On the Core Mechanisms of Consensus Algorithms for Benign and Byzantine Faults (Preliminary Version)

Zarko Milosevic Olivier Rütti André Schiper
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
{*zarko.milosevic,andre.schiper*}*@epfl.ch, olivier.rutti@gmail.com*

Abstract

Consensus is a fundamental and difficult problem in fault tolerant distributed computing, and numerous consensus algorithms have been published. The paper proposes a generic consensus algorithm that highlights, through well chosen parameters, the core mechanisms of a number of well-known consensus algorithms including Paxos, OneThirdRule, PBFT, FaB Paxos. Interestingly, the generic algorithm allowed us to identify a new Byzantine consensus algorithm that requires $n > 4b$, inbetween the requirement $n > 5b$ of FaB Paxos and $n > 3b$ of PBFT (b is the maximum number of Byzantine processes). The paper contributes to identify key similarities rather than non fundamental differences between consensus algorithms.

1 Introduction

Consensus is a fundamental and difficult problem in fault tolerant distributed computing. This explains the numerous consensus algorithms that have been published, with different features and for different fault models. Understanding these numerous algorithms could be made easier by identifying the core mechanisms on which these algorithms rely. This is the purpose of the paper. Identifying these mechanisms allowed us also to derive a generic consensus algorithm, which highlights, through well chosen parameters, the core mechanisms of a number of well-known consensus algorithms including Paxos [11], OneThirdRule [5], PBFT [3] and FaB Paxos [17].

Our generic consensus algorithm assumes a partially synchronous system model [6], the system model assumed by the two most popular consensus algorithms, Paxos [11] (for benign faults) and PBFT [3] (for Byzantine faults). However, in order to improve the clarity of the algorithms and simplify the proofs, as in [6], we consider an abstraction on top of the system model, namely the round model. Expressing a consensus algorithm in the round model or directly in the partially synchronous system, does not change its core mechanisms. The generic algorithm consists of successive phases, where each phase is composed of three rounds: a *selection* round, a *validation* round and a *decision* round. The validation round may be skipped by some algorithms, which introduces a first dichotomy among consensus algorithms: those that require the validation round, and the others for which the validation round is not necessary. We further subdivide the former algorithms in two, based on the state variables required. This lead us to identify three classes of consensus algorithms: OneThirdRule and FaB Paxos belong to class 1, Paxos to class 2 and PBFT to class 3. Interestingly, this classification allowed us to discover a new Byzantine consensus algorithm that requires $n > 4b$ (inbetween the requirement $n > 5b$ of FaB Paxos and $n > 3b$ of PBFT).[1]

Our generic algorithm is based on four parameters: FLV function, $Validator$ function, threshold parameter T_D, and $FLAG$ ($FLAG = *$ or $FLAG = \phi$). The functions FLV and $Validator$ are characterized by abstract properties; T_D is defined with respect to n (number of processes), f (maximum number of benign faults) and b (maximum number of Byzantine processes). We prove correctness of the generic consensus algorithm by referring only to the abstract properties of our parameters. The correctness proof of any specific instantiated consensus algorithm consists simply in proving that the instantiations satisfy the abstract properties of FLV and $Validator$.

The paper is not the first one to propose a generic consensus algorithm, but it goes significantly beyond previous approaches. Mostéfaoui *et al.* [19] propose a consensus framework restricted to benign faults, which allows unification of leader oracle, random oracle and failure detector oracle. Guerraoui and Raynal [9] propose a generic consensus

[1] b is the maximum number of Byzantine processes.

algorithm, where generality is encapsulated in a function called $Lambda$. The $Lambda$ function encapsulates our selection and our validation rounds. This does not allow the paper to identify the differences between two of our three classes of consensus algorithms. Moreover, as for [19], the paper is restricted to benign faults. Later, Guerrraoui and Raynal [10] proposed a generic version of Paxos in which communication (using shared memory, storage area networks, message passing channels or active disks) is encapsulated in the *Omega* abstraction. The paper is also restricted to benign faults. Apart from this work, several other authors proposed abstractions related to Paxos-like protocols, e.g., [14, 15, 12]. More generally, Song et al. [21] proposed building blocks that allow the construction of consensus algorithms. The paper considers both benign and Byzantine faults. However, it ignores some seminal consensus algorithms such as PBFT and FaB Paxos, and therefore has a somehow limited scope.

The rest of the paper is organized as follows. Section 2 defines the consensus problem. Section 3 introduces the system model. We derive our generic consensus algorithm and prove its correctness in Section 4. Section 5 establishes minimality results related to T_D. In Section 6 we present three instantiations of the FLV function that lead to the three families of consensus algorithms. Section 7 gives examples of instantiations and Section 8 concludes the paper.

2 Consensus problem

The consensus problem is defined over a set of processes Π, where each process $p \in \Pi$ starts with a given initial value, and later *decides* on a common value. We differentiate *honest* processes that execute algorithms faithfully, from *Byzantine* processes [13], that exhibit arbitrary behavior. Honest processes can be *correct* or *faulty*. An honest process is faulty if it eventually crashes, and is correct otherwise. Among the n processes in our system, we assume at most b Byzantine processes and at most f faulty (honest) processes. The set of honest processes is denoted by $\mathcal{H}$ and the set of correct processes by $\mathcal{C}$.

The consensus problem is formally specified by the following properties:

• *Agreement:* No two honest processes decide differently;
• *Termination:* All correct processes eventually decide;
• *Validity:* If all processes are honest and if an honest process decides v, then v is the initial value of some process;
• *Unanimity* [21]: If all honest processes have the same initial value v and an honest process decides, then it decides v. Unanimity (which extends validity) is optional, and only makes sense with Byzantine processes

It is useful to classify properties of a system into two categories: *safety* properties and *liveness* properties. Roughly speaking, a safety property stipulates that "nothing bad" will ever happen; a liveness property stipulates that "something good" will eventually happen. More precisely, a safety property is a property whose violation can be observed by looking only at the prefix of an execution (i.e., by looking only at an execution up to a certain time). This is not the case for a liveness property. Agreement, validity and unanimity are *safety* properties, while termination is a *liveness* property.

3 System model

When solving consensus, one important parameter is the degree of *synchrony* of the system. The two main models considered in distributed computing are: the *synchronous* system model and the *asynchronous* system model. In a synchronous system model there is (1) a known bound Δ on the transmission delay of messages, and (2) a known bound Φ on the relative speed of processes. On the other hand, in an asynchronous system there is no bound on the transmission delay of messages and no bound on the relative speed of processes. This typically models a system with unpredictable load on the network and on the CPU.

Unfortunately, consensus is impossible to solve with a deterministic algorithm[2] in an asynchronous system even if only single process may crash [7]. Although it is possible to solve consensus in the synchronous system model with Byzantine processes, it is not considered as a good idea from a practical point of view. The reason is that the synchronous system model requires to be pessimistic when defining the bounds on message transmission delays (and process relative speeds). Pessimistic bounds have negative impact on the performance of consensus algorithms.

Therefore, Lynch, Dwork and Stockmayer proposed the partially synchronous system model [6] that lies between a synchronous system and an asynchronous system. Roughly speaking, a partially synchronous system is initially asynchronous and eventually becomes synchronous. The partially synchronous system model distinguishes partial synchrony for processes and partial synchrony for communication. It is possible to solve consensus in the partially synchronous system model in the presence of Byzantine faulty processes, and contrary to the synchronous system model, the partially synchronous system model does not require being too pessimistic when defining the bounds on message transmission delays and process relative speeds [3]. Unless stated otherwise, in the rest of the paper we consider the partially synchronous system model. More precisely, we consider a variant of a partially synchronous system model

[2]A deterministic algorithm is an algorithm that does not use randomization (random number generation).

[3]In one variant of the partially synchronous system model, upper bounds on message delays and process relative speeds exist, but depend on the run.

where we assume that the system alternates between good periods (during which the system is synchronous) and bad periods (during which the system is asynchronous).

3.1 Basic Round Model

As in [6], we consider an abstraction on top of the system model, namely a *basic round model*. Using this abstraction rather than the raw system model improves the clarity of the algorithms and simplifies the proofs. In the basic round model, distributed algorithms are expressed as a sequence of rounds. Each round r consists of a sending step, a receive step, and a state transition step:

1. In the sending step of round r, each process p sends a message to each process according to a "sending" function S_p^r.[4]

2. In the receive step of round r, each process q receives subset of all messages sent (it can be the empty set) in round r; messages received by process p in round r are denoted by $\vec{\mu}_p^r$ ($\vec{\mu}_p^r[q]$ is the message received from q). The receive step is implicit, i.e., it does not appear in the algorithm.

3. In the state transition step of round r (that takes place at the end of round r), each process p computes a new state according to a "transition" function T_p^r that takes as input the vector of messages it received at round r and its current state.

Note that this implies that a message sent in round r can only be received in round r (rounds are *closed*).

In every round of the basic round model, if an honest process sends v, then every honest process receives v or nothing. This can formally be expressed by the following predicate ($\perp$ represents no message reception, int stands for integrity):

$$\mathcal{P}_{int}(r) \equiv \forall p, q \in \mathcal{H} : (\vec{\mu}_p^r[q] = S_q^r(s_q^r)\,) \;\vee\; (\vec{\mu}_p^r[q] = \perp)$$

The state of process p in round r is denoted by s_p^r; the message sent by an honest[5] process is denoted by $S_p^r(s_p^r)$. We will refer to some field fld of a message m using $m.fld$ notation.

3.2 Characterizing a good period

During a bad period, except $\mathcal{P}_{int}$, no guarantees on the messages a process receives can be provided: it can even happen that no messages at all are received. During a good period it is possible to ensure, for all rounds r in the good period, that all messages sent in round r by a correct process are received in round r by all correct processes. This is formally expressed by the following predicate:

$$\mathcal{P}_{good}(r) \equiv \forall p, q \in \mathcal{C} :\; \vec{\mu}_p^r[q] = S_q^r(s_q^r)$$

The reader can find in [6] the implementation of rounds that satisfy $\mathcal{P}_{good}$ during a good period in the presence of Byzantine processes.

During good periods of our partially synchronous system, we can ensure instead of $\mathcal{P}_{good}$, the stronger predicate $\mathcal{P}_{cons}$ ($cons$ stands for consistency). The predicate $\mathcal{P}_{cons}$ additionally ensures that each correct process receives the same set of messages:

$$\mathcal{P}_{cons}(r) \equiv \mathcal{P}_{good}(r) \;\wedge\; \forall p, q \in \mathcal{C} :\; \vec{\mu}_p^r = \vec{\mu}_q^r$$

In the benign fault model (i.e., $b = 0$), this predicate can be implemented using the implementation of $\mathcal{P}_{good}$ described in [6] if we assume that no crash occurs in good periods. In the Byzantine fault model (i.e., $b \neq 0$), several implementations of $\mathcal{P}_{cons}$ have been proposed [18], for Byzantine faults and authenticated Byzantine faults. The two (coordinator-based) implementations have different costs: the latter requires two micro-rounds, the former three micro-rounds. Interestingly, there is also a decentralized (i.e., coordinator-free) implementation of $\mathcal{P}_{cons}$ for the Byzantine fault model that requires $b+1$ micro-rounds [1].

A *phase* is a sequence of rounds. We define a *good phase* of k rounds as a phase such that $\mathcal{P}_{cons}$ holds in the first round, and $\mathcal{P}_{good}$ holds in the remaining $k-1$ rounds.

4 Deriving a generic consensus algorithm

The goal of this section is understanding what are the core mechanisms (sometimes also called "building blocks" [21]) present in existing consensus algorithms. Identifying these mechanisms and properties they provide will allow us to derive a generic consensus algorithm.

4.1 Very simple consensus algorithm

We start our quest for a generic consensus algorithm with a very simple consensus algorithm (code shown as Algorithm 1). Algorithm 1 consists of a single round in which each process collects initial values from all processes, then applies a deterministic function to choose some value v (line 5) and then decides on v. The notation $\#(v)$ is used to denote the number of messages received with value v, *i.e.*, $\#(v) \equiv \left|\left\{q \in \Pi \;:\; \vec{\mu}_p^r[q] = v\right\}\right|$.

Theorem 1. *Algorithm 1 solves consensus if* $n > 2b + f$, *and if in addition* $\mathcal{P}_{cons}(1)$ *holds.*

[4] Without loss of generality, the same message is sent to all.

[5] Note that referring to the state of a Byzantine process does not make sense.

Algorithm 1 Very simple consensus algorithm

```
1: Round r = 1:
2:   S_p^r:
3:     send ⟨init_p⟩ to all
4:   T_p^r:
5:     v ← min {v : ∄v' ∈ V s.t. #(v') > #(v)}
6:     DECIDE v
```

Proof. Termination and Validity trivially holds. Agreement holds from $\mathcal{P}_{cons}(1)$ and the fact that all processes choose the decision values using the deterministic *min* function at line 5. We now prove that Unanimity also holds.

We assume that initial value of all honest processes is v. The proof is by contradiction. We assume by contradiction that there is an honest process p that decides $v' \neq v$. Therefore, v' is the smallest most frequent value received by p in round 1 at line 5. By $\mathcal{P}_{cons}(1)$, process p receives at least $n - b - f$ messages equal to v in round 1. Furthermore, since there are at most b Byzantine processes, p received at most b messages equal to v'. Since $n > 2b + f$, p received more than b messages equal to v, and at most b messages equal to v'. Therefore, the value selected at line 5 is v and p decided v. A contradiction. □

4.2 Generic Algorithm: Draft 1

Algorithm 1 is not correct in the partially synchronous system model because $\mathcal{P}_{cons}()$ cannot be ensured from the beginning. To remedy this, a consensus protocol has to invoke multiple instances of a sub-protocol. Such sub-protocols have been called rounds, phases, views or ballots. In this paper, we use the term *phase*, where a phase itself consists of *rounds*. Using this terminology, we can say that Algorithm 1 consists of a single phase with a single round. In the partially synchronous system model, algorithms consist of a sequence of phases, where each phase contains some fixed number of rounds.

Having multiple phases requires additional mechanisms compared to Algorithm 1:

(i) A mechanism to detect that a decision is possible in a given phase. Clearly, this mechanism must ensure that two honest processes that decides in a given phase, decide the same value.

(ii) A mechanism to ensure consistency among decisions made by honest processes in different phases.

(iii) A mechanism to ensure that all correct processes eventually decide.

Concerning (i), we introduce the notion of *decision quorum* captured by parameter T_D. The parameter T_D defines the number of identical votes required to decide. More precisely, once a process p observes that T_D processes (decision quorum) have voted for value v, then it can decide on v. There are some obvious restrictions on the value of T_D:

- To ensure unanimity, $T_D > b$, i.e., a decision quorum must contain at least one honest process.

- To ensure termination, the votes of faulty (honest) and Byzantine processes must not be required to decide. Hence, $T_D \leq n - b - f$.

Concerning (ii), we introduce the notion of *locked value*[6] and the function $FLV(\vec{\mu}_p^r)$ (stands for "Find the Locked Value") used to retrieve locked value (if there is some) in a set of messages received.[7] A value v is locked in round r if:

1. An honest process has decided v in round $r' < r$, or
2. All honest processes have the same initial value v.

Item 2 is meaningful only if unanimity has to be ensured, or if all processes are honest. In all other cases, item 2 can be ignored. From this definition it follows that, if v is locked in the context of a consensus algorithm then the configuration is v-valent. However, the opposite is not true (e.g., if a configuration is v-valent in round r, and the first honest process p decides v in round $r' \geq r$, then v is not locked in round r, but only in round $r' + 1 > r$).

The basic idea for ensuring agreement among different phases is the following. If some value v is locked in round r, then any honest process p that updates its variable $vote_p$[8] in round r, can only, thanks to the function $FLV(\vec{\mu}_p^r)$, update it to v. In addition to normal values, the function FLV may return the following special values:

- $?$ if no value is locked, i.e., any value can be assigned to $vote_p$

- $null$ if no enough information is provided to FLV through $\vec{\mu}_p^r$

The $FLV(\vec{\mu}_p^r)$ function is defined by the following three properties:

- *FLV-validity*: If all processes are honest and $FLV(\vec{\mu}_p^r)$ returns v such that $v \neq ?$ and $v \neq null$, then $\exists$ process q such that $v = \vec{\mu}_p^r[q].vote$.
- *FLV-agreement*: If value v is locked in round r, only v or $null$ can be returned.
- *FLV-liveness*: If $\forall q \in \mathcal{C} : \vec{\mu}_p^r[q] \neq \perp$, then $null$ cannot be returned.

[6] The definition of *locked value* in some other works differs from our definition.

[7] FLV is not really a function. It is rather a problem defined by properties. However, calling it a function is more intuitive.

[8] Variable $vote_p$ is p's estimate of the decision value.

FLV-validity and *FLV-agreement* are for safety, while *FLV-liveness* is for liveness. Note that, as FLV is used to find the locked value, its instantiations depend on the T_D parameter (since T_D defines when a value becomes locked). We discuss the relations between T_D and FLV instantiations in Section 5.

Starting from Algorithm 1 and using parameters T_D and $FLV(\vec{\mu}_p^r)$, we obtain a first draft of our generic consensus algorithm, see Algorithm 2. Algorithm 2 consists of a sequence of *phases* that can be seen as successive trials to decide on a value. Each phase ϕ consists of two rounds, respectively called *selection round* ($r = 2\phi - 1$) and *decision round* ($r = 2\phi$).

Algorithm 2 Generic Algorithm – Draft 1 (boxes represent parameters)

Initialization:
  $vote_p := init_p$ /* value considered for consensus */

Selection Round $r = 2\phi - 1$**:**
  S_p^r:
    send $\langle vote_p \rangle$ to all
  T_p^r:
    $select_p \leftarrow \boxed{FLV(\vec{\mu}_p^r)}$
    if $select_p = \ ?$ **then**
      $select_p \leftarrow$ choose deterministically a value among the votes received
    if $select_p \neq null$ **then**
      $vote_p \leftarrow select_p$

Decision Round $r = 2\phi$**:**
  S_p^r:
    send $\langle vote_p \rangle$ to all
  T_p^r:
    if received at least $\boxed{T_D}$ messages with the same vote $\langle v \rangle$ **then**
      DECIDE v

The selection round ($r = 2\phi - 1$) selects a value that will be considered for the decision. Each process p first sends its state ($vote_p$) to all processes. Based on the set of messages received, each honest process selects a value. If any value can be selected (i.e., $FLV(\vec{\mu}_p^r)$ returns ?), the selected value is deterministically chosen among $\vec{\mu}_p^r$. If $FLV(\vec{\mu}_p^r)$ returns neither ? nor $null$, then the returned value is selected. If $FLV(\vec{\mu}_p^r)$ returns $null$, then $vote_p$ is not updated.

The decision round ($r = 2\phi$) determines the conditions that must hold for a process to decide. Each process starts by sending its vote to all processes. A process then decides if it receives a threshold number T_D of identical votes. To ensure agreement we require $T_D > \frac{n+b}{2}$, so that two honest processes that decide in the same phase decide the same value.

In order to guarantee that all correct processes eventually decide we rely on the notion of *good phase* (Sect. 3.2). A good phase ensures that all correct processes receive the same set of messages in the selection round, and therefore select the same value.[9] Since all correct processes update $vote$ to the same value, they all decide in the decision round of the good phase.

[9] FLV-liveness ensures that the selected value cannot be null.

4.3 Generic Algorithm: Draft 2

With Draft 2 we manage to reduce T_D. Remember that $T_D \leq n - b - f$, i.e., $n \geq T_D + b + f$, which means that a smaller T_D leads to a smaller n.

In the decision round of Draft 1 (Algorithm 2), the votes sent by honest processes can be different. Therefore, to prevent two honest processes from deciding different values in the same phase, we must have $T_D > \frac{n+b}{2}$. This condition can be relaxed if honest processes would vote for at most one value in a phase—the *validated vote*. In this case, $T_D > b$ is enough, since it ensures that the decision quorum contains at least one honest process.

To ensure that honest processes vote for at most one value in a phase, we need to add one more round to Algorithm 2 — the *validation round* — and to introduce the timestamp variable ts_p. For every process p, the timestamp ts_p represents the most recent phase in which the vote of process p ($vote_p$) has been *validated* in the validation round. The second draft of our generic algorithm is presented as Algorithm 3.

The validation round is executed as follows. Each process p first sends the value selected in the selection round ($select_p$), if non-$null$. Based on the set of messages received, each process tries to determine a validated value v. If it observes that a majority of honest processes (i.e., more than $\frac{n+b}{2}$) have selected the same value v, then $vote_p$ is set to v and ts_p is updated to the current phase number ϕ. This mechanism ensures that all honest processes that validate some value v in phase ϕ, consider the same value.

Introducing the validation round and the timestamp variable requires changes in the decision and the selection rounds. In the decision round we need to distinguish two cases: (i) honest processes vote for at most one value (thanks to the validation round), and (ii) honest processes can vote for different values (no validation round). We introduce the parameter $FLAG$ to distinguish between these two cases. In case (i) $FLAG$ is an integer (the current phase number); in case (ii) $FLAG$ is the special wildcard value $*$. In line 29, if $FLAG = *$ then all votes are taken into account, and the validation round is not needed. Otherwise, $FLAG = \phi$ (current phase number), and only the votes with $ts_p = \phi$ are taken into account.

The $vote_p$ and ts_p variables are not only used within one phase, but also between phases, in order to ensure that if one honest process decides v in phase ϕ, honest processes select v in the selection round of phase $\phi + 1$. In the context of Byzantine faults, we need a mechanism to prove that some value v may have been validated in some previous phase (to filter out invalid votes sent by Byzantine pro-

Algorithm 3 Generic Algorithm – Draft 2 (boxes represent parameters)

1: **Initialization:**
2: $vote_p := init_p \in V$ /* value considered for consensus */
3: $ts_p := 0$ /* last phase in which $vote_p$ has been validated */
4: $history_p := \{(init_p, 0)\}$ /* updates to the variable $vote_p$ */

5: **Selection Round** $r = 3\phi - 2$**:**
6: S_p^r:
7: send $\langle vote_p, ts_p, history_p \rangle$ to all
8: T_p^r:
9: $select_p \leftarrow \boxed{FLV(\vec{\mu}_p^r)}$
10: **if** $select_p = ?$ **then**
11: $select_p \leftarrow$ choose deterministically a value among the votes received
12: **if** $select_p \neq null$ **then**
13: $vote_p \leftarrow select_p$
14: $history_p \leftarrow history_p \cup \{(vote_p, \phi)\}$

15: **Validation Round** $r = 3\phi - 1$**:** /* executed only if $FLAG = \phi$ */
16: S_p^r:
17: **if** $select_p \neq null$ **then**
18: send $\langle vote_p \rangle$ to all
19: T_p^r:
20: **if** there is a value v such that $\left| \left\{ q \in \Pi : \vec{\mu}_p^r[q] = \langle v \rangle \right\} \right| > \frac{n+b}{2}$ **then**
21: $vote_p \leftarrow v$
22: $ts_p \leftarrow \phi$
23: **else**
24: $vote_p \leftarrow v$ such that $(v, ts_p) \in history_p$ /* revert the value of $vote_p$ to ensure consistency with ts_p */

25: **Decision Round** $r = 3\phi$**:**
26: S_p^r:
27: send $\langle vote_p, ts_p \rangle$ to all
28: T_p^r:
29: **if** received at least $\boxed{T_D}$ messages with the same value $\langle v, \boxed{FLAG} \rangle$ **then**
30: DECIDE v

cesses). The mechanism is based on an additional variable $history_p$, which is a list of pairs (v, ϕ): each pair denotes that $vote$ has been set to v in the selection round of phase ϕ, i.e., that value v may have been validated in phase ϕ.[10] Variable $history$ is sent together with $vote$ and ts in the selection round, where it is used by the FLV function: a pair $(vote, ts)$ is considered valid if at least $b + 1$ processes sent it in their $history$ variable.[11] In the context of (only) benign faults, variable $history$ can be ignored.

4.4 Generic Algorithm: final version

In Algorithm 3 in every round all processes send messages to all processes. This can be avoided by introducing the notion of *validator*. Validators are processes that have a special role in the validation round. The intuition is the following: instead of all processes being involved in the selection of a validated vote, this task is devoted to a subset of processes called validators. The question is how to select the validators. We express this through the function $Validator(p, \phi)$[12] that returns a set of processes $S \subseteq \Pi$ representing the validators for process p in phase ϕ. Note that in the benign fault model, $Validator(p, \phi)$ can be one single process, namely the *leader* process or the process selected by the *rotating coordinator* paradigm.

$Validator(p, \phi)$ is defined by the following three properties:

- *Validator-validity*:
  If $|Validator(p, \phi)| > 0$ then $|Validator(p, \phi)| > b$.

- *Validator-agreement*:
  $\forall p, q \in \mathcal{H}$ and $\forall \phi$, if $|Validator(p, \phi)| > 0$ and $|Validator(q, \phi)| > 0$, then $Validator(p, \phi) = Validator(q, \phi)$.

- *Validator-liveness*:
  There exists a good phase ϕ_0 such that:
  $\forall p \in \mathcal{C} : |Validator(p, \phi_0) \cap \mathcal{C}| > \frac{|Validator(p, \phi_0)| + b}{2}$.

Introducing $Validator(p, \phi)$ leads to Algorithm 4. In the validation round, only validators send messages. Line 20 matches line 20 of Algorithm 3. Specifically, expression $\frac{|validators_p| + b}{2}$ of Algorithm 4 (majority of honest processes among validators) matches expression $\frac{n+b}{2}$ of Algorithm 4 (majority of honest processes among all processes). If p observes that a majority of honest validators have selected the same value v, then v is a validated value, and p sets $vote_p$ to v, and updates its timestamp ts_p to ϕ. Otherwise, the vote is reverted to the value corresponding to ts_p (line 24).[13]

4.5 Correctness of the Generic Algorithm

We now prove that the generic algorithm (Algorithm 4) solves consensus. Our proof is based on two lemmas.

Lemma 1. *If Validator-validity holds, then the following property holds on every honest process h and in every phase ϕ: if process h set $vote_h$ to v and ts_h to ϕ at lines 21-22, then at least one honest process has sent $\langle v \rangle$ at line 18.*

Proof. Assume that a process h set $vote_h$ to v and ts_h to ϕ at lines 21-22. Therefore, $Validator(h, \phi)$ is non empty at line 20 in phase ϕ. By $Validator$-validity, we have $|Validator(h, \phi)| > b$ and $\frac{|Validator(h,\phi)| + b}{2} > b$. Therefore, condition at line 20 can only be true for v if an honest process has sent $\langle v \rangle$ at line 18. □

[10] The size of variable $history_p$ is unbounded. Bounding the size of the variable $history_p$ requires an additional round of communication [2].

[11] Another solution is to rely on authentication, i.e., to consider *authenticated Byzantine fault model*.

[12] As for FLV, *Validator* is not really a function. It is rather a problem defined by properties. However, calling it a function is somehow more intuitive.

[13] Line 24 is not mandatory, but it allows us to simplify the instantiation of function $FLV(\vec{\mu}_p^r)$.

Algorithm 4 Generic Algorithm (boxes represent parameters)

1: **Initialization:**
2: $vote_p := init_p$ /* value considered for consensus */
3: $ts_p := 0$ /* last phase in which $vote_p$ has been validated */
4: $history_p := \{(init_p, 0)\}$ /* updates to the variable $vote_p$ */

5: **Selection Round** $r = 3\phi - 2$**:** /* round in which $\mathcal{P}_{cons}$ must eventually hold */
6: S_p^r:
7: send $\langle vote_p, ts_p, history_p \rangle$ to all
8: T_p^r:
9: $select_p \leftarrow \boxed{FLV(\vec{\mu}_p^r)}$
10: **if** $select_p = \ ?$ **then**
11: $select_p \leftarrow$ choose deterministically a value among the votes received
12: **if** $select_p \neq null$ **then**
13: $vote_p \leftarrow select_p$
14: $history_p \leftarrow history_p \cup \{(vote_p, \phi)\}$

15: **Validation Round** $r = 3\phi - 1$**:** /* executed only if $FLAG = \phi$; round in which $\mathcal{P}_{good}$ must eventually hold */
16: S_p^r:
17: **if** $p \in \boxed{Validator(p, \phi)}$ and $select_p \neq null$ **then**
18: send $\langle select_p \rangle$ to all
19: T_p^r:
20: **if** there is a value v such that $\left| \left\{ q \in \boxed{Validator(p,\phi)} : \vec{\mu}_p^r[q] = \langle v \rangle \right\} \right| > \frac{|\boxed{Validator(p,\phi)}| + b}{2}$ **then**
21: $vote_p \leftarrow v$
22: $ts_p \leftarrow \phi$
23: **else**
24: $vote_p \leftarrow v$ such that $(v, ts_p) \in history_p$ /* revert the value of $vote_p$ to ensure consistency with ts_p */

25: **Decision Round** $r = 3\phi$**:** /* round in which $\mathcal{P}_{good}$ must eventually hold */
26: S_p^r:
27: send $\langle vote_p, ts_p \rangle$ to all
28: T_p^r:
29: **if** received at least $\boxed{T_D}$ messages with the same value $\langle v, \boxed{FLAG} \rangle$ **then**
30: DECIDE v

Lemma 2. *In every phase ϕ, if (i) $Validator$-validity and $Validator$-agreement hold, (ii) an honest process p updates $vote_p$ to v and ts_p to ϕ, and (iii) another honest process q updates $vote_q$ to v' and ts_q to ϕ (lines 21-22), then $v = v'$.*

Proof. Assume for a contradiction that in some phase ϕ_0 two honest processes p and q have respectively $vote_p = v$ and $ts_p = \phi_0$, and $vote_q = v'$ and $ts_q = \phi_0$. This means that in round $3\phi_0 - 1$ at least $x - b$ honest processes ($x > \frac{|Validator(p,\phi_0)|+b}{2}$) send message $\langle v, - \rangle$ and at least $y - b$ honest processes ($y > \frac{|Validator(q,\phi_0)|+b}{2}$) send message $\langle v', - \rangle$. By $Validator$-agreement we have that $Validator(p, \phi_0) = Validator(q, \phi_0)$. Therefore, $x - b + y - b > |Validator(p, \phi_0)| - b$. By $Validator$-validity, it follows that at least one honest process h sent $\langle v, - \rangle$ to one process and $\langle v', - \rangle$ to another process. A contradiction with the fact that h is an honest process.

□

Theorem 2. *If (i) function $FLV(\vec{\mu}_p^r)$ satisfies FLV-validity and FLV-agreement, (ii) function $Validator(p, \phi)$ satisfies $Validator$-validity and $Validator$-agreement, (iii-a) $FLAG = \phi$ and $T_D > b$ or (iii-b) $FLAG = *$ and $T_D > \frac{n+b}{2}$, then Algorithm 4 ensures validity, unanimity and agreement.*

Termination holds if (iv) $T_D \leq n - b - f$, (v) function $FLV(\vec{\mu}_p^r)$ satisfies FLV-liveness, and (vi) there is a good phase ϕ_0 in which $Validator$-liveness holds.

Proof:

(a) Agreement: Assume for a contradiction that process p decides v in round $r = 3\phi$, and process p' decides $v' \neq v$ in round $r' = 3\phi'$. We consider the following two cases for line 29: $FLAG = *$ and $FLAG = \phi$.

$FLAG = *$: This means that at least T_D processes (at least $T_D - b$ honest) sent $\langle v, \rangle$ in round $r = 3\phi$, and at least T_D processes (at least $T_D - b$ honest) sent $\langle v', \rangle$ in round $r' = 3\phi'$ (*). We have two cases to consider: $\phi = \phi'$ and $\phi > \phi'$.

- $\phi = \phi'$: By (*), $T_D - b$ honest processes sent $\langle v, \rangle$ and $T_D - b$ honest processes sent $\langle v', \rangle$ in round $r = 3\phi$. From (iii-b), $(T_D - b) + (T_D - b) > n - b$. It follows that one honest process h sent $\langle v, \rangle$ to one process and $\langle v', \rangle$ to another process. A contradiction with the fact that h is an honest process.
- $\phi' > \phi$: Let ϕ' be the smallest phase $> \phi$ in which

some honest process decides $v' \neq v$. By definition of a locked value, v is locked in all phases $> \phi$. Together with the FLV-agreement property, no honest process updates its vote with a value $\hat{v} \neq v$ in the selection round of a phase $> \phi$. Since there is no validation round (i.e., $FLAG = *$), no honest process updates its vote with a value $\hat{v} \neq v$ after phase ϕ. Together with (*), $T_D - b$ honest processes sent $\langle v, \rangle$, and $T_D - b$ honest processes sent $\langle v', \rangle$ in round $r' = 3\phi'$. From (iii-b), $(T_D - b) + (T_D - b) > n - b$. It follows that one honest process h sent $\langle v, \rangle$ to one process and $\langle v', \rangle$ to another process. A contradiction with the fact that h is an honest process.

$FLAG = \phi$: This means that at least T_D processes (at least $T_D - b$ honest) sent $\langle v, \phi \rangle$ in round $r = 3\phi$, and at least T_D processes (at least $T_D - b$ honest) sent $\langle v', \phi' \rangle$ in round $r' = 3\phi'$ (**). We have two cases to consider: $\phi = \phi'$ and $\phi > \phi'$.

- $\phi = \phi'$: By (**), $T_D - b$ honest processes sent $\langle v, \phi \rangle$ and $T_D - b$ honest processes sent $\langle v', \phi \rangle$. From (iii-a), there is an honest process h that validates v (set $vote_h$ to v and ts_h to ϕ) and an honest process h' that validates v' (set $vote_{h'}$ to v' and $ts_{h'}$ to ϕ) at lines 21-22 of round $\hat{r} = 3\phi - 1$. A contradiction with Lemma 2 and (ii).
- $\phi' > \phi$: Let ϕ' be the smallest phase $> \phi$ in which some honest process decides $v' \neq v$. By definition of a locked value, v is locked in all phases $> \phi$. By (**), $T_D - b$ honest processes sent $\langle v', \phi \rangle$. From (iii-a), there is an honest process h that validates v' (set $vote_{h'}$ to v' and $ts_{h'}$ to ϕ) at lines 21-22 of round $\hat{r}' = 3\phi' - 1$. By (ii) and Lemma 1, there is an honest process h' that sent $\langle v', - \rangle$ at line 18. Therefore, the function $FLV(\vec{\mu}_p^r)$ returns v' or ? at line 9 on process h'. A contradiction with the FLV-agreement property and the fact that v is locked.

(b) Validity: Follows from the FLV-validity property and the assumption that all processes are honest.

(c) Unanimity: Unanimity follows from Lemma 1 together with (ii), the FLV-agreement property, (iii-a) and (iii-b).

(d) Termination: Let ϕ_0 be the good phase in which $Validator$-liveness holds. By $\mathcal{P}_{cons}(3\phi_0 - 2)$, all correct processes receive the same set of messages in round $3\phi_0 - 2$ and therefore select the same value. By FLV-liveness and $\mathcal{P}_{cons}(3\phi_0 - 2)$, the value selected cannot be $null$. Thus, at the end of round $r = 3\phi_0 - 2$, all correct processes have the same value for $select_p$, and $vote_p$ (***). Let denote this value with v. We have two cases to consider: $FLAG = *$ and $FLAG = \phi$.

$FLAG = *$: The validation $3\phi_0 - 1$ round is skipped. Together with (***), all correct processes send the same message $\langle v, - \rangle$ at line 27. By $\mathcal{P}_{good}(3\phi_0)$ and (iv), all correct processes receives at least T_D messages $\langle v, - \rangle$ in round $r = 3\phi_0$, and therefore decide.

$FLAG = \phi$: Let us call *validator* any process that is in a set $Validator(p, \phi_0)$ where p is a correct process. By $Validator$-liveness and $Validator$-agreement, all correct processes p consider the same set $Validator(p, \phi_0)$ at line 20. By $Validator$-liveness, the set $Validator(p, \phi_0)$ contains more than $\frac{|Validator(p,\phi_0)|+b}{2}$ correct processes (****). Since $FLAG = \phi$, the validation round $3\phi_0 - 1$ is executed. Together with (****), $\mathcal{P}_{good}(3\phi_0 - 1)$ and lines 20-22, all correct processes update $vote_p$ to v and ts_p to ϕ_0. By $\mathcal{P}_{good}(3\phi_0)$ and (iv), all correct processes receives at least T_D messages $\langle v, \phi_0 \rangle$ in round $r = 3\phi_0$, and therefore decide. □

4.6 Optimizations

We point out here several simple optimizations of Algorithm 4, to which we will refer later when discussing instantiations of our generic algorithm.

(i) If $FLAG = \phi$, processes in the selection round can send their message only to the processes in the $Validator$ set (instead to all processes).

(ii) The selection round can be suppressed in the first phase. As a consequence, if $FLAG = *$ then a decision is possible in one round if all correct processes have the same initial value and $\mathcal{P}_{good}$ holds in the first round. If $FLAG = \phi$, suppressing the selection round in the first phase requires to initialize the variable $selected_p$ to $init_p$.[14]

(iii) If Unanimity is not considered, then the selection round of the first phase can be simplified by having a predetermined process (an initial coordinator) that sends its initial value to all processes. Processes set $selected_p$ to the received value without executing the $FLV(\vec{\mu}_p^r)$ function. If the initial coordinator is a correct process, then all correct processes might set $selected_p$ to the same value, and a decision in possible in the first phase.

(iv) The functionality of the decision round of phase ϕ and of the selection round of phase $\phi + 1$ can be provided in one single round.

5 Minimality results related to T_D

This section states two minimality results related to T_D. Theorem 3 establishes the necessity of $T_D > \frac{n+3b+f}{2}$ when

[14]Note that it is safe to select $init_p$ at the first round for the following reason. If no value is initially locked, then any value may be selected by honest process. If some value v is initially locked, then by definition all honest processes have $init_p = v$, and all honest processes select v.

$FLAG = *$. Theorem 4 establishes the necessity of $T_D > 2b + f$ when $FLAG = \phi$.

To simplify the proofs, we consider here the binary consensus problem. Furthermore, we introduce several new notations; a message $\langle v, 0, \{(v, 0)\}\rangle$ sent when the local vote 0 has never been updated, neither validated, is denoted by m_v; a message $\langle v, 0, \{(v, 0), (v', 1)\}\rangle$ sent when the local vote has been updated only once, but has never been validated, is denoted by $m_{v,v'}$. Finally, the sign $*$ means 0 or 1. Therefore, m_* denotes any message sent when the local vote has never been updated, neither validated.

Theorem 3. *If $FLAG = *$, then there is no function implementing $FLV(\vec{\mu}_p^r)$ with $T_D \leq \frac{n+3b+f}{2}$.*

Proof. The proof is by contradiction and is based on the construction of three different runs. Assume that there is a function implementing $FLV(\vec{\mu}_p^r)$ with $T_D \leq \frac{n+3b+f}{2}$ and $FLAG = *$.

Let run $\mathcal{R}$ be such that (1) a set Π_0 of $T_D - 2b - f$ correct processes and f faulty (but honest) processes have initial value 0, and (2) a set Π_1 of $n - T_D + b$ correct processes have initial value 1. Imagine that in the first three rounds all messages are lost. Then, assume that in selection round $r = 4$, a correct process $p \in \Pi_1$ receives all messages sent by correct processes, while all other messages are lost. By the FLV-liveness property, the value returned on p by $FLV(\vec{\mu}_p^r)$ is not $null$. Therefore, if $FLV(\vec{\mu}_p^r)$ considers $T_D - 2b - f$ messages m_0 and $n - T_D + b$ messages m_1, it cannot return $null$ (*).

Let us now consider run $\mathcal{R}'$ in which (1) a set Π_0 of $T_D - b - f$ correct processes and the f faulty (but honest) processes have initial value 0, and (2) a set Π_1 of $n - T_D$ correct processes have initial value 1. Assume that in the first two rounds, all messages are lost. Imagine that in decision round $r = 3$, a correct process $q \in \Pi_0$ receives T_D messages m_0 from all correct processes in the Π_0, all f faulty processes and all Byzantine processes, while all other messages are lost. Therefore, process q decides 0 in round $r = 3$, and 0 is locked in $r > 3$ (**). Then, consider that in selection round $r = 4$, a correct process $p \in \Pi_1$ receives (1) $T_D - 2b - f$ messages m_0 from processes in Π_0 (excluding q), and (2) $n - T_D + b$ messages m_1 from all processes in Π_1 and all b Byzantine processes. Thus, by property (*), $FLV(\vec{\mu}_p^r)$ cannot return $null$ on p. By (**) and the FLV-agreement property, it can only return 0. Therefore, if $FLV(\vec{\mu}_p^r)$ considers $T_D - 2b - f$ messages m_0 and $n - T_D + b$ messages m_1, it returns 0 (***).

Let us now consider run $\mathcal{R}''$ in which (1) a set Π_0 of $T_D - 3b - f$ correct processes have initial value 0, and (2) a set Π_1 of $n - T_D + 2b$ correct processes and the f faulty (but honest) processes have initial value 1. Assume that in the first two rounds, all messages are lost. Imagine that in decision round $r = 3$, a correct process $q' \in \Pi_1$ receives a message m_1 from all processes in the set Π_1, all f faulty processes and all Byzantine processes, while all other messages are lost. Because $T_D < \frac{n+3b+f}{2}$, we have that $|\Pi_1| + b = n - T_D + 3b + f \geq T_D$. Therefore, process q' decides 1 in round $r = 3$, and 1 is locked in $r > 3$ (****). Then, consider that in selection round $r = 4$, a correct process $p \in \Pi_1$ receives (1) $T_D - 2b - f$ messages m_0 from processes in Π_0 and all b Byzantine processes, and (2) $n - T_D + b$ messages m_1 from processes in Π_1 (excluding q'). Thus, by property (***), $FLV(\vec{\mu}_p^r)$ must return 0 on p. A contradiction with the FLV-agreement property and (****). $\square$

Theorem 4. *If $FLAG = \phi$, then there is no function implementing $FLV(\vec{\mu}_p^r)$ with $T_D \leq 2b + f$.*

Proof. The proof is by contradiction and is based on the construction of three runs. Consider any deterministic function for line 11 of Algorithm 4 and an instantiation of $Validator$ function that always return the whole set of processes Π. Let us denote by X the threshold such that $\frac{n-2f-b}{2} < X < \frac{n-b}{2}$. Assume that there is a function implementing $FLV(\vec{\mu}_p^r)$ with $T_D \leq 2b + f$.

Consider a run $\mathcal{R}$ where at the end of the first selection round $r = 1$, (1) X correct processes select 0, and (2) $n - b - f - X$ correct processes select 1. Imagine that all messages are lost in the second and third round. Then, assume that all messages from correct processes are received by a process p in round $r = 4$ (selection round of phase $\phi = 2$). By the FLV-liveness property, the value returned on p by $FLV(\vec{\mu}_p^r)$ is not $null$. Therefore, if $FLV(\vec{\mu}_p^r)$ considers X messages $m_{*,0}$ and $n - b - f - X$ messages $m_{*,1}$, it cannot return $null$ (*).

Consider a run $\mathcal{R}'$ that is initially the same as run $\mathcal{R}$. Furthermore, assume that at the end of the first selection round $r = 1$, (1) X correct processes and all faulty (but honest) processes select 0, and (2) $n - b - f - X$ correct processes select 1. In validation round $r = 2$, only b correct processes (that selected 0) and f faulty processes receive $X + f + b$ messages equal to 0 (from X correct and f faulty (but honest) processes that selected 0 and b from Byzantine processes). All other messages are lost.

Since $\frac{n-2f-b}{2} < X$, we have $X + f + b > \frac{n+b}{2}$, i.e., b correct processes and the f faulty processes validate value 0. Other honest processes does not validate any value since they haven't received any message. In decision round $r = 3$, consider that a correct process q receives a validated vote 0 from the b correct processes, the f faulty processes and the b Byzantine processes. Therefore, process q decides 0, which means that 0 is locked in $r > 3$ (**). In selection round $r = 4$, assume that a correct process p receives (1) $X - b$ messages $m_{*,0}$ from the correct processes that select 0 in round $r = 1$ but do not validate 0 in round $r = 2$, (2) b messages $m_{*,0}$ from the Byzantine processes, and (3)

$n-b-f-X$ messages $m_{*,1}$ from the correct processes that select 0 in round $r=1$. By (*), (**) and the FLV-agreement property, the function $FLV(\vec{\mu}_p^r)$ returns 0. Thus, if $FLV(\vec{\mu}_p^r)$ considers X messages $m_{*,0}$ and $n-b-f-X$ messages $m_{*,1}$, it must return 0 (***).

Consider a run $\mathcal{R}''$ that is initially the same as run $\mathcal{R}$ and run $\mathcal{R}'$. Furthermore, we assume that at the end of the first selection round $r=1$, (1) X correct processes select 0, and (2) $n-b-f-X$ correct processes and all faulty (but honest) processes select 1. In validation round $r=2$, only b correct processes (that selected 0) and f faulty processes receive $n-X$ message equal to 1 ($n-b-f-X$ messages from correct processes that selected 1, f from faulty (but honest) that selected 1 and b messages from Byzantine processes). All other messages are lost.

Since $X < \frac{n-b}{2}$, we have $n-X > \frac{n+b}{2}$, and b correct processes and the f faulty processes validate the vote 1. In decision round $r=3$, consider that a correct process q receives a validated vote 1 from the b correct processes, the f faulty processes and the b Byzantine processes. Therefore, process q decides 1, which means that 1 is locked in $r>3$ (****). In selection round $r=4$, assume that a correct process p receives (1) $X-b$ messages $m_{*,0}$ from the correct processes that select 0 in round $r=1$ (2) $n-b-f-X$ messages $m_{*,1}$ from the correct processes that select 1 in round $r=1$ but do not validate 1 in round $r=2$, and (3) b messages $m_{*,0}$ from the Byzantine processes. Thus, by property (***), $FLV(\vec{\mu}_p^r)$ must return 0 on p. A contradiction with the FLV-agreement property and (****). □

6 Instantiations of Parameters and Classification of Algorithms

We present now instantiations of FLV and $Validator$. The FLV function is used to find the locked value, therefore depends on the decision mechanism, i.e., on T_D and $FLAG$. We identify three instantiations of the FLV function. The first one is for the case $FLAG = *$ and $T_D > \frac{n+3b+f}{2}$, and uses only variable $vote_p$; the second one is for the case $FLAG = \phi$ and $T_D > 3b+f$, and uses variables $vote_p$ and ts_p; the last one is for the case $FLAG = \phi$ and $T_D > 2b+f$, and uses all three variables $vote_p$, ts_p and $history_p$. This leads to three classes of consensus algorithms, as shown in Table 1. Algorithms that belong to the same class have the same values for the parameters $FLAG$ and T_D. Therefore algorithms from the same class have the same constraint on n (follows from $n \geq T_D + b + f$) and have the same number of rounds (depending on the value of $FLAG$). Note that the first round of each phase is simulated using several micro-rounds, in order for $\mathcal{P}_{cons}$ to eventually hold (Sect. 3.2). The other rounds of each phase are ordinary rounds in which $\mathcal{P}_{good}$ eventually holds.

One can observe the following tradeoff among these three classes. When $FLAG = *$ and $T_D > \frac{n+3b+f}{2}$ (class 1), only two rounds per phase are needed and the process state is the smallest, but class 1 requires the largest n ($n > 5b+3f$). The "Examples" column of Table 1 lists well-known algorithms that correspond to a given class. These examples are discussed in Section 7.

We can make the following comments. First, to the best of our knowledge, no existing algorithm corresponds to class 2, case $f=0$ (Byzantine faults). We call this new algorithm MQB (*Masking Quorum Byzantine* consensus algorithm).[15] Second, Table 1 shows that despite its name, the FaB Paxos algorithm does not belong to the same class as the Paxos algorithm.

We now present the three instantiations of the FLV function that lead to the three classes of consensus algorithms. Instantiations of the $Validator$ function are discussed later.

6.1 Instantiations of $FLV(\vec{\mu}_p^r)$

We give here the instantiations of FLV for the three classes of algorithm.

6.1.1 $FLV(\vec{\mu}_p^r)$ for class 1

We start with the FLV function for class 1 ($FLAG = *$ and $T_D > \frac{n+3b+f}{2}$), see Algorithm 5.

Algorithm 5 $FLV(\vec{\mu}_p^r)$ for class 1

1: $correctVotes_p \leftarrow \Big\{ v : \Big| \Big\{ (v,-,-) \in \vec{\mu}_p^r \Big\} \Big| > n - T_D + b \Big\}$
2: **if** $|correctVotes_p| = 1$ **then**
3:     **return** v $s.t.$ $v \in correctVotes_p$
4: **else if** $|\vec{\mu}_p^r| > 2(n - T_D + b)$ **then**
5:     **return** ?
6: **else**
7:     **return** $null$

Line 1 is for FLV-agreement, as we now explain with a simple example. Let v_1 be locked in round r because some honest process p has decided v_1 in round $r-1$. By Algorithm 4, p has received in the decision round $r-1$ at least T_D votes v_1. At least $T_D - b$ votes v_1 are from honest processes, i.e., a process can receive at most $n-(T_D-b)$ votes equal to $v_2 \neq v_1$ (*). Therefore, the condition of line 1 can only hold for v_1, i.e., among the values different from ? and $null$, FLV can only return v_1. For FLV-agreement to hold, Algorithm 5 must also prevent ? to be returned when v_1 is locked. The condition of line 4 ensures this. Here is why. Assume that the condition of line 4 holds. This means that $\vec{\mu}_p^r$ contains more than $2(n-T_D+b)$ messages. With (*), $\vec{\mu}_p^r$ contains more than $n-T_D+b$ messages equal to v_1. By line 1, we have $v_1 \in correctVotes_p$, and as shown

[15] The quorums used in this algorithm satisfy the property of *masking quorums* [16]. Note that with respect to the definitions in [16], algorithms of class 1 use *opaque quorums*, and algorithms of class 3 use *dissemination quorums*.

Table 1. The three classes of consensus algorithms.

	$FLAG$	T_D	n	Variables	Rounds per phase	Examples
1	$*$	$> \frac{n+3b+f}{2}$	$> 5b+3f$	$(vote_p)$	2 $(\mathcal{P}_{cons}; \mathcal{P}_{good})$	OneThirdRule [5] ($b=0$) FaB Paxos [17] ($f=0$)
2	ϕ	$> 3b+f$	$> 4b+2f$	$(vote_p, ts_p)$	3 $(\mathcal{P}_{cons}; \mathcal{P}_{good}; \mathcal{P}_{good})$	Paxos [11] ($b=0$), CT [4] ($b=0$) MR [20] ($b=0$), b-DLS [6] ($b=0$) MQB ($f=0$) *(new alg)*
3	ϕ	$> 2b+f$	$> 3b+2f$	$(vote_p, ts_p, history_p)$	3 $(\mathcal{P}_{cons}; \mathcal{P}_{good}; \mathcal{P}_{good})$	PBFT [3] ($f=0$) B-DLS [6] ($f=0$)

above, only v_1 can be in $correctVotes_p$. Therefore, the condition of line 2 holds: Algorithm 5 cannot return ? when v_1 is locked.

Property FLV-liveness is ensured by lines 4 and 5. This is because when $T_D > \frac{n+3b+f}{2}$, we have $n-b-f > 2(n-T_D+b)$. Therefore, receiving a message from all correct processes (i.e., $|\vec{\mu}_p^r| \geq n-b-f$) implies that the condition of line 4 holds, i.e., $null$ is not returned. Property FLV-validity is trivially ensured by lines 1-3.

Theorem 5. *If $FLAG = *$, then Algorithm 5 ensures FLV-validity and FLV-agreement. FLV-liveness holds if $T_D > \frac{n+3b+f}{2}$.*

Proof.

FLV-validity: FLV-validity follows from lines 1-3.

FLV-agreement: Let $r = 3\phi - 2$ be the smallest selection round in which value v is locked. By definition of a locked value, we have two cases to consider (1) all honest processes have $vote_p = v$ and unanimity must be ensured (and $r=1$), or (2) v has been decided in round $r' = 3\phi - 3$ by some honest process p. We now show that for both cases at least $T_D - b$ honest processes sent $\langle v, -, - \rangle$ in round r (*).

Case 1: Trivially follows from initialization and $T_D \leq n-b-f$.

Case 2: By Algorithm 4, the process p received at least T_D messages $\langle v, - \rangle$ in round r'. Therefore, at least $T_D - b$ honest processes send $\langle v, - \rangle$ in round r', i.e., at least $T_D - b$ honest processes have their vote set to v, and send $\langle v, -, - \rangle$ in round r.

We now show that when property (*) holds, Algorithm 5 ensures FLV-agreement. Assume for the contradiction that a non null value $v' \neq v$ is returned. Two cases must be considered.

v' is returned at line 3: Because $correctVotes_p$ is not empty, the set $\vec{\mu}_p^r$ contains more than $n - T_D + b$ messages $\langle v', -, - \rangle$. A contradiction with (*).

? is returned at line 5: This means that $\vec{\mu}_p^r$ contains more than $2(n - T_D + b)$ messages. By (*), $\vec{\mu}_p^r$ contains at most $n - T_D + b$ messages $\langle v' \neq v, -, - \rangle$, and therefore, more than $n - T_D + b$ messages $\langle v, -, - \rangle$. By line 1, the set $correctVotes_p$ is not empty. By lines 2-3, value v is returned. A contradiction.

This shows that Algorithm 5 ensures FLV-agreement in round r. Therefore, no honest process p updates its variable $vote_p$ and $select_p$ to a value $v' \neq v$ in selection round r. Because $FLAG = -$, the validation round is skipped. Therefore, property (*) holds in selection round $r'' = 3\phi + 1$. With similar arguments as above, we can show that Algorithm 5 ensures FLV-agreement in round r''. By a simple induction on ϕ, we can show that Algorithm 5 ensures FLV-agreement in all rounds.

FLV-liveness: Property FLV-liveness is ensured by lines 4-5. This is because when $T_D > \frac{n+3b+f}{2}$, we have $n-b-f > 2(n-T_D+b)$. Therefore, receiving messages from all correct processes (i.e., $|\vec{\mu}_p^r| \geq n-b-f$) implies that the condition of line 4 holds. □

6.1.2 $FLV(\vec{\mu}_p^r)$ for class 2

For class 2 we can have $T_D \leq \frac{n+3b+f}{2}$, which means that the locked value cannot be detected only based on votes: the timestamp ts_p is needed. The FLV function for class 2 ($FLAG = \phi$ and $T_D > 3b+f$) is shown in Algorithm 6.

Lines 1 (where $\{\# \ldots \#\}$ denotes a multiset) and 2 are for FLV-agreement, as we now explain with a simple example. Let v_1 be locked in round r, phase $\phi_1 + 1$, because some honest process p has decided v_1 in round $r-1$, phase ϕ_1. By Algorithm 4, p has received in the decision round $r-1$ at least T_D messages $\langle v_1, \phi_1 \rangle$. At least $T_D - b$ messages are from honest processes that have $vote_p = v_1$ and $ts_p = \phi_1$, i.e., at most $n - b - (T_D - b) = n - T_D$ honest processes have $vote_p = v_2 \neq v_1$ (*). Because only one value can be validated by honest processes in phase ϕ_1

Algorithm 6 $FLV(\vec{\mu}_p^r)$ for class 2

1: $possibleVotes_p \leftarrow \{\#\,(vote, ts, -) \in \vec{\mu}_p^r :$
   $|\{(vote', ts', -) \in \vec{\mu}_p^r : \ vote = vote' \vee ts > ts'\}|$
   $> n - T_D + b\,\#\}$
2: $correctVotes_p \leftarrow \{(vote,-,-) \in possibleVotes_p :$
   $|\{\#(vote',-,-) \in possibleVotes_p : \ vote = vote'\#\}| > b\}$
3: **if** $|correctVotes_p| = 1$ **then**
4:   **return** $v \ s.t. \ (v, -, -) \in correctVotes_p$
5: **else if** $|\vec{\mu}_p^r| > n - T_D + 2b$ **then**
6:   **return** ?
7: **else**
8:   **return** $null$

(Lemma 2), all honest processes with $vote_p = v_2 \neq v_1$ have $ts_p < \phi_1$. It follows that for every honest process p, we have $vote_p = v_1$ or $ts_p < \phi_1$ (**). Together with (*), no message $\langle v_2 \neq v_1, -, - \rangle$ sent by an *honest* process can satisfy the condition of line 1. In other words, the set $possibleVotes_p$ may contain at most b messages $\langle v_2 \neq v_1, -, - \rangle$, namely the messages sent by Byzantine processes. Line 2 prevents such messages to be in $correctVotes_p$. This shows that among the values different from ? and $null$, only v_1 can be returned.

For FLV-agreement to hold, Algorithm 6 must also prevent ? to be returned when v_1 is locked. The condition of line 5 ensures this. Here is why. Assume that the condition of line 5 holds. This means that $\vec{\mu}_p^r$ contains more than $n - T_D + 2b$ messages. From (*), a process can receive at most $n - T_D + b$ messages with $vote = v_2 \neq v_1$. It follows that the set $\vec{\mu}_p^r$ contains at least $b + 1$ messages $\langle v_1, \phi_1, - \rangle$ from honest processes. With (**) and the fact that $\vec{\mu}_p^r$ contains more than $n - T_D + b$ messages from honest processes, the $b + 1$ messages $\langle v_1, \phi_1, - \rangle$ satisfy the condition of line 1. By line 2, $\langle v_1, \phi_1, - \rangle$ is in $correctVotes_p$. Moreover, as discussed above, only v_1 can be in $correctVotes_p$. Therefore, the condition of line 3 holds: Algorithm 6 cannot return ? when v_1 is locked.

Property FLV-liveness is ensured by lines 5, 6. This is because when $T_D > 3b + f$, we have $n - b - f > n - T_D + 2b$. Therefore, receiving a message from all correct processes (i.e., $|\vec{\mu}_p^r| \geq n - b - f$) ensures that the condition of line 5 holds, i.e., $null$ cannot be returned. Property FLV-validity is trivially ensured by lines 1-4.

Theorem 6. *If $FLAG = \phi$, $Validator(p, \phi)$-validity and $Validator(p, \phi)$-agreement hold, then Algorithm 6 ensures FLV-validity and FLV-agreement. FLV-liveness holds if in addition $T_D > 3b + f$.*

Proof.

FLV-validity: FLV-validity follows from lines 1-4.

FLV-agreement: Let $r = 3\phi - 2$ be the smallest selection round in which value v is locked. By definition of a locked value, we have two cases to consider (1) all honest processes have $vote_p = v$ and unanimity must be ensured (and $r = 1$), or (2) v has been decided in round $r' = 3\phi - 3$ by some honest process p. We now show that for both cases in round r at least $T_D - b$ honest processes sent $\langle v, \hat{\phi} \geq \phi - 1, - \rangle$ (*), and for all honest processes q, we have $(vote_q = v \vee ts_q < \phi - 1)$ (**).

Case 1: Trivially follows from initialization and $T_D \leq n - b - f$.

Case 2: By Algorithm 4, the process p received at least T_D messages $\langle v, \phi - 1 \rangle$ in round r'. Therefore, at least $T_D - b$ honest processes send $\langle v, \phi - 1 \rangle$ in round r', i.e., at least $T_D - b$ honest processes have their vote set to v, and send $\langle v, \phi - 1, - \rangle$ in round r (which shows (*)). This means that an honest process p updates $vote_p$ to v and ts_p to ϕ in the validation round of phase $\phi - 1$. By $Validator(p, \phi)$-validity, $Validator(p, \phi)$-agreement and Lemma 2, no honest process update its vote to a value $v' \neq v$ in this validation round (which shows (**)).

We now show that when properties (*) and (**) hold, Algorithm 6 ensures FLV-agreement. Assume for the contradiction that a non null value $v' \neq v$ is returned. Two cases must be considered.

v' is returned at line 4: Because $correctVotes_p$ is not empty, the multi-set $possibleVotes_p$ contains more than b messages $\langle v', -, - \rangle$. It follows that an honest process sent a message $\langle v', \phi', - \rangle$. By (**), $\phi' < \phi - 1$. By line 1, more than $n - T_D$ honest processes sent a message $\langle v'', \phi'', - \rangle$ with $v'' = v'$ or $\phi'' < \phi' < \phi - 1$. A contradiction with (*).

? is returned at line 6: This means that $\vec{\mu}_p^r$ contains more than $n - T_D + 2b$ messages (and more than $n - T_D + b$ messages from honest processes (***)). By (*), $\vec{\mu}_p^r$ contains at most $n - T_D + b$ messages different from $\langle v, \hat{\phi} \geq \phi - 1, - \rangle$, and therefore, more than b messages $\langle v, \hat{\phi} \geq \phi - 1, - \rangle$. By line 1, (**) and (***), the multi-set $possibleVotes_p$ contains more than b messages $\langle v, -, - \rangle$. Therefore, the set $correctVotes_p$ contains a message $\langle v, -, - \rangle$. By lines 3-4, value v is returned. A contradiction.

This shows that Algorithm 6 ensures FLV-agreement in round r. Therefore, no honest process p updates its variable $vote_p$ and $select_p$ to a value $v' \neq v$ in selection round r. By $Validator(p, \phi)$-validity and Lemma 1, no honest process p updates its vote to a value $v' \neq v$ in the validation round $r + 1$. Therefore, properties (*) and (**) hold in selection round $r'' = 3\phi + 1$. With similar arguments as above, we can show that Algorithm 6 ensures FLV-agreement in round r''. By a simple induction on ϕ, we can show that Algorithm 6 ensures FLV-agreement in all rounds.

FLV-liveness: Property FLV-liveness follows from lines 5 and 6. This is because when $T_D > 3b + f$, we have

$n - b - f > n - T_D + 2b$. Therefore, receiving messages from all correct processes (i.e., $\vec{\mu}_p^r \geq n-b-f$) ensures that the condition of line 5 holds. □

6.1.3 $FLV(\vec{\mu}_p^r)$ for class 3

For class 3 we can have $T_D \leq 3b+f$, which means that the locked value cannot be detected based on votes and timestamps: the $history$ log is needed. The FLV function for class 3 ($FLAG = \phi$ and $T_D > 2b + f$) is shown in Algorithm 7.

Algorithm 7 $FLV(\vec{\mu}_p^r)$ for class 3

1: $possibleVotes_p \leftarrow \{ (vote, ts, -) \in \vec{\mu}_p^r :$
$\quad |\{(vote', ts', -) \in \vec{\mu}_p^r : vote = vote' \vee ts > ts'\}| > n - T_D + b \}$
2: $correctVotes_p \leftarrow \{v : (v, ts, -) \in possibleVotes_p \wedge$
$\quad |\{(vote', ts', history') \in \vec{\mu}_p^r : (v, ts) \in history'\}| > b \}$
3: **if** $|correctVotes_p| = 1$ **then**
4: $\quad$ **return** v $s.t.$ $(v, -, -) \in correctVotes_p$
5: **else if** $|correctVotes_p| > 1$ **then**
6: $\quad$ **return** ?
7: **else if** $| \left\{(vote, ts, -) \in \vec{\mu}_p^r : ts = 0\right\} | > n - T_D + b$ **then**
8: $\quad$ **if** there is a value v such that $\vec{\mu}_p^r$ contains a majority of messages $(v,-,-)$ **then** /* only for unanimity */
9: $\quad\quad$ **return** v
10: $\quad$ **else**
11: $\quad\quad$ **return** ?
12: **else**
13: $\quad$ **return** $null$

Similarly to Algorithm 6, lines 1 and 2 are for FLV-agreement, as we now explain with a simple example. Let v_1 be locked in round r, phase $\phi_1 + 1$, because some honest process p has decided v_1 in round $r - 1$, phase ϕ_1.

For the same reason as for Algorithm 6, at least $T_D - b$ honest processes have $vote_p = v_1$ and $ts_p = \phi_1$ (*), i.e., at most $n - T_D$ honest processes have $vote_p = v_2 \neq v_1$. Furthermore, as for class 2, for every honest process p, we have $vote_p = v_1$ or $ts_p < \phi_1$ (**). Together with (*), no message $\langle v_2 \neq v_1, -, -\rangle$ sent by an *honest* process can satisfy the condition of line 1. Said differently, apart from messages $\langle v_1, -, -\rangle$, only messages $\langle v_2 \neq v_1, \phi_2, -\rangle$ sent by Byzantine processes can be in the set $possibleVotes_p$. Because honest processes can only update history at line 14 of Algorithm 4, no honest process has a pair $(-, \phi_2 > \phi_1)$ in its history in the sending step of round r. It follows that only messages $\langle v_1, -, -\rangle$ can be in $correctVotes_p$ at line 2. Therefore, when a value v_1 is locked, lines 1 and 2 prevent any value $v \neq v_1$ or $v = ?$ to be returned at lines 4 and 6. By (*) together with $\phi_1 > 0$, condition of line 7 never holds in our example.

To understand the role of lines 8-11, we have to consider another example. Let all honest processes have initially $vote_p = v_1$. With the same arguments as above, it follows that no value different from v_1 or $null$ can be returned at lines 4 and 6. However, the condition of line 7 might hold. In this case, $\vec{\mu}_p^r$ contains more than $n - T_D$ messages $\langle v_1, 0, -, -\rangle$ from honest processes, and at most b messages $\langle v_2 \neq v_1, 0, -\rangle$ from Byzantine processes. Because $T_D \leq n - b - f$, we have $n - T_D \geq b$, and v_1 is returned at line 9. In other words, line 9 ensures FLV-agreement when unanimity is considered.

Let us now discuss FLV-liveness. For this property to hold, we need a stronger variant of $Validator\text{-}validity$:[16]

- $Validator\text{-}strongValidity$:
  If $|Validator(p, \phi)| > 0$ then $|Validator(p, \phi)| > 3b$.

With $Validator\text{-}strongValidity$ we can have a stronger variant of Lemma 1 (the proof follows directly from the proof of Lemma 1):

Lemma 3. *If $Validator\text{-}strongValidity$ holds, then the following property holds on every honest process h and in every phase ϕ: if process h set $vote_h$ to v and ts_h to ϕ at lines 21-22 of Algorithm 4, then at least $b+1$ honest process has sent $\langle v\rangle$ at line 18.*

Let $\vec{\mu}_p^r$ contain the messages from all the $n - b - f$ correct processes. There are two cases to consider: (1) correct processes sent only $\langle -, 0, -\rangle$, (2) at least one correct process sent $\langle -, ts > 0, -\rangle$. Note that $T_D > 2b + f$ ensures $n - b - f > n - T_D + b$ (*). In case (1), by (*) the condition of line 7 holds, and $null$ cannot be returned at line 13 of Algorithm 7. In case (2), let ν denote the subset of messages in $\vec{\mu}_p^r$ that are from correct processes, and let ts_ν be the highest timestamp in ν. By Lemma 2 there is a unique value v_ν such that $\langle v_\nu, ts_\nu, -\rangle \in \nu$. Together with (*), this ensures that the set $possibleVotes_p$ is not empty, and contains $\langle v_\nu, ts_\nu, -\rangle$. By Lemma 3, any correct process that validates v_ν in the validation round $3\,ts_\nu - 1$ received v_ν from at least $b+1$ correct processes. Therefore, at least $b+1$ correct processes have selected v_ν in round $3\,ts_\nu - 2$, and these processes have (v_ν, ts_ν) is their history. This implies that the set $correctVotes_p$ is non empty, and a non-$null$ value is returned at line 4 or 6.

Theorem 7. *If $FLAG = \phi$, $Validator(p, \phi)$-validity and $Validator(p, \phi)$-agreement hold, then Algorithm 7 ensures FLV-validity and FLV-agreement. FLV-liveness holds if in addition $T_D > 2b + f$ and $Validator$-strongValidity holds.*

Proof.

FLV-validity: FLV-validity follows from the lines 1-4 and 8-9.

FLV-agreement: Let $r = 3\phi - 2$ be the smallest selection round in which value v is locked. By definition of a

[16] This stronger variant was not introduced in Section 4.4, since the proof of the generic Algorithm 4 does not require the stronger variant. In the proof of Algorithm 4, the stronger variant is hidden in the FLV-$liveness$ property.

locked value, we have two cases to consider (1) all honest processes have $vote_p = v$ and unanimity must be ensured (and $r = 1$), or (2) v has been decided in round $r' = 3\phi - 3$ (and thus, $\phi - 1 \geq 1$) by some honest process p. We now show that for both cases in round r at least $T_D - b$ honest processes sent $\langle v, \hat{\phi} \geq \phi - 1, - \rangle$ (*), for all honest processes q, we have $(vote_q = v \vee ts_q < \phi - 1)$ (**), and for any element $(vote, ts)$ in the set $history_q$ of any honest process q, we have $(ts \leq \phi - 1)$ (***). In addition, if less than $T_D - b$ honest processes have $ts_p > 0$, then all honest processes have $vote_p = v$ (****).

Case 1: Trivially follows from initialization and $T_D \leq n - b - f$.

Case 2: By Algorithm 4, the process p received at least T_D messages $\langle v, \phi - 1 \rangle$ in round r'. Therefore, at least $T_D - b$ processes send $\langle v, \phi - 1 \rangle$ in round r', i.e., at least $T_D - b$ honest processes have their vote set to v, and send $\langle v, \phi - 1, - \rangle$ in round r (which shows (*)). This means that an honest process p updates $vote_p$ to v and ts_p to ϕ in the validation round of phase $\phi - 1$. By $Validator(p, \phi)$-validity, $Validator(p, \phi)$-agreement and Lemma 2, no honest process update its vote to a value $v' \neq v$ in this validation round (which shows (**)).

Property (***) trivially follows from the fact that for each honest process the last update of history occured in round $3(\phi - 1) - 2$. Property (****) trivially follows from $\hat{\phi} \geq \phi - 1 \geq 1$ and (*), which implies that the precondition of (****) cannot be true.

We now show that when properties (*), (**), (***) and (****) hold, Algorithm 7 ensures FLV-agreement. Assume for the contradiction that a non null value $v' \neq v$ is returned. Four cases must be considered.

v' is returned at line 4: Because $correctVotes_p$ is not empty, the set $\vec{\mu}_p^r$ contains a message $\langle v', \phi', - \rangle$ in $possibleVotes_p$ such that an honest process h has (v', ϕ') in $history_h$ (see line 2). By (***), $\phi' \leq \phi - 1$. By line 1, more than $n - T_D$ honest processes sent a message $\langle v'', \phi'', - \rangle$ with $v'' = v'$ or $\phi'' < \phi' \leq \phi - 1$. A contradiction with (*).

? is returned at line 6: Same arguments as the case v' is returned at line 4.

v' is returned at line 9: By line 7, the set $\vec{\mu}_p^r$ contains more than $n - T_D + b$ messages $\langle -, 0, - \rangle$. Therefore, less than $T_D - b$ honest processes has $ts_p > 0$. By (****), all honest processes has $vote_p = v$. Therefore $\vec{\mu}_p^r$ contains more than $n - T_D$ messages $\langle v, 0, - \rangle$ and at most b messages $\langle v', 0, - \rangle$. Because $T_D \leq n - b - f$, there is a majority of messages $\langle v, 0, - \rangle$ in $\vec{\mu}_p^r$. A contradiction with line 8 and the fact that v' is returned at line 9.

? is returned at line 11: Same arguments as the case v' is returned at line 9.

This shows that Algorithm 7 ensures FLV-agreement in round r. Therefore, no honest process p updates its variable $vote_p$ and $select_p$ to a value $v' \neq v$ in selection round r. Furthermore, no honest process p adds a tuple $(v' \neq v, \phi)$ in selection round r. By $Validator(p, \phi)$-validity and Lemma 1, no honest process p updates its vote to a value $v' \neq v$ in the validation round $r + 1$. Therefore, properties (*), (**), (***) and (****) hold in selection round $r'' = 3\phi + 1$. With similar arguments as above, we can show that Algorithm 7 ensures FLV-agreement in round r''. By a simple induction on ϕ, we can show that Algorithm 7 ensures FLV-agreement in all rounds.

FLV-liveness: Let $\vec{\mu}_p^r$ contain the messages from all the $n - b - f$ correct processes. There are two cases to consider: (1) correct processes sent only $\langle -, 0, - \rangle$, (2) at least one correct process sent $\langle -, ts > 0, - \rangle$. Note that $T_D > 2b + f$ ensures $n - b - f > n - T_D + b$ (*). In case (1), by (*) the condition of line 7 holds, and $null$ cannot be returned at line 13. In case (2), let S denote the subset of messages in $\vec{\mu}_p^r$ that are from correct processes, and let ts_S be the highest timestamp in S. By $Validator(p, \phi)$-validity, $Validator(p, \phi)$-agreement and Lemma 2, there is a unique value v_S such that $\langle v_S, ts_S, - \rangle \in S$. Together with (*), this ensures that the set $possibleVotes_p$ is not empty, and contains $\langle v_S, ts_S, - \rangle$. $Validator$-strongValidity ensures that $|validators_p| > 0$ implies $|validators_p| > 3b$. As a result, any correct process that validates v_S in the validation round $3\,ts_S - 1$ received $\langle v_S, - \rangle$ from more than $\frac{(3b)+b}{2} = 2b$ processes. Therefore, at least $b + 1$ correct processes have selected v_S in round $3\,ts_S - 2$, and these processes have (v_S, ts_S) in their history. This implies that the set $correctVotes_p$ is non empty, and a non-$null$ value is returned at line 4 or 6. □

6.2 Instantiations of $Validator(p, \phi)$

A trivial instantiation of the $Validator$ function consists in always returning the whole set of processes Π. This trivially satisfies $Validator$-validity, $Validator$-strongValidity, $Validator$-agreement and $Validator$-liveness. To our knowledge, this instantiation is used in all algorithms for Byzantine faults. However, another possible instantiation can be considered in the Byzantine fault model: it consists in returning the same set S of $b + 1$ processes at every process, e.g., S defined by a deterministic function of the phase ϕ.[17]

In the benign fault model, it is sufficient that the $Validator$ function always returns a single process rather than a set of processes. One such instantiation is the well known *rotating coordinator* function used in [4]. Another example involves message exchange (these messages can

[17] Note that this instantiation does not satisfy $Validator$-strongValidity.

be piggybacked on existing messages). In each phase every process chooses a potential validator q and informs q by sending him a message. If some process q receives messages from a majority, then q becomes the validator, and q informs all processes that the output of $Validator$ function is q. If a process does not receive such a message within some timeout, the $Validator$ function returns $\emptyset$. It can easily be shown that this instantiation satisfies $Validator$-validity, $Validator$-agreement and $Validator$-liveness.[18] We call this instantiation *majority voting validator selection*; it is used for example in the *prepare phase* of Paxos [11].

7 Instantiation examples

In this section we show several well-known consensus algorithms derived from Algorithm 4. Note that even though the instantiated algorithms are expressed in the round model, which is not the case of many well-known consensus algorithms, the core mechanisms are the same.[19]

7.1 Class 1 algorithms

OneThirdRule [5] The OneThirdRule algorithm assumes benign faults only ($b = 0$) and requires $n > 3f$ to tolerate f benign faults.

The instantiated version of the OneThirdRule algorithm (that we call Inst-OneThirdRule), is obtained from Algorithm 4 with the following parametrization: $T_D = \lceil \frac{2n+1}{3} \rceil$,[20] $FLAG = *$ and Algorithm 5 with $T_D = \lceil \frac{2n+1}{3} \rceil$ as the FLV instantiation.

Algorithm 8 OneThirdRule algorithm ($n > 3f$) [5]

1: **Initialization:**
2: $vote_p := init_p$

3: **Round** r**:**
4: S_p^r:
5: send $\langle vote_p \rangle$ to all
6: T_p^r:
7: **if** received more than $2n/3$ messages **then**
8: $x_p :=$ the smallest most often received value
9: **if** more than $2n/3$ values received are equal to v **then**
10: DECIDE v

We now compare Inst-OneThirdRule with the original algorithm (Algorithm 8). In Algorithm 8, the functionality of the selection round and of the decision round are merged into one single round (see optimization (iv), Sect. 4.6). With $T_D = \lceil \frac{2n+1}{3} \rceil$, it is easy to see that the condition for deciding is the same in the two algorithms (compare line 29 of Algorithm 4 and line 9 of Algorithm 8). However, the selection condition of the two algorithms have (minor) differences. Specifically, it is easy to see that whenever some value is selected by Algorithm 8 (lines 7 and 8), then some value (not necessarily the same) is also selected by Algorithm 5. The opposite is not true. If the number of messages received is not larger than $2n/3$, Algorithm 8 will not select any value, while Algorithm 5 may still select a value by line 3.

FaB Paxos [17] The FaB Paxos algorithm is designed for the Byzantine fault model ($f = 0$) and requires $n > 5b$ to tolerate b Byzantine faults. The algorithm is expressed in the context of "proposers", "acceptors" and "learners". For simplicity, in our framework, consensus algorithms are expressed without considering these roles.

The following parametrization leads to Inst-FaB Paxos: $T_D = \lceil (n + 3b + 1)/2 \rceil$, $FLAG = *$ and Algorithm 9 as an instantiation of FLV function (Algorithm 5 with $T_D = \lceil (n + 3b + 1)/2 \rceil$).

We now compare Inst-FaB Paxos with FaB Paxos. With $T_D = \lceil (n+3b+1)/2 \rceil$, the deciding condition is the same in both algorithms. However, the selection condition of the two algorithms have (minor) differences. With FaB Paxos, the selection rule is applied when $n - b$ messages are received. In that case, a value v is selected if it appears at least $\lceil (n - b + 1)/2 \rceil$ times in the set of received messages; otherwise any value can be selected.[21] Therefore, if a number of received messages is smaller than $n - b$, FaB Paxos will not select any value, while Algorithm 9 may still select a value by line 3.

Another difference is that FaB Paxos does not ensure the Unanimity property, which allows a simpler selection round in the first phase (see Optimization (iii), Sect. 4.6).

In [8], Friedman et al. propose an algorithm that is similar to FaB Paxos. The algorithm is designed for Byzantine faults ($f = 0$) and requires $n > 6b$. The algorithm implements the selection round slightly differently than in Algorithm 4. Namely, the selection round is implemented with two micro-rounds, where the logic that provides consistency ($\mathcal{P}_{cons}$) is mixed with the functionality of the selection round.[22] The mechanism requires $n > 6b$. Our

[18] For $Validator$-liveness to hold, it is necessary to have a phase in which all correct processes choose the same validator.

[19] We ignore differences in the way algorithms implement phase change (timeout based mechanisms, failure detector based approaches, sending NACK messages, etc), message acceptance policies, retransmission rules, etc.

[20] T_D is chosen such that the same number of messages allow the condition at line 29 of Algorithm 4 and the condition at line 4 of Algorithm 5 to hold.

[21] Note that the condition at line 1 of Algorithm 9 for selecting a value v requires smaller number of messages to be received than in FaB Paxos. For example, when $n = 7$ and $b = 1$, FaB Paxos requires at least 4 messages equal to v to be received (at least $\lceil (n - b + 1)/2 \rceil (= 4)$), while Algorithm 9 requires 3 messages (more than $\frac{n-b-1}{2} (= 2)$).

[22] Remember that the selection round of Algorithm 4 is simulated using several micro-rounds, in order for $\mathcal{P}_{cons}$ to eventually hold. However, the functionality of the selection round is executed at the end of the $\mathcal{P}_{cons}$ simulation.

intuition is that this mechanism cannot be extended to algorithms that use $(vote_p, ts_p)$ or $(vote_p, ts_p, history_p)$.

Algorithm 9 FLV for class 1 with $T_D = \lceil (n+3b+1)/2 \rceil$

```
correctVotes_p ← { v : | { (v, −, −) ∈ μ⃗_p^r } | > (n−b−1)/2 }
if |correctVotes_p| = 1 then
    return v s.t. v ∈ correctVotes_p
else if |μ⃗_p^r| > n − b − 1 then
    return ?
else
    return null
```

7.2 Class 2 algorithms

Algorithm 10 FLV for class 2 with $b = 0$, $T_D = \lceil \frac{n+1}{2} \rceil$

```
possibleVotes_p ← { (vote, ts, −) ∈ μ⃗_p^r :
    |{(vote′, ts′, −) ∈ μ⃗_p^r : vote = vote′ ∨ ts > ts′}| > n/2
if |possibleVotes| = 1 then
    return v s.t. (v, −, −) ∈ possibleVotes
else if |μ⃗_p^r| > n/2 then
    return ?
else
    return ⊥
```

Paxos [11] Paxos assumes benign faults only ($b = 0$) and requires $n > 2f$.

We get Inst-Paxos from Algorithm 4 with the following parametrization: $T_D = \lceil \frac{n+1}{2} \rceil$,[23] $FLAG = \phi$, $Validator(p, \phi)$ implemented by majority voting validator selection (with messages piggybacked on the selection and validation round messages), and Algorithm 10 as the FLV instantiation. In addition, we apply Optimization (i), Sect. 4.6.

With only benign faults, the instantiation of the function FLV can be simplified. We now explain how to get Algorithm 10 from Algorithm 6. When $b = 0$, the set $correctVotes_p$ is the same as the set $possibleVotes_p$, which means that the set $correctVotes_p$ is not needed.[24]

We now compare Inst-Paxos with Paxos. The decision rule is the same in both algorithms. The selection conditions are not necessarily the same. Paxos selects the vote with the highest timestamp, or any value if there are no votes with $ts > 0$. On the other hand, Inst-Paxos selects the value with the highest timestamp that is locked (returned by $FLV(\vec{\mu}_p^r)$ function, see Algorithm 10). Otherwise, if the $FLV(\vec{\mu}_p^r)$ function returns ?, Inst-Paxos selects any value chosen by some deterministic function (see line 11 of Algorithm 4). If the deterministic function at line 11 returns the value with the highest timestamp, then the selection condition of Inst-Paxos and Paxos are the same.

[23] Same argument as in footnote 20: same value for T_D in the decision round and in the FLV function.

[24] We can also use a set instead of a multiset for $possibleVotes$.

CT [4] Like Paxos, CT — the Chandra-Toueg consensus algorithm using the $\Diamond\mathcal{S}$ failure detector — assumes benign faults only ($b = 0$) and requires $n > 2f$. Paxos and CT use the same selection and decision conditions, and from this point of view rely on the same core mechanisms. The difference is in the $Validator(p, \phi)$ implementation: CT is based on a rotating coordinator.

MR [20] The Mostéfaoui-Raynal algorithm (MR) is designed for benign faults and requires $n > 2f$. It assumes "reliable channels",[25] which allows for some optimizations.

Let us consider an instantiation of Algorithm 4, with $\mathcal{P}_{rel}$ in every round (see Footnote 25), and the following parametrization: $T_D = \lceil \frac{n+1}{2} \rceil$, $Validator(p, \phi)$ implementing the *rotating coordinator function* and Algorithm 11 as the FLV instantiation. We call this algorithm Inst-MR.

We now compare Inst-MR with MR. The validation and the decision condition are the same in both algorithms.[26] Furthermore, the validation round for phase $\phi + 1$ is executed in parallel with decision (and selection) round of phase ϕ. The selection condition of MR algorithm expressed as a $FLV(\vec{\mu}_p^r)$ function (Algorithm 11) is a variant of Algorithm 6. We now explain how to get Algorithm 11 from Algorithm 6. Because $n - f > n - T_D$ and $\mathcal{P}_{rel}$ hold in every round, we have that $|\vec{\mu}_p^r| \geq n - f$ in all rounds and lines 5,7 and 8 of Algorithm 6 can be suppressed. Since the algorithm considers only benign faults and assumes reliable channels, line 1 of Algorithm 11 is equivalent to lines 1-3 of Algorithm 6.

Algorithm 11 Instantiation of FLV function based on MR algorithm [20]

```
1: if received message ⟨v, φ − 1⟩ then
2:     return v
3: else
4:     return ?
```

Note that in the original MR algorithm, a variable ts_p is not explicitly used. Basically, MR needs to distinguish only two cases, $ts_p = \phi$ and $ts_p < \phi$. Instead of $(vote_p, ts_p)$, the first case can be represented by $vote_p$, while the second case can be represented by $vote_p = \perp$, where $\perp$ is a special value different from all "normal" values of $vote_p$.

DLS algorithms [6] There are three consensus algorithms in [6]: one for benign faults (requires $n > 2f$) one for authenticated Byzantine faults ($n > 3b$) and one for Byzantine faults (also $n > 3b$). Let us denote these three algo-

[25] The reliable channel assumption can be expressed in the round model by the following predicate: $\mathcal{P}_{rel}(r) \equiv \forall p \in \mathcal{C} : |\{m \in \vec{\mu}_p^r : m \neq \perp\}| \geq n - f$.

[26] In MR algorithm the functionalities of the selection and decision rounds are provided in the same round (see Optimization (iv)).

rithms by b-DLS (*benign*), a-DLS (*authenticated*) and B-DLS (*Byzantine*) and let us concentrate only on the former and the latter. The algorithm B-DLS belongs to class 3, and b-DLS to class 2. As both algorithms are based on the same mechanisms, we discuss only b-DLS in more details since it is simpler.

Let us consider an instantiation of Algorithm 4, with $T_D = f+1$, $FLAG = \phi$ and $Validator(p, \phi)$ implementation based on the rotating coordinator paradigm and Algorithm 12 as the instantiation of the FLV function. We call this algorithm Inst-b-DLS.

We now compare Inst-b-DLS with b-DLS. The validation and the decision condition are the same in both algorithms. Although the selection conditions are the same, there is a small difference in how the selection logic is executed. Namely, b-DLS algorithm relies on a mechanism called *locking* (which is different from the notion of locking used in this paper): Whenever $vote_p = v$ with v different from a special value $\mathcal{A}$, then p has *locked* value v; If $vote_p = \mathcal{A}$ (special value) then p has not locked any value. Furthermore, b-DLS relies on lock-release mechanism that takes place in the additional round (called lock-release round), which is executed immediately before the selection round of the next phase, in which processes exchange messages ($vote$ and ts) to possibly unlock locked values, i.e., reset $vote_p$ to $\mathcal{A}$. When $\mathcal{P}_{good}$ holds in the lock-release round, then at most one value is locked.

By contrast, in Inst-b-DLS the lock-release mechanism takes place inside the FLV function, i.e., there is no need for the additional round. Lines 1-4 of Algorithm 12 correspond to the lock-release mechanism and they ensure that there is at most one value locked in the set V_p once $\mathcal{P}_{good}$ holds.

Algorithm 12 Instantiation of FLV function based on [6]

1: $V_p \leftarrow \vec{\mu}_p^r$
2: **for** $i = 1$ to n **do**
3:    **if** $\exists(vote, ts) \in \vec{\mu}_p^r \; s.t. \; ts > V_p[i].ts$ **then**
4:       $V_p[i] \leftarrow (\mathcal{A}, -)$
5: $possibleVotes_p \leftarrow \{ (vote, -) \in V_p :$
      $|\{(vote', -) \in V_p^r : \; vote = vote' \vee vote' = \mathcal{A}\}| \geq n - f$
6: **if** $|possibleVotes_p| = 1$ **and** $\mathcal{A} \notin possibleVotes_p$ **then**
7:    **return** $v \; s.t. \; (v, -) \in possibleVotes_p$
8: **else if** $|possibleVotes_p| > 0$ **then**
9:    **return** $?$
10: **else**
11:    **return** $null$

MQB MQB is a *new* Byzantine consensus algorithm that requires $n > 4b$. It is obtained by instantiation of Algorithm 4 with $FLAG = \phi$ and $Validator(p, \phi)$ returning always Π; this corresponds to Algorithm 3. We consider the FLV instantiation given by Algorithm 6 and $T_D = \lceil \frac{n+2b+1}{2} \rceil$.[27]

Compared to PBFT, MQB has the advantage not to need the (unbounded) variable $history_p$, at the cost of requiring $n > 4b$ instead of $n > 3b$ (for PBFT).

7.3 Class 3 algorithms

PBFT [3] PBFT is an algorithm that solves a sequence of instances of consensus (in the context of state machine replication). We consider here the instantiation of a single instance of consensus that represents the "core" of the PBFT algorithm. PBFT is designed for Byzantine faults ($f = 0$) and requires $n > 3b$.

We get Inst-PBFT from Algorithm 4 with the following parametrization: $T_D = 2b + 1$, $FLAG = \phi$, $Validator(p, \phi) = \Pi$ and Algorithm 13 as the FLV instantiation. We also set $n = 3b + 1$, as in PBFT.

Furthermore, since PBFT does not consider the Unanimity property, we do not consider it with Inst-PBFT. This allows us to get the FLV Algorithm 13 from Algorithm 7. Indeed, in this case, lines 8-9 of Algorithm 7 can be removed, and the conditions of line 5 and line 7 of Algorithm 7 can be merged into line 5 of Algorithm 13.

Algorithm 13 FLV for class 3 with $T_D = 2b + 1$ and $n = 3b + 1$

1: $possibleVotes_p \leftarrow \{ (vote, ts, -) \in \vec{\mu}_p^r :$
      $|\{(vote', ts', -) \in \vec{\mu}_p^r : \; vote = vote' \vee ts > ts'\}| > 2b$
2: $correctVotes_p \leftarrow \{v : (v, ts, -) \in possibleVotes_p \wedge$
      $|\{(vote', ts', history') \in \vec{\mu}_p^r : \; (v, ts) \in history'\}| > b \}$
3: **if** $|correctVotes_p| = 1$ **then**
4:    **return** $v \; s.t. \; (v, -, -) \in correctVotes_p$
5: **else if** $|correctVotes_p| > 1$ **or**
   $| \left\{ (vote, ts, -) \in \vec{\mu}_p^r : \; ts = 0 \right\} | > 2b$ **then**
6:    **return** $?$
7: **else**
8:    **return** $null$

We now compare Inst-PBFT with PBFT. The validation and decision rounds are the same in both algorithms. There is a small difference in the selection condition of the selection round: whenever Inst-PBFT selects any value using some deterministic function (see line 11 of Algorithm 4), PBFT selects a special "null" value. Therefore, in PFBT the decision can be on a special "null" value, while in Inst-PBFT the decision is always on a "real" value.

Since the Unanimity property is not considered, we can apply optimization (iii) (Sect. 4.6) to the selection round of Inst-PBFT. With this modification, the first phase of Inst-PBFT corresponds to the "common case" and all later phases correspond to the "view change protocol" of PBFT.

[27] Same argument as in footnote 20: same value for T_D in the decision round and in the FLV function.

8 Conclusion

The paper has presented a generic consensus that highlights the core mechanisms of various consensus algorithms for benign and Byzantine faults. The generic algorithm has four parameters: T_D, $FLAG$, $Validator$ and FLV. Instantiation of these parameters led us to distinguish three classes of consensus algorithms into which well-known algorithms fit. It allowed us also to identify the new MQB algorithm. We believe that our classification should contribute to a better understanding of the jungle of consensus algorithms.

References

[1] F. Borran and A. Schiper. A Leader-Free Byzantine Consensus Algorithm. In *ICDCN*, pages 67–78, 2010.

[2] M. Castro. Practical Byzantine fault-tolerance. PhD thesis. Technical report, MIT, 2000.

[3] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, Nov. 2002.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *JACM*, 43(2):225–267, Mar. 1996.

[5] B. Charron-Bost and A. Schiper. The Heard-Of model: computing in distributed systems with benign failures. *Distributed Computing*, 22(1):49–71, 2009.

[6] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[7] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[8] R. Friedman, A. Mostéfaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *TDSC*, 2(1):46–56, 2005.

[9] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Trans. Comput.*, 53(4):453–466, Apr. 2004.

[10] R. Guerraoui and M. Raynal. The alpha of indulgent consensus. *Comput. J.*, 50(1):53–67, Jan. 2007.

[11] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[12] L. Lamport. Byzantizing Paxos by refinement. In *DISC*, pages 211–224, 2011.

[13] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[14] B. Lampson. The abcd's of paxos. In *PODC*, pages 13–, 2001.

[15] H. C. Li, A. Clement, A. S. Aiyer, and L. Alvisi. The paxos register. In *SRDS*, pages 114–126, 2007.

[16] D. Malkhi and M. Reiter. Byzantine quorum systems. In *STOC*, pages 569–578, 1997.

[17] J.-P. Martin and L. Alvisi. Fast byzantine consensus. In *IEEE Transactions On Dependable And Secure Computing*, pages 402–411, 2005.

[18] Z. Milosevic, M. Hutle, and A. Schiper. Unifying Byzantine consensus algorithms with Weak Interactive Consistency. In *OPODIS*, pages 300–314, 2009.

[19] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. A versatile and modular consensus protoco. In *DSN*, pages 364–373, 2002.

[20] A. Mostfaoui and M. Raynal. Solving consensus using chandra-touegs unreliable failure detectors: A general quorum-based approach. *Distributed Computing*, 1693:847–847, 1999.

[21] Y. J. Song, R. van Renesse, F. B. Schneider, and D. Dolev. The building blocks of consensus. In *ICDCN*, pages 54–72, 2008.