Generic Construction of Consensus Algorithms for Benign and Byzantine Faults

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

Abstract

The paper proposes a generic consensus algorithm that highlights the basic and common features of known consensus algorithms. The parameters of the generic algorithm encapsulate the core differences between various consensus algorithms, including leader-based and leader-free algorithms, addressing benign faults, authenticated Byzantine faults and Byzantine faults. This leads to the identification of three classes of consensus algorithms. With the proposed classification, Paxos and PBFT indeed belong to the same class, while FaB Paxos belongs to a different class. Interestingly, the classification allowed us to identify a new Byzantine consensus algorithm that requires \( n > 4b \), where \( b \) is the maximum number of Byzantine processes.

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. Considering these numerous algorithms, it would be helpful to classify them, in order to identify the basic mechanisms on which they rely. This would allow a better understanding of consensus algorithms, particularly for a classification encompassing benign faults and malicious (Byzantine) faults.

The paper provides such a classification by proposing a generic consensus algorithm, which highlights the basic and common features of known consensus algorithms. The parameters of the generic algorithm encapsulate the core differences between various consensus algorithms, including leader-based and leader-free algorithms, addressing benign faults, authenticated Byzantine faults and Byzantine faults. Instantiations of the parameters allow us to obtain these various algorithms. The generic algorithm also allows us to discuss randomized consensus algorithms.

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 class in two, based on the state variables required. This lead us to identify three classes of consensus algorithms, and tradeoffs between these classes. With this classification, Paxos [11] (benign faults) and PBFT [4] (Byzantine faults) indeed belong to the same class, while FaB Paxos [16] belongs to a different class. Interestingly, the classification 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).

Our generic algorithm is based on four parameters: the \( FLV \) function, the \( \text{Selector} \) function, the threshold parameter \( T_D \), and the flag \( \text{FLAG} \). The functions \( FLV \) and \( \text{Selector} \) 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 can 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 the corresponding functions.

The paper is not the first one to propose a generic consensus algorithm, but it goes beyond previous approaches. Mostéfaoui et al. [18] 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 algorithm, where generality is encapsulated in a function called \( \Lambda \). The \( \Lambda \) function encapsulates both our selection and our validation rounds. This does not allow the authors of [9] to identify the differences between two of our three classes of consensus algorithms. Moreover, as for [18], the paper is restricted to benign faults. Later, Guerraoui and Raynal [10] propose a generic version of Paxos in

\[ b \] is the maximum number of Byzantine processes.
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., [13] and [14]. Recently, Song et al. [20] proposed building blocks that allow the construction of consensus algorithms. They consider both benign and Byzantine faults. However, they ignore some seminal consensus algorithms such as PBFT and FaB Paxos, and their framework therefore has a somehow limited scope.

The rest of the paper is organized as follows. Section 2 is devoted to the system model and to definitions. Section 3 introduces the generic consensus algorithm and its parameters. Section 4 presents three classes of instantiations for these parameters, and classifies consensus algorithms such as Paxos, FaB Paxos, PBFT into these classes. Section 5 gives examples of instantiations of the generic algorithm. In Section 6 we show how the generic algorithm can be adapted to include randomized consensus algorithms, and Section 7 concludes the paper.

2 Model and Definitions

2.1 System Model

We consider a variant of a partially synchronous system [7]: we assume that the system alternates between good periods (during which the system is synchronous) and bad periods (during which the system is asynchronous). We differentiate honest processes that execute algorithms faithfully, from Byzantine processes [12], 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 all processes is denoted by \( \Pi \), the set of honest processes by \( \mathcal{H} \) and the set of correct processes by \( \mathcal{C} \).

**Round Model.** Distributed algorithms can be expressed as a sequence of rounds. In each round \( r \), a process \( p \) sends a message to a subset of processes according to a “sending” function \( S_p^r \), and at the end of this round, 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).

Honest processes cannot be impersonated: if an honest process receives \( v \) from \( p \) in round \( r \), and \( p \) is honest, then \( p \) sent \( v \) in round \( r \). The state of process \( p \) in round \( r \) is denoted by \( s_p^r \); the message sent by an honest process is denoted by \( S_p^r(s_p^r) \); messages received by process \( p \) in round \( r \) are denoted by \( \mu_p^r \) (\( \mu_p^r[q] \) is the message received from \( q \)).

**Communication Predicates.** During good periods of our partially synchronous system, we assume the following two communication predicates that are sufficient to solve consensus: \( \mathcal{P}_{\text{good}} \) and \( \mathcal{P}_{\text{cons}} \). The predicate \( \mathcal{P}_{\text{good}} \) ensures that correct processes receive every message sent by a correct process:

\[
\mathcal{P}_{\text{good}}(r) \equiv \forall p, q \in \mathcal{C} : \mu_p^r[q] = S_q^r(s_q^r)
\]

An implementation of \( \mathcal{P}_{\text{good}} \) on top of the basic partially synchronous system model with benign and Byzantine faults has been proposed in [7].

The predicate \( \mathcal{P}_{\text{cons}} \) provides the same guarantees as the predicate \( \mathcal{P}_{\text{good}} \), but additionally ensures that each correct process receives the same set of messages. In the benign fault model (i.e., \( b = 0 \)), this predicate can be implemented using the implementation of \( \mathcal{P}_{\text{good}} \) described in [7] if we assume that no crash occurs in good periods. In the Byzantine fault model (i.e., \( b \neq 0 \)), several implementations of \( \mathcal{P}_{\text{cons}} \) have been proposed [17, 2]:

\[
\mathcal{P}_{\text{cons}}(r) \equiv \mathcal{P}_{\text{good}}(r) \land \forall p, q \in \mathcal{C} : \mu_p^r = \mu_q^r
\]

Based on these definitions, we define the notion of a good phase. A phase is a sequence of rounds. A good phase \( \phi \) of \( k \) rounds is defined as a phase such that \( \mathcal{P}_{\text{cons}} \) holds in the first round, and \( \mathcal{P}_{\text{good}} \) holds in the remaining \( k-1 \) rounds.

2.2 Unifying Byzantine Faults

Two different models for Byzantine faults have been considered in literature [7]: (1) authenticated Byzantine faults, where messages can be signed by the sending process (with the assumption that signatures cannot be forged by any other process), and (2) Byzantine faults, where there is no mechanism for signatures (but the receiver of a message knows the identity of the sender).³

As shown in [17], the predicate \( \mathcal{P}_{\text{cons}} \) allows the unification of these two fault models: (i) \( \mathcal{P}_{\text{cons}} \) allows us to express a generic consensus algorithm that is the same for both fault models, and (ii) \( \mathcal{P}_{\text{cons}} \) can be implemented out of \( \mathcal{P}_{\text{good}} \) in the two fault models. The implementation in the authenticated Byzantine fault model is simpler and requires two rounds; three rounds are needed in the Byzantine fault model. To summarize, the predicate \( \mathcal{P}_{\text{cons}} \) allows us to describe consensus algorithms without making difference between authenticated Byzantine faults and Byzantine faults. Therefore, in the paper we use the term Byzantine faults for both fault models, except if explicitly mentioned.

³In [12], these models are respectively called (1) Byzantine faults with signed messages, and (2) Byzantine faults with oral messages.
2.3 The Consensus Problem

In the consensus problem, each process starts with a given initial value, and later possibly decides on a value. The problem is 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** [20]: 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.

**Locked Value.** In the context of a consensus algorithm, we refer below to the notion of locked value.\(^4\) This notion has similarities with the notion of univalent configuration defined in [8], but is actually different (see below). 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 must 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 $r'$, then $v$ is not locked in round $r$, but only in round $r' + 1 > r$).

3 A Generic Consensus Algorithm

We now present a generic algorithm (see Algorithm 1), from which well-known consensus algorithms can be instantiated (see Section 5). Generality is obtained by parametrization of Algorithm 1: parameters appear in a box.

3.1 Generic Algorithm

The generic Algorithm 1 consists of a sequence of phases that can be seen as successive trials to decide on a value. Each phase $\phi$ consists of three rounds, respectively called selection round ($r = 3\phi - 2$), validation round ($r = 3\phi - 1$), and decision round ($r = 3\phi$). We will see that some values of the parameters allow us to skip the validation round. We first describe process states, and then the three rounds.

\(^4\)Note that the definition of the term locked value in some other works differs from our definition.

\(^5\)Bounding the size of the variable $history_p$ requires an additional round of communication. More details can be found in [3].
Algorithm 1 Generic Algorithm

1: Initialization:
2: \( \text{vote}_p := \text{init}_p \)
3: \( \text{ts}_p := 0 \)
4: \( \text{history}_p := \{ (\text{init}_p, 0) \} \)

5: Selection Round \( r = 3\phi - 2 \):
6: \( S^r_p \):
7: \( \text{send} \langle \text{vote}_p, \text{ts}_p, \text{history}_p, \text{Selector}(p, \phi) \rangle \) to \( \text{Selector}(p, \phi) \)
8: \( T^r_p \):
9: \( \text{select}_p \leftarrow \text{FLV}(\mu^r_p) \)
10: if \( \text{select}_p = ? \) then
11: \( \text{select}_p \leftarrow \) choose deterministically a value in \( \{ v : (v, -, -) \in \mu^r_p \} \)
12: if \( \text{select}_p \neq \text{null} \) then
13: \( \text{vote}_p := \text{select}_p \)
14: \( \text{history}_p := \text{history}_p \cup \{ (\text{vote}_p, \phi) \} \)
15: \( \text{validators}_p \leftarrow S \) if there exists the set \( S \) such that more than \( \frac{n+b}{2} \) messages \( (\cdot, -, S) \) have been received else \( \emptyset \)

16: Validation Round \( r = 3\phi - 1 \):
17: \( S^r_p \):
18: if \( p \in \text{validators}_p \) then
19: \( \text{send} \langle \text{select}_p, \text{validators}_p \rangle \) to all
20: \( T^r_p \):
21: \( \text{validators}_p \leftarrow S \) if there exists the set \( S \) such that \( b+1 \) messages \( (\cdot, S) \) have been received else \( \emptyset \)
22: if there is a value \( v \) such that \( \{ q \in \text{validators}_p : \mu^r_p[q] = (v, \cdot) \} \) > \( \frac{|\text{validators}_p|+b}{2} \) then
23: \( \text{vote}_p := v \)
24: \( \text{ts}_p := \phi \)
25: else
26: \( \text{vote}_p := v \) such that \( (v, \text{ts}_p) \in \text{history}_p \)

27: Decision Round \( r = 3\phi \):
28: \( S^r_p \):
29: \( \text{send} \langle \text{vote}_p, \text{ts}_p \rangle \) to all
30: \( T^r_p \):
31: if received at least \( T_p \) messages with the same value \( \langle v, \text{FLAG} \rangle \) then
32: \( \text{DECIDE} v \)

\( P_{cons} \) holds, since all correct processes execute the function \( \text{FLV}(\mu^r_p) \) on the same set of messages. 

Optimization: The selection round can be suppressed in the first phase. This requires to initialize the variable \( \text{select}_p \) with \( \text{init}_p \), and \( \text{validators}_p \) with the same set \( S \) on each process \( p \). Note that it is safe to select \( \text{init}_p \) at the first round for the following reason. If no value is initially locked, then any value may be selected by honest selectors. If some value \( v \) is initially locked, then by definition all honest selectors have \( \text{init}_p = v \), and all honest selectors select \( v \).

The Validation Round \( r = 3\phi - 1 \). The role of this round is for every honest process \( p \) to determine which value selected by validators in the selection round is a valid value. Among all honest processes, at most one value may be considered to be valid.

The validation round is executed as follows. Each validator first sends the value selected in the selection round together with the set of validators (line 19). Then, each honest process \( p \) computes the set of validators as follows: \( \text{validators}_p \) is the set \( S \) such that \( p \) receives \( b+1 \) messages \( (\cdot, S) \), or \( \emptyset \) if no set satisfies this condition (line 21). Based on this set, each process tries to determine a valid value \( v \). If it observes that a majority of correct validators have selected the same value \( v \), then \( v \) is a valid value. In that case a process \( p \) sets its vote \( \text{vote}_p \) to \( v \), and updates its timestamp \( \text{ts}_p \) to the current phase \( \phi \) (lines 22-24). Otherwise, the vote is reverted to the value corresponding to \( \text{ts}_p \) (line 26).7

Optimizations: If the function \( \text{Selector}(p, \phi) \) returns the same set of processes on each process \( p \) and in all phases \( \phi \), then \( \text{validators}_p \) at line 15 can be set to \( \text{Selector}(p, \phi) \), and line 21 can be suppressed. As a result, the sets \( \text{Selector}(p, \phi) \) and \( \text{validators}_p \) do not need to be sent respectively at lines 7 and 19.

The same holds with benign faults when \( |\text{Selector}(p, \phi)| = 1 \) on each process \( p \) and in all phases \( \phi \). \( \text{validator}_p \) at line 21 is the process from which the validation round message is received.

6\( P_{cons} \) is defined in [17] for rounds in which all processes send to all processes. We assume here variant of \( P_{cons} \) that does not require all-to-all message exchange. The adaptation is trivial.

7Line 26 is not mandatory, but it allows us to simplify the instantiation of function \( \text{FLV}(\mu^r_p) \).
The Decision Round \((r = 3\phi)\). The decision round determines the conditions that must hold for a process to decide. Concretely, each process starts by sending its vote and its timestamp (line 29). A process then decides if it receives a threshold number \(T_D\) of identical votes which satisfy some criteria defined by the flag \(FLAG\). To our knowledge, only two criteria have been considered in literature: (1) \(FLAG = \phi\): only votes that have been validated in the current phase \(\phi\) are considered, and (2) \(FLAG = *\): all votes are considered. In the latter case, the validation round can be suppressed. As a consequence, variables \(tsp\) and \(history_p\) are no more necessary. Moreover, the set \(Selector(p, \phi)\) does not need to be sent at line 7, and line 15 can be suppressed.

Optimization: The decision round of phase \(\phi\) can be executed concurrently with the selection round of phase \(\phi + 1\).

3.2 Parameters

We identify two categories of parameters. The first category is related to the decision round, and contains the parameters \(T_D\) and \(FLAG\). As shown in Section 4, these two parameters influence the properties of the instantiated algorithm (i.e., \(n\), process state, and number of rounds per phase). The second category contains the functions \(Selector(p, \phi)\) and \(FLV(\mu_p)\) which define the selection and the validation rounds.

\(FLAG\): The parameter \(FLAG\) defines which votes are taken into account in the decision round: all votes (if \(FLAG = *\)), or only the votes that are valid in the current phase (if \(FLAG = \phi\)). In the former case, the validation round can be suppressed.

\(T_D\): The parameter \(T_D\) defines the number of identical votes that is required to decide. To ensure termination, the votes of faulty (honest) and Byzantine processes must not be required to decide. Hence, \(T_D \leq n - b - f\).

\(Selector(p, \phi)\): The function \(Selector(p, \phi)\)\(^8\) returns a set of processes \(S \subseteq P\) that represents \(p\)'s suggestion for the set \(validators\) in phase \(\phi\). It must satisfy the following two properties:

- **Selector-validity**: If \(|Selector(p, \phi)| > 0\), then \(|Selector(p, \phi)| > b\).
- **Selector-liveness**: There exists a good phase \(\phi_0\) such that:
  - \(SL1\): \(\forall p, q \in C: Selector(p, \phi_0) = Selector(q, \phi_0)\).
  - \(SL2\): if \(FLAG = *\), then \(|Selector(p, \phi_0) \cap C| \geq T_D\).
  - \(SL3\): if \(FLAG = \phi\), then \(|Selector(p, \phi_0) \cap C| > \frac{|Selector(p, \phi_0)| + b}{2}\).

\(FLV(\mu_p)\): The function \(FLV(\mu_p)\) must satisfy the following three properties:

- **FLV-validity**: If \(FLV(\mu_p)\) returns \(v\) such that \(v \neq ?\) and \(v \neq null\), then \(v \in \{vote, (vote, -, -)\} \in \mu_p\).
- **FLV-agreement**: If value \(v\) is locked in round \(r\), only \(v\) or \(null\) can be returned.
- **FLV-liveness**: If \(\forall q \in C : \mu_p[q] \neq \perp\), then \(null\) cannot be returned.

3.3 Correctness of the Generic Algorithm

Correctness of our generic algorithm is based on the following two lemmas from which Theorem 1 can be proved. All proofs can be found in [19].

**Lemma 1.** If \(Selector\)-validity holds, then the following property holds on every honest process \(h\) and in every phase \(\phi\): if process \(h\) set \(v_{eq}\) to \(v\) and \(ts_h\) to \(\phi\) at lines 23-24, then at least one honest process has sent \((v, -)\) at line 19.

**Lemma 2.** In every phase \(\phi\), if (i) \(Selector\)-validity holds, (ii) an honest process \(p\) updates \(vote_p\) to \(v\) and \(tsp\) to \(\phi\), and (iii) another honest process \(q\) updates \(vote_q\) to \(v'\) and \(tsp\) to \(\phi\) (lines 23-24), then \(v = v'\).

**Theorem 1.** If (i) function \(FLV(\mu_p)\) satisfies \(FLV\)-validity and \(FLV\)-agreement, (ii) function \(Selector(p, \phi)\) satisfies \(Selector\)-validity, (iii-a) \(FLAG = \phi\) and \(T_D > b\), or (iii-b) \(FLAG = *\) and \(T_D > \frac{n+b}{3}\), then Algorithm 1 ensures validity, unanimity and agreement.

Termination holds if (iv) \(T_D \leq n - b - f\), (v) function \(FLV(\mu_p)\) satisfies \(FLV\)-liveness, and (vi) there is a good phase \(\phi_0\) in which \(Selector\)-liveness holds (SL1, SL2, SL3).

4 Instantiations of the Parameters and Classification

We present now instantiations of the functions \(FLV\) and \(Selector\). We identify three instantiations of \(FLV\) function. The first instantiation uses only the variable \(vote_p\), the second uses the variables \(vote_p\) and \(tsp\), and the last one uses all three variables \(vote_p\), \(tsp\) 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 (follows from the value of \(FLAG\), see Section 3.2).

One can observe a tradeoff among these three classes. For instance, when \(FLAG = *\), \(T_D > \frac{n+3b+f}{2}\), only two rounds per phase are needed and the process state is the
smallest, but it requires the largest \( n (n > 5b + 3f) \). The “Examples” column of Table 1 shows which known algorithms correspond to a given class. These examples are discussed in Section 5.

We can make the following comments. First, if \( b = 0 \) (benign faults), classes 2 and 3 are identical, since \( \text{history}_p \) can be ignored with benign faults. Therefore Paxos and CT\(^9\), which belong to class 2, also trivially belong to class 3, case \( b = 0 \). Second, to the best of our knowledge, no existing algorithm corresponds to the class 2 for the case \( f = 0 \) (Byzantine faults). We call this new algorithm MQB (Masking Quorum Byzantine consensus algorithm).\(^{10}\) Finally, 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 \( \text{Selector} \) function are discussed later.

4.1 Instantiations of \( \text{FLV}(\vec{\mu}_p) \)

We give here the intuition of the instantiations. The proofs that the properties defined in Section 3.2 hold can be found in [19].

4.1.1 \( \text{FLV}(\vec{\mu}_p) \) for class 1

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

Line 1 is for \( \text{FLV} \)-agreement. We now explain its role with a simple example. Let \( v_1 \) be locked in round \( r \). For simplicity, let us reason only on the following case: some honest process \( p \) has decided \( v_1 \) in round \( r - 1 \). By Algorithm 1, \( p \) 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., at most \( n - (T_D - b) \) processes have \( \text{vote}_p \).

\(^9\)CT refers to the Chandra-Toueg consensus algorithm with the failure detector \( \mathcal{OS} \).

\(^{10}\)The quorums used in this algorithm satisfy the property of masking quorums [15]. Note that with respect to the definitions in [15], algorithms of class 1 use opaque quorums, and algorithms of class 3 use dissemination quorums.

Table 1. The three classes of consensus algorithms.

<table>
<thead>
<tr>
<th>( \text{FLAG} )</th>
<th>( T_D )</th>
<th>( n )</th>
<th>Process state</th>
<th>Rounds per phase</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>&gt; ( \frac{n + 3b + f}{2} )</td>
<td>&gt; 5b + 3f</td>
<td>(( \text{vote}_p ))</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>( \phi )</td>
<td>&gt; 3b + f</td>
<td>&gt; 4b + 2f</td>
<td>(( \text{vote}_p, \text{ts}_p ))</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>( \phi )</td>
<td>&gt; 2b + f</td>
<td>&gt; 3b + 2f</td>
<td>(( \text{vote}_p, \text{ts}_p, \text{history}_p ))</td>
<td>3</td>
</tr>
</tbody>
</table>

Algorithm 2 \( \text{FLV}(\vec{\mu}_p) \) for class 1

1: \( \text{correctVotes}_p \leftarrow \{ v | \{ (v, -, -) \in \vec{\mu}_p \} | > n - T_D + b \} \)
2: if \( |\text{correctVotes}_p| = 1 \) then
3: \( \text{return } v \in \text{correctVotes}_p \)
4: else if \( |\vec{\mu}_p| > 2(n - T_D + b) \) then
5: \( \text{return } ? \)
6: else
7: \( \text{return } \text{null} \)

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 \( \text{null} \), \( \text{FLV} \) can only return \( v_1 \). For \( \text{FLV} \)-agreement to hold, Algorithm 3 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 \( \vec{\mu}_p \) contains more than \( 2(n - T_D + b) \) messages. With (*), any set of more than \( 2(n - T_D + b) \) messages contains more than \( n - T_D + b \) messages equal to \( v_1 \) (this is illustrated in Figure 1 with the case \( n = 6, b = 1, f = 0 \) and \( T_D = 5 \)). By line 1, we have \( v_1 \in \text{correctVotes}_p \), and as explained above, only \( v_1 \) can be in \( \text{correctVotes}_p \). Therefore, the condition of line 2 holds: Algorithm 2 cannot return \( ? \) when \( v_1 \) is locked.

Figure 1. Illustration for \( \text{FLV} \) for class 1 \((n = 6, b = 1, f = 0, T_D = 5)\)

Property \( \text{FLV} \)-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 a message from all correct processes (i.e., \(|\vec{\mu}_p| \geq n - b - f \)) implies that the condition of line 4 holds. Property \( \text{FLV} \)-validity is ensured by lines 1-3.
4.1.2 FLV($\vec{\mu}_p$) for class 2

The FLV function for class 2 ($\text{FLAG} = \phi$ and $T_D > 3b + f$) is shown in Algorithm 3, where \{# \ldots #\} at line 1 denotes a multiset. Note that when $T_D \leq \frac{n + b + f}{2}$ which can be the case for instantiations of classes 2 and 3, detecting the locked value only based on votes, as done by Algorithm 2, does not work. Therefore, an additional mechanism is needed: timestamps.

**Algorithm 3 FLV($\vec{\mu}_p$) for class 2**

```plaintext
1: possibleVotes_p ← \{# \{vote, ts, −, −\} ∈ \vec{\mu}_p : |
| (\{vote', ts', −, −\} ∈ \vec{\mu}_p : vote = vote' ∧ ts > ts') |
| > n − T_D + b #\}
2: correctVotes_p ← \{(vote, −, −, −) ∈ possibleVotes_p :
| (\{vote', −, −, −\} ∈ possibleVotes_p : vote = vote') > b \}
3: if |correctVotes_p| = 1 then
4: return v s.t. (v, −, −, −) ∈ correctVotes_p
5: else if |correctVotes_p| > n − T_D + 2b then
6: return \vec{\mu}
7: else
8: return null
```

Lines 1 and 2 are for FLV-agreement. We now explain their role with a simple example. Let $v_1$ be locked in round $r$ that belongs to phase $\phi_1 + 1$. For simplicity, let us reason only on the following case: some honest process $p$ has decided $v_1$ in round $r − 1$ that belongs to phase $\phi_1$. By Algorithm 1, $p$ has received $T_D$ messages $\langle v_1, \phi_1 \rangle$ in the decision round $r − 1$. Therefore, at least $T_D − b$ honest processes have $\text{vote}_p = v_1$ and $\text{ts}_p = \phi_1$, i.e., at most $n − T_D$ honest processes have $\text{vote}_p = v_2 \neq v_1$. Because only one value can be validated by honest processes in phase $\phi_1$ (see Lemma 2), all honest processes with $\text{vote}_p = v_2 \neq v_1$ have $\text{ts}_p < \phi_1$. It follows that for every honest process $p$, we have $\text{vote}_p = v_1$ or $\text{ts}_p < \phi_1$. Together with $\phi_1$, 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 $\text{possibleVotes}_p$ may contain at most $b$ messages $\langle v_2 \neq v_1, −, −, −\rangle$, i.e., the messages sent by Byzantine processes. Line 2 prevents such messages to be in $\text{correctVotes}_p$. This shows that among the values different from $\phi_1$ and null, only $v_1$ can be returned.

![Diagram](image)

**Figure 2. Illustration for FLV for class 2 ($n = 5$, $b = 1$, $f = 0$, $T_D = 4$)**

For FLV-agreement to hold, Algorithm 3 must also prevent $\phi_1$ 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 $\vec{\mu}_p$ contains more than $n − T_D + 2b$ messages. With $\phi_1$, the set $\vec{\mu}_p$ contains at least $b + 1$ messages $\langle v_1, \phi_1, −, −\rangle$ from honest processes (this is illustrated in Figure 2 for the case $n = 5$, $b = 1$, $f = 0$, $T_D = 4$). With $\phi_1$ and the fact that $\vec{\mu}_p$ contains more than $n − T_D + 2b$ messages from honest processes (see Figure 2), 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 $\text{correctVotes}_p$. Moreover, as discussed above, only $v_1$ can be in $\text{correctVotes}_p$. Therefore, the condition of line 3 holds: Algorithm 3 cannot return $\phi_1$ 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., $|\vec{\mu}_p| \geq n − b − f$) ensures that the condition of line 5 holds. Property FLV-validity is ensured by lines 1-4.

4.1.3 FLV($\vec{\mu}_p$) for class 3

The FLV function for class 3 ($\text{FLAG} = \phi$ and $T_D > 2b + f$) is shown in Algorithm 4. Observe that for instantiations of class 3, $T_D$ can be $\leq 2b + f$. Therefore, detecting the locked value only based on votes and timestamps, as done by Algorithm 3, does not work. Therefore, an additional mechanism is needed: the history log.

**Algorithm 4 FLV($\vec{\mu}_p$) for class 3**

```plaintext
1: possibleVotes_p ← \{# \{vote, ts, −, −\} ∈ \vec{\mu}_p :
| (\{vote', ts', −, −\} ∈ \vec{\mu}_p : vote = vote' ∧ ts > ts') |
| > n − T_D + b \}
2: correctVotes_p ← \{v : (v, ts, −, −) ∈ possibleVotes_p ∧ |
| (\{vote', ts', history, −\} ∈ \vec{\mu}_p : (v, ts, history) > b \}
3: if |correctVotes_p| = 1 then
4: return v s.t. (v, ts, −, −) ∈ correctVotes_p
5: else if |correctVotes_p| > n − T_D + b then
6: return \vec{\mu}
7: else if \{\langle vote, ts, −, −\rangle : ts = 0 \} > n − T_D + b then
8: if there is a value $v$ such that $\vec{\mu}_p$ contains a majority of messages $\langle v, −, −\rangle$ then
9: return $v$
10: return \vec{\mu}
11: else
12: return null
13: return null
```

Similarly to Algorithm 3, lines 1 and 2 are for FLV-agreement. Their role can be explained with a simple example. Consider that value $v_1$ is locked in round $r$ that belongs to phase $\phi_1 + 1$. For simplicity, let us first assume that some honest process $p$ has decided $v_1$ in round $r − 1$ that belongs to phase $\phi_1$. Consider Figure 3. For the same reason as for Algorithm 3, at least $T_D − b$ honest processes have $\text{vote}_p = v_1$ and $\text{ts}_p = \phi_1$ (i.e., at most $n − T_D$ honest processes have $\text{vote}_p = v_2 \neq v_1$). Furthermore, for every honest process $p$, we have $\text{vote}_p = v_1$ or $\text{ts}_p < \phi_1$. Together with $\phi_1$, 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 \( \langle v_1, -,-,- \rangle \), the messages \( \langle v_2 \neq v_1, \phi_2, -,- \rangle \) sent by Byzantine processes can be in the set \( \text{possibleVotes}_p \). Because honest processes can only update history at line 14 of Algorithm 1, 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 \( \text{correctVotes}_p \) 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 \( \text{vote}_p = v_1 \). With the same arguments as above, it follows that no value different from \( v_1 \) or \( \text{null} \) can be returned at lines 4 and 6. However, the condition of line 7 might hold. In this case, \( \mu^p \) contains more than \( n - T_D \) messages \( \langle v_1, 0, -,- \rangle \) from honest processes, and at most \( n \) messages \( \langle v_2 \neq v_1, 0, -,- \rangle \) from Byzantine processes. Because \( T_D \leq n - b - f \), we have \( n - T_D > b \), 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 \( \text{Selector}-\text{validity}^{11} \).

- **Selector-strongValidity**: If \( |\text{Selector}(p, \phi)| > 0 \), then \( |\text{Selector}(p, \phi)| > 3b + 2f \).

This requirement can be explained as follows. Let \( \mu^p \) contains the messages from all the \( n - b - f \) correct processes. There are two cases to consider: (1) correct processes sent only \( (-,0,-,-) \), (2) at least one correct process sent \( (-, ts > 0, -,-) \). 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 \( \text{null} \) cannot be returned at line 13. In case (2), let \( \nu \) denote the subset of messages in \( \mu^p \) that are from correct processes, and let \( ts_\nu \) be the highest timestamp in \( \nu \). By Lemma 2 and Algorithm 1, there is a unique value \( v_\nu \) such that \( \langle v_\nu, ts_\nu, -,- \rangle \in \nu \). Together with (*), this ensures that the set \( \text{possibleVotes}_p \) is non-empty, and contains \( \langle v_\nu, ts_\nu, -,- \rangle \). The **Selector-strongValidity** allows us to get a stronger variant of Lemma 1: it ensures that if process \( h \) set \( \text{vote}_h \) to \( v \) and \( ts_h \) to \( \phi \) at lines 23-24, then at least \( b + 1 \) correct processes have sent \( \langle v, -\rangle \) at line 19. As a result, any correct process that validates \( v_\nu \) in the validation round \( 3ts_\nu - 1 \) received \( v_\nu \) from at least \( b + 1 \) correct processes. Therefore, at least \( b + 1 \) correct processes have selected \( v_\nu \) in round \( 3ts_\nu - 2 \), and these processes have \( \langle v_\nu, ts_\nu \rangle \) is their history. This implies that the set \( \text{correctVotes}_p \) is non empty, and a non-null value is returned at line 4 or 6.

\(^{11}\) This stronger variant was not introduced in Section 3.2, since the proof of the generic Algorithm 1 does not require the stronger variant. In the proof of Algorithm 1, the stronger variant is hidden in the FLV-liveness property.

\(^{12}\) This stronger variant was not introduced in Section 3.2, since the proof of the original Algorithm 1 does not require the stronger variant. In the proof of Algorithm 1, the stronger variant is hidden in the FLV-liveness property.

---

**Figure 3. Illustration for FLV for class 3** \( n = 4, b = 1, f = 0, T_D = 3 \)

**4.2 Instantiations of **\( \text{Selector}(p, \phi) \)

A trivally instantiation of the \( \text{Selector} \) function consists in always returning the whole set of processes \( \Pi \). This trivially satisfies \( \text{Selector}-\text{validity}, \text{Selector}-\text{strongValidity} \) and \( \text{Selector}-\text{liveness} \). 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, with \( S \) being different in every phase.

In the benign fault model, it is sufficient that the \( \text{Selector} \) function always returns a single process rather than a set of processes. One such instantiation of the \( \text{Selector} \) function is the well known **rotating coordinator function** used in [5]. Another example is the **leader election** function used in [11].

---

**5 Instantiation examples**

In this section we show several well-known consensus algorithms obtained from Algorithm 1. Note that because the instantiated algorithms are expressed in the round model, some details of the original algorithms (retransmission rules, leader election, message acceptance policies, etc.) are hidden.

**5.1 Class 1 - OneThirdRule and FaB Paxos**

**OneThirdRule [6]** The OneThirdRule algorithm, which assumes benign faults only, is obtained from Algorithm 1 with the following parametrization: \( T_D = \lceil \frac{2n+2}{7} \rceil \). **FLAG = *, Selector\((p, \phi)\)** returning always \( \Pi \) and Algorithm 2 with \( T_D = \lceil \frac{2n+2}{7} \rceil \) as a FLV instantiation. It can be noticed that the instantiation leads to a (small) improvement of the original OneThirdRule algorithm. Details can be found in [19].

**FaB Paxos [16]** 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. We get FaB Paxos algorithm from Algorithm 1 by applying the following parametrization: \( T_D = \left( (n+3b+1)/2 \right) \), \( \text{FLAG} = \ast \), \( \text{Selector}(p, \phi) \) returning always \( \Pi \), and Algorithm 5 as an instantiation of \( \text{FLV} \) function (Algorithm 2 with \( T_D = \left( (n+3b+1)/2 \right) \)).

We now compare the instantiated version of FaB Paxos with the original algorithm. Since \( T_D = \left( (n+3b+1)/2 \right) \), it is easy to see that the deciding condition is the same in both algorithms. However, the selection condition of the two algorithms have (minor) differences. With the original 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.\(^{13}\) Therefore, if a number of received messages is smaller than \( n - b \), FaB Paxos will not select any value, while Algorithm 5 may still select a value by line 3. In this sense, the instantiation of Algorithm 1 is a (small) improvement of the original FaB Paxos algorithm.

The original FaB Paxos algorithm uses a coordinator-based implementation of the \( \mathcal{P}_\text{cons} \) predicate, based on signed messages \(^{17} \). By using the coordinator-free and signature-free implementation of \( \mathcal{P}_\text{cons} \), we can obtain coordinator-free and signature-free variant of FaB Paxos.

Algorithm 5 \( \text{FLV} \) for class 1 with \( T_D = \left( (n+3b+1)/2 \right) \)

1: \( \text{correctVotes}_p \leftarrow \left\{ v : \left( (v, -, -) \in \vec{\mu}_p \right) \right\} > \frac{n - b - 1}{2} \)
2: if \(|\text{correctVotes}_p| = 1\) then
3: return \( v \), s.t. \( v \in \text{correctVotes}_p \)
4: else if \(|\vec{\mu}_p| > n - b - 1\) then
5: return ?
6: else
7: return null

5.2 Class 2 - MQB

MQB is our new Byzantine consensus algorithm that requires \( n > 4b \). Compared to PBFT, it has the advantage not to need the (unbounded) variable \( \text{history}_p \), at the cost of requiring \( n > 4b \) instead of \( n > 3b \) (for PBFT). We get MQB from Algorithm 1 with the following parametrization:\(^{14}\) \( T_D = \left( \frac{n+2b+1}{2} \right) \), \( \text{FLAG} = \phi \), \( \text{Selector}(p, \phi) = \Pi \) and Algorithm 3 with \( T_D = \left( \frac{n+2b+1}{2} \right) \) as a \( \text{FLV} \) instantiation. Depending on the implementation of the \( \mathcal{P}_\text{cons} \) predicate (coordinator-based or coordinator-free), we get coordinator-based or coordinator-free variants of MQB.

5.3 Class 3 - Paxos and PBFT

We discuss Paxos as part of class 3 (rather than as part of class 2) to show the similarities between Paxos and PBFT, namely that the selection round for Paxos and PBFT are derived from the \( \text{FLV} \) function for class 3. Paxos and PBFT are algorithms that solves a sequence of instances of consensus (state machine replication). We consider here the instantiation of a single instance of consensus that represents the “core” of these algorithms. Both algorithms incorporate the optimizations related to \( \text{Selector}(p, \phi) \) and \( \text{validators} \) mentioned in Section 3.1.

Algorithm 6 \( \text{FLV} \) for class 3 with \( b = 0 \), \( T_D = \left( \frac{n+1}{2} \right) \)

1: \( \text{possibleVotes} \leftarrow \left\{ \text{vote}, ts \right\} \in \vec{\mu}_p \left\{ \right| \text{vote} = \text{vote} \lor ts > ts \text{\text{'s}} \left\} > \frac{n}{2} \)
2: if \(|\text{possibleVotes}| = 1\) then
3: return \( v \), s.t. \( v \in \text{possibleVotes} \)
4: else if \(|\vec{\mu}_p| > n - b - 1\) then
5: return ?
6: else
7: return null

Paxos [11] Paxos assumes benign faults only (\( b = 0 \)) and requires \( n > 2f \). We get Paxos from Algorithm 1 with the following parametrization:\(^{15}\) \( T_D = \left( \frac{n+1}{2} \right) \), \( \text{FLAG} = \phi \), \( \text{Selector}(p, \phi) \) implementing leader election, and Algorithm 6 as a \( \text{FLV} \) instantiation.

With only benign faults, the instantiation of the function \( \text{FLV} \) can be simplified. We now explain how to get Algorithm 6 from Algorithm 4. First, we can observe that any message \( \text{(vote, ts, history)} \) has the following property: \( \text{(vote, ts)} \in \text{history} \). Therefore, the set \( \text{correctVotes}_p \) is the same as the set \( \text{possibleVotes}_p \), which means that the set \( \text{correctVotes}_p \) is not needed. It follows that \( \text{history} \) is not needed in the \( \text{FLV} \) function, and by extension, the variable \( \text{history}_p \) is not needed in the consensus algorithm.

Because the unanimity property is not relevant in the benign case, lines 8-9 of Algorithm 4 can be removed. This allows us to merge lines 5-11 of Algorithm 4 into lines 4-5 of Algorithm 6.

PBFT [4] PBFT is designed for Byzantine faults (\( f = 0 \)) and requires \( n > 3b \). We get PBFT from Algorithm 1 with the following parametrization:\(^{16}\) \( T_D = 2b + 1 \), \( \text{FLAG} = \phi \), \( \text{Selector}(p, \phi) = \Pi \) and Algorithm 7 as a \( \text{FLV} \) instantiation. To get the instantiation as close as possible to PBFT, we have set \( n = 3b + 1 \), as in PBFT.

We explain now how to get Algorithm 7 from Algorithm 4. PBFT does not consider the unanimity property, which allows a significant simplification of Algorithm 4.

Footnote 12, here with reference to line 7 of Algorithm 4 instead of line 4 of Algorithm 2.

Footnote 15, here with reference to line 7 of Algorithm 4 instead of line 4 of Algorithm 2.
Algorithm 7 FLV for class 3 with $T_D = 2b + 1$ and $n = 3b + 1$

1: possibleVotes$_p \leftarrow \{(vote, ts, \neg) \in \overrightarrow{m}_p^v :$
\quad $|\{(vote', ts', \neg) : vote = vote' \land ts > ts'\}| > 2b$
2: correctVotes$_p \leftarrow \{v : (v, ts, \neg) \in$ possibleVotes$_p \land$
\quad $|\{(vote', ts', history') \in \overrightarrow{m}_p^v : (vote, ts) \in history'\}| > b\}$
3: if $|correctVotes_p| = 1$ then
4:    return $v \ s.t. \ (v, \neg, \neg) \in correctVotes_p$
5:  else if $|correctVotes_p| > 1$ or
6:    $|\{(vote, ts, \neg) \in \overrightarrow{m}_p^v : ts > ts\}| > 2b$ then
7:    return ?
8:  else
9:    return $null$

Indeed, without the unanimity property, lines 8-9 of Algorithm 4 can be removed. Then, we can merge the conditions of line 5 and line 7 of Algorithm 4 into line 5 of Algorithm 7.

PBFT uses a coordinator-based implementation of $\mathcal{P}_{cons}$ predicate that does not require signed messages [17]. By using the coordinator-free implementation of $\mathcal{P}_{cons}$ [2], we get a coordinator-free variant of PBFT.

6 Randomized consensus algorithms

Algorithm 1 can be adapted to support randomized consensus algorithms. The first modification is the introduction of randomization. In the context of binary consensus (initial value 0 or 1), line 11 is replaced with “$\text{select}_p := 1 \text{ or } 0$ with probability 0.5”. This allows all correct processes, by repeating the execution of the selection round, to select the same value with probability 1.

A second modification is needed, which is related to the “reliable channel” assumption of these algorithms. This assumption can be expressed by the following communication predicate that is required to hold in every round $r$ instead of predicates $\mathcal{P}_{cons}$ and $\mathcal{P}_{good}$:

$$\mathcal{P}_{rel}(r) \equiv \forall p \in C : |\{m \in \overrightarrow{m}_p^v : m \neq \bot\}| \geq n - b - f.$$  

Therefore, randomized protocols need a slightly different FLV-liveness property: for any set $\overrightarrow{m}_p^v$ with $n - b - f$ messages (instead of any set with all messages from correct processes), FLV must return a value different from $null$. Note that Algorithms 2 and 3 ensure this property, but not Algorithm 4. In other words, we can easily transform any consensus algorithm of class 1 or 2 into a randomized algorithm. We believe that this is not possible for consensus algorithms of class 3.

The instantiations of Ben-Or’s binary consensus algorithms [1] from Algorithm 1 can be found in [19].

7 Conclusion

The paper has presented a generic consensus algorithm parameterized with $T_D$, $\text{FLAG}$, $\text{Select}$ and $\text{FLV}$. Instantiation of these parameters led us to distinguish three classes of consensus algorithms (into which known consensus algorithms fit), and to identify the new MQB algorithm. As future work, we plan to develop a framework around our generic algorithm.

Acknowledgements: We would like to thank Fatemeh Borran, Martin Hutle, Segio Mena and Nuno Santos for their comments on an earlier version of the paper.

References