v_1, while processes p_4 and p_5 has "locked" v_2. It is clear that as long as processes p_4 and p_5 have locked v_2, no additional process can decide. Therefore, an *unlocking* mechanism is needed. This is the role of lines 26 and 28 of Algorithm 6. If process p_4 receives a message $(v, ts, -)$ from a process with (v, ts) in *history* of $\alpha + 1$ messages received, and $vote_{p_4} \neq v$, $ts_{p_4} < ts$, then process p_4 unlocks $vote_{p_4}$ by setting the variable to NONE (line 28). The second part of condition at line 26 is for the case where not all processes have the same initial value (termination would be violated without it). Now in round $3(\phi_1 + 1) - 2$, let a process receive, from processes p_1 to p_5:

$$(v_1, -),\ (v_1, -),\ (v_1, -),\ (\text{NONE}, -),\ (v_2, -).$$

This leads to have $v_1 \in validV$, and $select_p$ is set to v_1.

Observe that the unlocking mechanism requires $history_p$ (line 22). Therefore, we can also summarize the two mechanisms by saying that the last voting mechanism requires $history_p$ in phase $3\phi - 2$, while the locking mechanism requires $history_p$ in phase 3ϕ (for unlocking).

7.4 Summary of *BOTR*, *BLV* and *BLK*

Table 1 summarizes the resilience (right column) and the predicate for termination (middle column) of our three algorithms *BOTR*, *BLV* and *BLK*. We can observe that *BOTR* has the weakest predicate for termination, and the strongest condition for resilience. *BLV* and *BLK* have the same resilience, while *BLK* has a slightly stronger predicate for termination (it requires a safe kernel in one more round).

8 Deriving the overall resilience of *BLV*

In this section we look at the overall resilience of the *BLV* consensus algorithm together with the $\mathcal{P}^f_{BLV}$ predicate simulation algorithm. A similar derivation can be done for the *BOTR* and *BLK* algorithms.

When solving consensus in the presence of (only) static value faults ($\mathcal{P}_{\diamond SK} \wedge \mathcal{P}^f_{stat}$), both algorithms (*BLV* and the simulation algorithm) require $n > 3f$. This follows from Theorem 4, Corollary 4 and the fact that $\mathcal{P}^{f,0,3}_{\diamond cons \oplus SK}$ implies $\mathcal{P}^f_{BLV}$. However, these algorithms have different requirements on n in the presence of dynamic value faults ($\mathcal{P}_{\diamond SK} \wedge \mathcal{P}^f_{dyn}$).

From Corollary 5 and the fact that $\mathcal{P}^{f,0,3}_{\diamond cons \oplus SK}$ implies $\mathcal{P}^f_{BLV}$ we get:

Corollary 7 *If* $n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1}$, $n > \alpha + f$, $\alpha \geq f$, *and* $\beta \geq \alpha$, *then Algorithm 2 simulates* $\mathcal{P}^f_{BLV} \wedge \mathcal{P}^\beta_{dyn}$ *from* $\mathcal{P}^{6(f+1)+5}_{\diamond SK} \wedge \mathcal{P}^\alpha_{dyn}$.

From Corollary 7 for any $\beta \geq \alpha$, we can simulate $\mathcal{P}^f_{BLV} \wedge \mathcal{P}^\beta_{dyn}$ from $\mathcal{P}^{f,6(f+1)+5}_{\diamond SK} \wedge \mathcal{P}^\alpha_{dyn}$ if

$$n > \frac{(\beta+1)(\alpha+f)}{\beta-\alpha+1} \quad \wedge \quad n > \alpha + f$$

On the other hand, from Theorem 3 we know that we can solve consensus with BLV under $\mathcal{P}^f_{BLV} \wedge \mathcal{P}^\beta_{dyn}$ if

$$n > 2(\beta + f).$$

Combining these conditions and setting $\beta = k\alpha$, where $k \in \mathbb{R}$, $k \geq 1$, we can solve consensus with Algorithm 4 and Algorithm 2 under $\mathcal{P}^{f,6(f+1)+5}_{\diamond SK} \wedge \mathcal{P}^\alpha_{dyn}$ if the following two conditions hold:

$$n > \frac{(k\alpha+1)(\alpha+f)}{k\alpha-\alpha+1} \tag{2}$$

$$n > 2(k-1)\alpha + 2(\alpha+f). \tag{3}$$

We first consider $\alpha > 1$, then $\alpha = 1$.

Case $\alpha > 1$: We can obtain different resilience bounds depending on the choice of k.

Choosing $k = 1$ leads to the quadratic dependency from Corollary 2, and is thus not what we want to achieve here.

For $k \geq 2$, condition (3) implies condition (2) for any $\alpha > 1$, because $\frac{k\alpha+1}{k\alpha-\alpha+1} \leq 2$. Thus, when choosing $k \geq 2$, the smallest n is obtained with $k = 2$:

$$n > 4\alpha + 2f.$$

In case $1 < k < 2$, the optimal choice of k depends on α and f. As special case we get for $k = 1.5$ from condition (2), $n > \frac{3\alpha+2}{\alpha+2}(\alpha + f)$, i.e.,

$$n > 3(\alpha + f)$$

while from condition (3) we get

$$n > 3\alpha + 2f$$

Since both conditions should hold, it follows that $n > 3(\alpha + f)$.

Case $\alpha = 1$: For the special case $\alpha = 1$ and $f = 1$, conditions (3) and (2) become $n > 2(k-1)+4$ and $n > \frac{2(k+1)}{k}$. We obtain the smallest value for n by choosing $k = 1$, which leads to $n > 4$.

Algorithm	*Communication Predicate for Termination under* $\mathcal{P}_{dyn}^{\alpha}$	*Resilience*
$BOTR$	$\forall \phi, \forall p, \exists \phi_0 > \phi : \ CONS(2\phi_0 - 1) \ \wedge \ \lvert HO(p, 2\phi_0 - 1)\rvert \geq n - f \wedge (\exists \phi_1 \geq \phi_0 : \lvert SHO(p, 2\phi_1)\rvert \geq n - f)$	$n > 4\alpha + 3f$
BLV	$\forall \phi > 0, \exists \phi_0 > \phi : CONS(3\phi_0 - 2) \wedge \forall r \in \{3\phi_0 - 2, \ldots, 3\phi_0\} : \lvert SK(r)\rvert \geq n - f$	$n > 2\alpha + 2f$
BLK	$\forall \phi > 0, \exists \phi_0 > \phi : CONS(3\phi_0 - 2) \ \wedge \forall r \in \{3\phi_0 - 3, \ldots, 3\phi_0\} : \lvert SK(r)\rvert \geq n - f$	$n > 2\alpha + 2f$

Table 1 Summary of the three consensus algorithms $BOTR$, BLV and BLK.

Discussion: The results show that $k = 1$ (i.e. $\beta = \alpha$) leads to the smallest value of n only when $\alpha = 1$. In cases where $\alpha > 1$, a better choice is e.g. $k = 1.5$ (i.e. $\beta = 1.5\alpha$). This is a non intuitive result.

9 Direct implementation of eventual consistency using authentication

In Section 6 we gave two simulations of $\mathcal{P}_{\diamond cons}$ from $\mathcal{P}_{\diamond SK}$. In this section we show that in some systems we can get $\mathcal{P}_{\diamond cons}$ with sufficiently high coverage without such a simulation, but simply using authentication. Authentication has been introduced very early in distributed computing research to solve consensus. Nevertheless, people were always struggling to give a rigorous formal definition of authentication.

The first observation is that in a transmission fault model, the introduction of authentication makes the model in fact benign: if every process signs its messages and upon reception only correctly signed messages are processed, no corruptions can occur. This implies that with authentication (whatever it means) transmission faults are not able to capture Byzantine process faults. However, even if we consider process faults, it is hard to formalize authentication in a precise manner. A possible approach to this open question is, instead of trying define authentication, state what can be achieved with authentication. As we will show, (eventual) consistency is what we naturally get from authentication assuming (eventual) synchrony.

For the clarity of the presentation, we explain how eventual consistency can be achieved using authentication in two steps. In Section 9.1, we show how to obtain $\mathcal{P}_{IC}$ from synchrony and a correct leader using authentication. In Section 9.2, we slightly modify the algorithm of Section 9.1 to obtain $\mathcal{P}_{\diamond cons}$ from eventual synchrony and eventual correct leader.

9.1 Ensuring $\mathcal{P}_{IC}$ from synchrony and correct leader using authentication

Consistency, namely $\mathcal{P}_{IC}^{f}$ (Sect. 5), can be achieved with high probability using cryptographic signatures in a synchronous system with f Byzantine processes (note that we are then no more in the scope of the transmission fault model; for a discussion for the relation between these two models see Section 10). To that end, in every round that should be consistent, every process signs its messages before sending it to the (correct) leader. The leader collects all the messages it receives and forwards them to all processes. The processes deliver all correctly signed messages that are received from the leader as the messages of this round. Technically this procedure requires two "subrounds" that can be obtained in a similar way as the normal round structure. However, the algorithm is not a simulation as in the previous section, since the correctness is *conditional.*

Assuming the (i) signatures cannot be forged, (ii) the system is synchronous and (iii) the leader is correct, it is easy to see that (a) all processes have the same reception vector, and (b) all processes receive at least $n - f$ messages. Therefore, $\mathcal{P}_{IC}^{f}$ holds.

9.2 Ensuring $\mathcal{P}_{\diamond cons}$ from eventual synchrony and eventual correct leader using authentication

The above leader-based procedure can be used, with a small modification, to ensure $\mathcal{P}_{\diamond cons}$ from eventual synchrony. It is sufficient to replace the fixed correct leader with a rotating leader. This ensures an eventual correct leader when synchrony holds. The result follows directly.

10 Communication predicates and corresponding systems

In the HO model, there are no faulty processes and no state corruption. Nevertheless, for predicates that characterize permanent faults, the model can be used to reason about classical Byzantine faults. This implies that the algorithms in this paper can be used also to solve consensus in the classical Byzantine fault model. We develop this observation first for a synchronous system (for simplicity), and then extend it to our model.[12]

[12] This observation was made already in [19] and [4], but without giving algorithms supporting the observation.

Algorithm	*Synchrony*	*Static and Permanent*	*Dynamic and Transient*	*Process faults*	*Link faults*	*Resilience*
[22,18]	synchronous	✓		✓		$n > 3f$
[28]	synchronous	✓			✓	$n > 2l_a + 2$
[31,30]	synchronous	✓		✓	✓	$n > 3f$ and $c > 2f + l_a$
OMH [5]	synchronous	✓	✓	✓	✓	$n > 3f + f_l^{ra} + 2f_l^s + f_l^r + 2f_s + 2f_o + f_m$
FaB Paxos [20]	partially synchronous	✓		✓		$n > 5f$
PBFT [10]	partially synchronous	✓		✓		$n > 3f$
DLS [12]	partially synchronous	✓		✓		$n > 3f$
$\mathcal{A}_{T,E}$ [4]	partially synchronous		✓		✓	$n > 4\alpha$
$\mathcal{U}_{T,E,\alpha}$ [4]	partially synchronous		✓		✓	$n > 2\alpha$
$BOTR$	partially synchronous	✓	✓	✓	✓	$n > 3f + 6\alpha$
BLV	partially synchronous	✓	✓	✓	✓	$n > 3f + 3\alpha$
BLK	partially synchronous	✓	✓	✓	✓	$n > 3f + 3\alpha$

Table 2 Summary of consensus algorithms that tolerate arbitrary faults. The parameter f denotes the number of Byzantine faulty processes; l_a denotes the number of links subjected to arbitrary faults. In [31,30], the parameter c denotes the network connectivity (value c means that there exists at least c disjoint paths between any pairs of processes). In OMH, f_l^{ra} is the number arbitrary receive link failures, f_l^s the number of send link failures, f_l^r the number of receive link failures, f_s the number of symmetrical Byzantine faulty processes, f_o the number of omission faulty processes and f_m the number of manifest faulty processes. The parameter α denotes the maximum number of corrupted messages a process can receive per round.

Let S_f denote a synchronous system with reliable links and at most f Byzantine processes, and consider on the other hand an HO machine with $|SK| \geq n - f$. For correct processes, a run in S_f is indistinguishable from a run of the HO machine. Therefore, an algorithm that solves consensus with $|SK| \geq n-f$ allows in S_f correct processes to solve consensus. Note that in S_f faulty processes do not follow the protocol. It is then natural that they do not follow the specification of consensus.

The same indistinguishability argument can be applied to (i) the weaker partial synchronous system [12] with at most f Byzantine processes and (ii) the HO model with $\mathcal{P}^f_{stat} \wedge \mathcal{P}^{f,\infty}_{\diamond SK}$. For correct processes in the model (i), a run is indistinguishable from a run in model (ii), and so an HO algorithm that solves consensus allows correct processes in the fault-prone system to solve consensus.

The predicate $\mathcal{P}^{\alpha}_{dyn} \wedge \mathcal{P}^{f,k}_{\diamond SK}$, $\alpha \geq f$, can correspond to a partially synchronous system with at most f Byzantine processes, where in addition, before stabilization time, in every round processes can receive $\alpha - f$ corrupted messages from correct processes (*). This spectrum of interpretations, which includes permanent faults (see Sect. 4.3) contrary to [4], shows the benefit of considering the consensus problem in a model with (only) transmission faults.

Further, these interpretations show that the consensus algorithms $BOTR$, BLV and BLK presented in this paper can be used in classical system models. This allows us to compare $BOTR$, BLV and BLK with existing consensus algorithms, specifically consensus algorithms that tolerate arbitrary faults (process and/or link faults). The comparison appears in Table 2. For $BOTR$, BLV and BLK, we assume the interpretation (*) in the preceding paragraph.

11 Conclusion

The transmission fault model allows us to reason about permanent and transient value faults in a uniform way, which makes the model very attractive. However, all existing solutions to consensus in this model are either in the synchronous system, or require strong conditions for termination that exclude the case where all messages of a process can be corrupted. The paper has shown that this limitation can be overcome thanks to the eventual consistency predicate that states the existence of a round where all processes receive the same set of messages. Two simulations of eventual consistency have been given, both from a predicate that corresponds to a partially synchronous system parameterized with α (in every round each process can receive up to α corrupted messages) and f (at most f processes are corrupted). The first simulation, which refers only to the parameter f, is for static faults. The second simulation, which refers to the parameters f and α, includes static and dynamic faults, and is compatible with permanent and transient faults. The paper has pointed out two options for this second simulation: preserving or not the number of corrupted messages in each round. The first option requires $n > (\alpha+1)(\alpha+f)$. The second option requires $n > \eta(\alpha+f)$. Combining the BLV consensus algorithm with this second simulation leads to $n > 3(\alpha + f)$ for $\alpha > 1$ and $n > 4$ for $\alpha = 1$.

References

1. Abraham, I., Chockler, G., Keidar, I., Malkhi, D.: Byzantine disk paxos: optimal resilience with byzantine shared memory. Distributed Computing **18**(5), 387–408 (2006)
2. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Consensus with byzantine failures and little system synchrony. In: Dependable Systems and Networks (DSN 2006), pp. 147–155 (2006)
3. Anagnostou, E., Hadzilacos, V.: Tolerating transient and permanent failures (extended abstract). In: Proceedings of the 7th International Workshop on Distributed Algorithms, WDAG '93, pp. 174–188. Springer-Verlag (1993)
4. Biely, M., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A., Widder, J.: Tolerating corrupted communication. In: Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC'07). ACM Press (2007)
5. Biely, M., Schmid, U., Weiss, B.: Synchronous consensus under hybrid process and link failures. Theor. Comput. Sci. **412**(40), 5602–5630 (2011)
6. Borran, F., Hutle, M., Santos, N., Schiper, A.: Quantitative analysis of consensus algorithms. IEEE Trans. Dependable Sec. Comput. **9**(2), 236–249 (2012)
7. Borran, F., Hutle, M., Schiper, A.: Timing analysis of leader-based and decentralized byzantine consensus algorithms. In: LADC, pp. 166–175 (2011)
8. Borran, F., Schiper, A.: A leader-free byzantine consensus algorithm. In: ICDCN, pp. 67–78 (2010)
9. Brasileiro, F.V., Greve, F., Mostéfaoui, A., Raynal, M.: Consensus in one communication step. In: Proceedings of the 6th International Conference on Parallel Computing Technologies, PaCT '01, pp. 42–50. Springer-Verlag, London, UK (2001)
10. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems **20**(4), 398–461 (2002)
11. Charron-Bost, B., Schiper, A.: The heard-of model: computing in distributed systems with benign faults. Distributed Computing **22**(1), 49–71 (2009)
12. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM **35**(2), 288–323 (1988)
13. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM **32**(2), 374–382 (1985)
14. Gafni, E.: Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In: Proceeding of the 16th Annual ACM Symposium on Principles of Distributed Computing (PODC'98), pp. 143–152. ACM Press, Puerto Vallarta, Mexico (1998)
15. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**, 133–169 (1998)
16. Lamport, L.: Fast paxos. Tech. Rep. MSR-TR-2005-12, Microsoft Research (2005)
17. Lamport, L.: Byzantizing paxos by refinement. In: Proceedings of the 25th international conference on Distributed computing, DISC'11, pp. 211–224. Springer-Verlag, Berlin, Heidelberg (2011)
18. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
19. Lampson, B.: The abcd's of paxos. In: Proceeding of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'01), p. 13. ACM Press (2001)
20. Martin, J.P., Alvisi, L.: Fast byzantine consensus. Transactions on Dependable and Secure Computing **3**(3), 202–214 (2006)
21. Milosevic, Z., Hutle, M., Schiper, A.: Unifying Byzantine consensus algorithms with weak interactive consistency. In: 12th International Conference on Principles of Distributed Systems (OPODIS 2009) (2009)
22. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM **27**(2), 228–234 (1980)
23. Pedone, F., Schiper, A., Urbán, P., Cavin, D.: Solving agreement problems with weak ordering oracles. In: Proceedings of the 4th European Dependable Computing Conference on Dependable Computing, EDCC-4, pp. 44–61. Springer-Verlag, London, UK (2002)
24. Pinter, S.S., Shinahr, I.: Distributed agreement in the presence of communication and process failures. In: Proceedings of the 14th IEEE Convention of Electrical & Electronics Engineers in Israel. IEEE (1985)
25. Rutti, O., Milosevic, Z., Schiper, A.: Generic construction of consensus algorithms for benign and byzantine faults. Dependable Systems and Networks, International Conference on **0**, 343–352 (2010)
26. Santoro, N., Widmayer, P.: Time is not a healer. In: Proc. 6th Annual Symposium on Theor. Aspects of Computer Science (STACS'89), *LNCS*, vol. 349, pp. 304–313. Springer-Verlag, Paderborn, Germany (1989)
27. Santoro, N., Widmayer, P.: Agreement in synchronous networks with ubiquitous faults. Theor. Comput. Sci. **384**(2-3), 232–249 (2007)
28. Sayeed, H.M., Abu-Amara, M., Abu-Amara, H.: Optimal asynchronous agreement and leader election algorithm for complete networks with byzantine faulty links. Distrib. Comput. **9**, 147–156 (1995)
29. Schmid, U., Weiss, B., Rushby, J.: Formally verified byzantine agreement in presence of link faults. In: 22nd International Conference on Distributed Computing Systems (ICDCS'02), pp. 608–616. Vienna, Austria (2002)
30. Siu, H.S., Chin, Y.H., Yang, W.P.: Byzantine agreement in the presence of mixed faults on processors and links. IEEE Trans. Parallel Distrib. Syst. **9**, 335–345 (1998)
31. Yan, K.Q., Chin, Y.H., Wang, S.C.: Optimal agreement protocol in malicious faulty processors and faulty links. IEEE Trans. on Knowl. and Data Eng. **4**(3), 266–280 (1992). DOI 10.1109/69.142017. URL http://dx.doi.org/10.1109/69.142017