Unifying Byzantine Consensus Algorithms with Weak Interactive Consistency

Zarko Milosevic, Martin Hutle, and André Schiper

Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
{zarko.milosevic,martin.hutle,andre.schiper}@epfl.ch

Abstract. The paper considers the consensus problem in a partially synchronous system with Byzantine processes. In this context, the literature distinguishes *authenticated Byzantine* faults, where messages can be signed by the sending process (with the assumption that the signature cannot be forged by any other process), and *Byzantine* faults, where there is no mechanism for signatures (but the receiver of a message knows the identity of the sender). The paper proposes an abstraction called *weak interactive consistency* (*WIC*) that unifies consensus algorithms with and without signed messages. WIC can be implemented with and without signatures.

The power of WIC is illustrated on two seminal Byzantine consensus algorithms: the Castro-Liskov PBFT algorithm (no signatures) and the Martin-Alvisi FaB Paxos algorithms (signatures). WIC allows a very concise expression of these two algorithms.

1 Introduction

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

With respect to process faults, consensus can be considered with different fault assumptions. On the one end of the spectrum, processes fail only by crashing (so called *benign* faults); on the other end, faulty processes can exhibit an arbitrary (and even malicious) behavior. Among the latter, two fault models are considered in literature [1]: *authenticated Byzantine* faults, where messages can be signed by the sending process (with the assumption that the signature cannot be forged by any other process), and *Byzantine* faults, where there is no mechanism for signatures (but the receiver of a message knows the identity of the sender).[1] Consensus protocols that assume *Byzantine* faults (without authentication) are harder to develop and prove correct [3]. As a consequence, they tend to be more complicated and harder to understand than the protocols that assume

[1] In [2], the latter is called Byzantine faults with *oral messages*.

T. Abdelzaher, M. Raynal, and N. Santoro (Eds.): OPODIS 2009, LNCS 5923, pp. 300–314, 2009.

authenticated Byzantine faults, even when they are based on the same idea. The existence of these two fault models raises the following question: is there a way to transform an algorithm for authenticated Byzantine faults into an algorithm for Byzantine faults, or vice versa?

This question has been addressed by Srikanth and Toueg in [3] for the Byzantine agreement problem,[2] by defining the *authenticated broadcast* primitive. Authenticated broadcast is a communication primitive that provides additional guarantees compared to, *e.g.*, a normal (unreliable) broadcast. Srikanth and Toueg solve Byzantine agreement using authenticated broadcast, and show that authenticated broadcast can be implemented with and without signatures.

However, authenticated broadcast does not encapsulate all the possible uses of signed messages when solving consensus. One typical example is the Fast Byzantine Paxos algorithm [4], which relies on signed messages whenever the coordinator changes.

Complementing the approach of [3], we define an abstraction different from authenticated broadcast that we call *weak interactive consistency*.[3] Interactive consistency is defined in [7] as a problem where correct processes must agree on a vector such that the ith element of this vector is the initial value of the ith process if this process is correct. Our abstraction is a weaker variant of interactive consistency, hence the name "weak" interactive consistency. Similarly to authenticated broadcast, weak interactive consistency can be implemented with and without signatures. We illustrate the power of weak interactive consistency by reexamining two seminal Byzantine consensus algorithms: the Castro-Liskov PBFT algorithm, which does not use signatures [8], and the Martin-Alvisi FaB Paxos algorithm, which relies on signatures [4]. We show how to express these two algorithms using the weak interactive consistency abstraction, and call these two algorithms CL (for Castro-Liskov), resp. MA (for Martin-Alvisi).

Both CL and MA are very concise algorithms. Moreover, replacing in CL weak interactive consistency with a signature-free implementation basically leads to the original signature-free PBFT algorithm, while replacing in MA weak interactive consistency with a signature-based implementation basically leads to the original signature-based FaB Paxos algorithm. In the latter case, the algorithm obtained is almost identical to the original algorithm; in the former case, the differences are slightly more important (the differences are explained in [9]). In addition, using MA with a signature-free implementation of WIC allows us to derive a signature-free variant of FaB Paxos.

---

[2] In this problem, a transmitter sends a message to a set of processes, all processes eventually deliver a single message, and (i) all correct processes agree on the same message, (ii) if the transmitter is correct, then all correct processes agree on the message of the transmitter.

[3] In [5], Lamport defines "Weak Interactive Consistency Problem", as a general problem of reaching agreement. In [6], Doudou et al. define an abstraction called "Weak Interactive Consistency", with a different definition than ours. They use this abstraction to derive a state machine replication protocol resilient to authenticated Byzantine faults.

The rest of the paper is structured as follows. Weak interactive consistency is informally introduced in Section 2. Section 3 defines our model, and formally defines weak interactive consistency. In Section 4 we show that weak interactive consistency can be implemented with and without signatures. Section 5 describes the MA consensus algorithm (FaB Paxos expressed using weak interactive consistency) and the CL consensus algorithm (PBFT expressed using weak interactive consistency). Section 6 discusses related work, and Section 7 concludes the paper. For space reason, some proofs are omitted. They can be found in [9].

2 Weak Interactive Consistency: An Informal Introduction

2.1 On the Use of Signatures

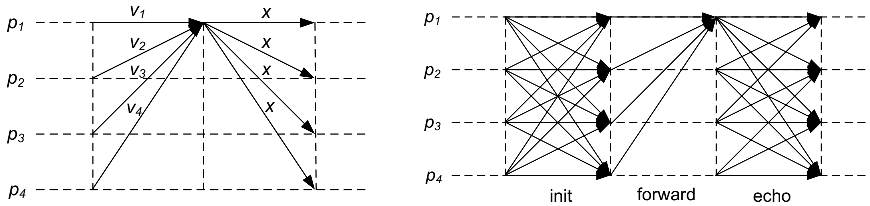
We start by addressing the following question: where are signatures used in coordinator based consensus algorithms? Signatures are typically used each time the coordinator changes, as done for example in the FaB Paxos algorithm [4]. The corresponding communication pattern is illustrated in Fig. 1(a), and addresses the following issue. Assume that the previous coordinator has brought the system into a configuration where a process already decided v; in this case, in order to ensure safety (*i.e.*, agreement) the new coordinator can only propose v. This is done as follows. First every process sends its current estimate to the new coordinator (v_i sent by p_i to p_1 in Fig. 1(a)). Second, if the coordinator p_1 receives a quorum of messages, then p_1 applies a function f that returns some value x. The quorum ensures that if a process has already decided v, then f returns v. Finally, the value returned by f is then sent to all (x sent by p_1 in Fig. 1(a)).

This solution does not work with a Byzantine coordinator: the value sent by the coordinator p_1 might not be the value returned by f. Safety can here be ensured using signatures: Processes p_i sign the estimates v_i sent to the coordinator p_1, and p_1 sends x together with the quorum of signed estimates it received. This allows a correct process p_i, receiving x from p_1, to verify whether x is consistent with the function f. If not, then p_i ignores x.

Are signatures mandatory here? We investigate this question, first addressing safety and then liveness.

2.2 Safe Updates Requires Neither Signatures Nor a Coordinator

As said, safety means that if a process has decided v, and thus a quorum of processes had v as their estimate at the beginning of the two rounds of Fig. 1(a), then each process can only update its estimate to v. This property can be ensured without signatures and without coordinator: each process p_i simply sends v_i to all, and each process p_i behaves like the coordinator: if p_i receives a quorum of messages, it updates its estimate with the value returned by f.



(a) Coordinator change: p_1 is the new coordinator.

(b) Three rounds to get rid of signatures when changing coordinator to p_1

Fig. 1.

This shows that updating the estimate maintaining safety does not require a coordinator. However, as we show in the next section, a coordinator is reintroduced for liveness.

2.3 Coordinator for Liveness

The coordinator in Fig. 1(a) has two roles: (i) it ensures safety (using signatures), and (ii) it tries to bring the system into a univalent configuration (if not yet so), in order to ensure liveness (*i.e.*, termination) of the consensus algorithm. A configuration typically becomes v-valent as soon as a quorum of correct processes update their estimate to v. This is ensured by a correct coordinator, if its message is received by a quorum of correct processes. Ensuring that a quorum of correct processes update their estimate to the same value v can also be implemented without signatures with an all-to-all communication schema, *if all correct processes receive the same set (of quorum size) of values.* Indeed, if two correct processes apply f to the same set of values, they update their estimate to the same value.

However, ensuring that all correct processes receive the same set of messages is problematic in the presence of Byzantine processes: (i) a Byzantine process can send v to some correct process p_i and v' to some other correct process p_j, and (ii) a Byzantine process can send v to some correct process p_i and nothing to some other correct process p_j.

These problems can be addressed using two all-to-all rounds and one all-to-coordinator rounds, as shown in Fig. 1(b) (to be compared with the "init" round followed by the "echo" round of authenticated broadcast, see [3]).

These three rounds can be seen as one all-to-all super-round that "always" satisfies integrity and "eventually" satisfies consistency:

Integrity. If a correct process p receives v from a correct process q in super-round r, then v was sent by q in super-round r.

Consistency. (i) If a correct process p_i sends v in super-round r, then every correct process receives v from p_i in super-round r, and (ii) all correct processes receive the same set of messages in super-round r.

As noted in Section 2.2, integrity ensures safety. As noted at the beginning of this section, eventual consistency allows us to eventually bring the system into a univalent configuration, thus ensuring liveness.

In the scheme of Fig. 1(b) we combine the concept of a coordinator as depicted in Fig. 1(a) with the authentication scheme of [3].

This scheme provides that in synchronous rounds (which eventually exist in a partially synchronous model, see Section 3), messages received by a correct coordinator in the “forward” round (see Fig. 1(b)), are received by all correct processes in the “echo” round (see Fig. 1(b)).[4] Note that without having the coordinator, the authentication scheme of [3] is not able to provide a super-round such that all processes receive the same set of messages at the end of this super-round, since a Byzantine process can always prevent this from happening.

We call the problem of always ensuring integrity and eventually consistency the *weak interactive consistency* problem, or simply *WIC*.[5] We show below that WIC is a unifying concept for Byzantine consensus algorithms. WIC can be implemented with signatures in two rounds (Fig. 1(a)), or without signatures in three rounds (Fig. 1(b)), as shown in Section 4.

3 Model and Definition of WIC

Assuming synchronous rounds is a strong assumption that we do not want to consider here. On the other side, an asynchronous system is not strong enough: WIC is not implementable in such a system. We consider a third option, *i.e.*, a partially synchronous system [1], or rather a slightly weaker variant of this model: we assume that the system alternates between good periods (during which the system is synchronous) and bad periods (during which the system is asynchronous). As in [1], we consider an abstraction on top of the system model, namely a round model, defined next. Using this abstraction rather than the raw system model improves the clarity of the algorithms and simplifies the proofs.

Among the n processes in our system, we assume that at most t are Byzantine. We do not make any assumption about behavior of Byzantine processes. The set of correct processes is denoted by $\mathcal{C}$.

3.1 Basic Round Model

In each round r, a process p sends a message according to a sending function S_p^r to a subset of processes, and, at the end of this round, computes a new state according to a transition function T_p^r, based on the vector of messages it received 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*). The state of process p in round

[4] The relay property of authenticated broadcast ensures that if a messages is received by a correct process in some round r', then it is received by all correct processes the latest in round $r'+1$ in the synchronous case.

[5] The relation with “interactive consistency” [7], is explained in Section 1.

r is denoted by s_p^r; the message sent by a correct[6] process is denoted by $S_p^r(s_p^r)$; messages received by process p in round r are denoted by $\boldsymbol{\mu}_p^r$.

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

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

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:

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

The reader can find in [1] the implementation of rounds that satisfy $\mathcal{P}_{good}$ during a good period in the presence of Byzantine processes.

3.3 WIC Predicate

We have informally defined WIC by an integrity property and by a consistency property that must hold "eventually". The integrity property is expressed by the predicate $\mathcal{P}_{int}$. "Eventual" consistency formally means that there exists a round r in which consistency holds:

$$\mathcal{P}_{cons}(r) \equiv \forall p, q \in \mathcal{C} : \ (\boldsymbol{\mu}_p^r[q] = S_q^r(s_q^r)\,) \ \wedge \ (\boldsymbol{\mu}_p^r = \boldsymbol{\mu}_q^r)$$

Therefore, WIC is formally expressed by the following predicate:

$$\boxed{\forall r : \mathcal{P}_{int}(r) \wedge \exists r : \mathcal{P}_{cons}(r)}$$

Note that $\mathcal{P}_{cons}(r)$ is stronger than $\mathcal{P}_{good}(r)$. Consider two correct processes p and q, and a Byzantine process sending message m to all processes in round r: $\mathcal{P}_{good}(r)$ allows m to be received by p and not by q; $\mathcal{P}_{cons}(r)$ does not allow this.

4 Implementing WIC

For implementing WIC, we show in this section that rounds that satisfy $\mathcal{P}_{good}$ can be transformed into a round that satisfies $\mathcal{P}_{cons}$. This transformation can be formally expressed thanks to the notion of *predicate translation*. Given some round r, we say that an algorithm A is a k-round translation of predicate $\mathcal{P}$ (*e.g.*, $\mathcal{P}_{good}$) into predicate $\mathcal{P}'$ (*e.g.*, $\mathcal{P}_{cons}$), if round r consists of k micro-rounds $\langle r, 1\rangle$

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

Algorithm 1. Translation with signatures

1: **Initialization:**
2: $\forall q \in \Pi : \ received_p[q] \leftarrow \bot$
3: **Round** $\rho = \langle r, 1 \rangle$ **:**
4: S_p^ρ:
5: send $\sigma_p(m_p, r)$ to $coord(r)$
6: T_p^ρ:
7: **if** $p = coord(r)$ **then**
8: $received_p \leftarrow \boldsymbol{\mu}_p^\rho$
9: **Round** $\rho = \langle r, 2 \rangle$ **:**
10: S_p^ρ:
11: **if** $p = coord(r)$ **then**
12: send $received_p$ to all
13: T_p^ρ:
14: **for all** $q \in \Pi$ **do**
15: $\boldsymbol{M}_p[q] \leftarrow \bot$
16: **if** signature of $\boldsymbol{\mu}_p^\rho[coord(r)][q]$ is valid **then**
17: $(msg, round) \leftarrow \sigma^{-1}(\boldsymbol{\mu}_p^\rho[coord(r)][q])$
18: **if** $round = r$ **then**
19: $\boldsymbol{M}_p[q] \leftarrow msg$

to $\langle r, k \rangle$ such that: (i) $\mathcal{P}$ holds for each micro-round $\langle r, i \rangle$, $i \in [1, k]$; (ii) each process p execute A in each round $\langle r, i \rangle$, $i \in [1, k]$; (iii) for each process p, the message m_p sent by p in micro-round $\langle r, 1 \rangle$ is the message sent by p in round r; (iv) for each process p, the messages received by p in round r are computed by p at the end of micro-round $\langle r, k \rangle$; and (v) $\mathcal{P}'$ holds for round r. We also say that round r is *simulated* by the k micro-rounds $\langle r, 1 \rangle$ to $\langle r, k \rangle$.

We give two translations, one with and one without digital signatures. Both translations rely on a coordinator. The translation with signatures requires two micro-rounds with the communication pattern of Fig. 1(a) whereas the translation without signatures requires three micro-rounds with the communication pattern of Fig. 1(b)[7]. The coordinator of round r is denoted by $coord(r)$.

We will analyze the two translations in the following cases: (i) $coord(r)$ is correct and the micro-rounds satisfy $\mathcal{P}_{good}$, and (ii) $coord(r)$ may be faulty and only $\mathcal{P}_{int}$ holds for the micro-rounds. In case (i), we have a translation of $\mathcal{P}_{good}$ into $\mathcal{P}_{cons}$. Case (ii) ensures that the translation is harmless during bad periods, or if the coordinator is faulty.

Therefore, the big picture is the following. If we assume a sufficient long good period, then [1] shows how to implement rounds for which $\mathcal{P}_{good}$ eventually holds. Moreover, the rotating coordinator paradigm eventually ensures rounds with a correct coordinator. Together, this eventually ensures case (i).

4.1 Translation with Signatures

Algorithm 1 is a 2-round translation with signatures that preserves $\mathcal{P}_{int}$ (*i.e.*, if $\mathcal{P}_{int}$ holds for every micro-round, then $\mathcal{P}_{int}$ holds for the round). Moreover, when $coord(r)$ is correct, it translates $\mathcal{P}_{good}$ into $\mathcal{P}_{cons}$. At the beginning of Algorithm 1 every process p has a message m_p (line 5); at the end every process p has a vector $\boldsymbol{M}_p$ of received messages (lines 15, 19)[8]. Vector $received_p$ (line 8) represents the messages that p received (one element per process). Message m signed by p is denoted by $\sigma_p(m)$. The function σ^{-1} allows us to get back the original message out of a signed message.

[7] In Section 2 we used terms super-round and round. From here on, we use term *round* for what we called *super-round* and *micro-round* for what we called *round*.

[8] When round r is simulated using Algorithm 1, m_p is initially set to the $S_p^r(s_p^r)$ and in the end $\boldsymbol{\mu}_p^r$ is set to $\boldsymbol{M}_p$.

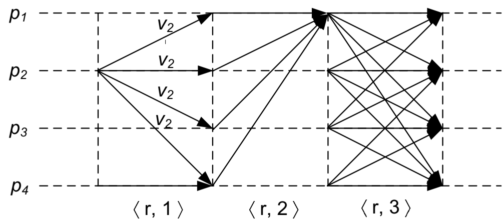


Fig. 2. Translation without signatures from the point of view of $\boldsymbol{v_2}$ sent by $\boldsymbol{p_2}$ ($\boldsymbol{p_1}$ is the coordinator)

Algorithm 1 is straightforward: each process p sends its signed message m_p to the coordinator (line 5) in micro-round $\langle r, 1\rangle$. In micro-round $\langle r, 2\rangle$, the coordinator forwards all messages received (line 12).

Proposition 1. *Algorithm 1 preserves* $\mathcal{P}_{int}(r)$*.*

Proposition 2. *If* $coord(r)$ *is correct, then Algorithm 1 translates* $\mathcal{P}_{good}$ *into* $\mathcal{P}_{cons}$*.*

4.2 Translation without Signatures

Algorithm 2 is a 3-round translation with signatures, inspired by [8], that preserves $\mathcal{P}_{int}$ (*i.e.*, if $\mathcal{P}_{int}$ holds for every micro-round, then $\mathcal{P}_{int}$ holds for the round). Moreover, when $coord(r)$ is correct, it translates $\mathcal{P}_{good}$ into $\mathcal{P}_{cons}$. It requires $n \geq 3t+1$. At the beginning of Algorithm 2 every process p has a message m_p (line 7); at the end every process p has a vector $\boldsymbol{M}_p$ of received messages (lines 22, 24)[9].

We informally explain Algorithm 2 using Fig. 2 Compared to Fig. 1(b), Fig. 2 shows only the messages relevant to v_2 sent by p_2. Process p_1 is the coordinator. In micro-round $\langle r, 1\rangle$, process p_2 sends v_2 to all. In micro-round $\langle r, 2\rangle$, all processes send the value received from p_2 to the coordinator. The coordinator then compares the value received from p_2 in micro-round $\langle r, 1\rangle$, say v_2, with the value indirectly received from the other processes. If at least $2t+1$ values v_2 have been received by the coordinator p_1, then p_1 keeps v_2 as the value received from p_2. Otherwise p_1 sets the value received from p_2 to $\perp$. This guarantees that, if p_1 keeps v_2, then at least $t+1$ correct processes have received v_2 from p_2 in micro-round $\langle r, 1\rangle$.

Finally, in micro-round $\langle r, 3\rangle$ every process sends the value received from p_2 in micro-round $\langle r, 1\rangle$ to all. The final value received from p_2 at the end of micro-round $\langle r, 3\rangle$ is computed as follows at each process p_i. Let val_i be the value received by p_i from coordinator p_1 in micro-round $\langle r, 3\rangle$. If val_i is $\perp$ then p_i receives $\perp$ from p_2. Process p_i receives $\perp$ from p_2 in another case: if p_i did not

[9] When round r is simulated using Algorithm 2, m_p is initially set to the $S_p^r(s_p^r)$ and in the end $\boldsymbol{\mu}_p^r$ is set to $\boldsymbol{M}_p$.

Algorithm 2. Translation without signatures ($n \geq 3t+1$)

1: **Initialization:**
2: $\forall q \in \Pi :\ received_p[q] \leftarrow \bot$
3: **Round** $\rho = \langle r, 1 \rangle$ **:**
4: S_p^ρ:
5: send m_p to all
6: T_p^ρ:
7: $received_p \leftarrow \boldsymbol{\mu}_p^\rho$
8: **Round** $\rho = \langle r, 2 \rangle$ **:**
9: S_p^ρ:
10: send $received_p$ to $coord(r)$
11: T_p^ρ:
12: **if** $p = coord(r)$ **then**
13: **for all** $q \in \Pi$ **do**
14: **if** $\left|\left\{ q' \in \Pi : \boldsymbol{\mu}_p^\rho[q'][q] = received_p[q] \right\}\right| < 2t+1$ **then**
15: $received_p[q] \leftarrow \bot$
16: **Round** $\rho = \langle r, 3 \rangle$ **:**
17: S_p^ρ:
18: send $\langle received_p \rangle$ to all
19: T_p^ρ:
20: **for all** $q \in \Pi$ **do**
21: **if** $(\boldsymbol{\mu}_p^\rho[coord(r)][q] \neq \bot) \wedge \left|\left\{ i \in \Pi : \boldsymbol{\mu}_p^\rho[i][q] = \boldsymbol{\mu}_p^\rho[coord(r)][q] \right\}\right| \geq t+1$ **then**
22: $\boldsymbol{M}_p[q] \leftarrow \boldsymbol{\mu}_p^\rho[coord(r)][q]$
23: **else**
24: $\boldsymbol{M}_p[q] \leftarrow \bot$

receive $t+1$ values equal to val_i in micro-round $\langle r, 3\rangle$. Otherwise, at least $t+1$ values received by p_i in micro-round $\langle r, 3\rangle$ are equal to val_i, and p_i receives val_i from p_2.

Proposition 3. *Algorithm 2 preserves $\mathcal{P}_{int}(r)$.*

Proof. Let p, q be two correct processes. Assume for the sake of contradiction that $S_p^r(s_p^r) = v$, $\boldsymbol{M}_q[p] = v'$, where $v' \neq v$, $v' \neq \bot$. Therefore, by line 21, we have $\left|\left\{ i : \boldsymbol{\mu}_q^\rho[i][p] = v' \right\}\right| \geq t+1$. Consequently, for at least one correct process c we have $\boldsymbol{\mu}_q^\rho[c][p] = v'$. Element $\boldsymbol{\mu}_q^\rho[c][p]$ is the message received by c from p in round $\langle r, 1\rangle$, which is $received_c[p]$. However, $received_c[p] = v'$ is in contradiction with the assumption that p and c are correct. □

Proposition 4. *If $coord(r)$ is correct, then Algorithm 2 translates $\mathcal{P}_{good}$ into $\mathcal{P}_{cons}$.*

Proof. Let p, q be two correct processes, and s some other process (not necessarily correct). Let c be the correct coordinator. Let $\mathcal{P}_{good}(\langle r, 1\rangle)$, $\mathcal{P}_{good}(\langle r, 2\rangle)$ and $\mathcal{P}_{good}(\langle r, 3\rangle)$ hold. We first show (i) $\boldsymbol{M}_p[q] = S_q^r(s_q^r)$, and then (ii) $(\boldsymbol{M}_p[s] = v \neq \bot) \Rightarrow (\boldsymbol{M}_q[s] = v)$. Note that from (ii) it follows directly that $(\boldsymbol{M}_p[s] = \bot) \Rightarrow (\boldsymbol{M}_q[s] = \bot)$.

(i): In micro-round $\langle r, 1\rangle$, process q sends $v = S_q^r(s_q^r)$ to all, and because of $\mathcal{P}_{good}(\langle r, 1\rangle)$, v is received by all correct processes. For all those correct processes i, we have $received_i[q] = v$ (*). In micro-round $\langle r, 2\rangle$, every correct process forwards v to the coordinator c, and c receives all these messages. Since $n \geq 3t+1$ there are at least $2t+1$ correct processes. Therefore the condition of line 14 is false for q because $|\{q' \in \Pi : \boldsymbol{\mu}_c^\rho[q'][q] = received_c[q]\}| \geq 2t+1$, *i.e.*, $received_c[q]$ is not set to $\bot$. By (*) above, we have $received_c[q] = v$. Because of $\mathcal{P}_{good}(\langle r, 3\rangle)$ all messages sent by correct processes in micro-round $\langle r, 3\rangle$ are received by all correct processes. Thus, for p at line 21, we have $\boldsymbol{\mu}_p^\rho[coord(r)][q] \neq \bot$. Moreover,

by (*), condition $\left|\left\{i \in \Pi : \boldsymbol{\mu}_p^\rho[i][q] = \boldsymbol{\mu}_p^\rho[coord(r)][q]\right\}\right| \geq t+1$ is true. This leads p to execute line 22, *i.e.*, assign v to $\boldsymbol{M}_p[q]$.

(ii): Let us assume $\boldsymbol{M}_p[s] = v \neq \bot$, and consider Algorithm 2 from the point of view of p. Consider the loop at line 20 for process s. By line 22, we have $\boldsymbol{\mu}_p^\rho[coord(r)][s] = v$. Since the coordinator is correct, in order to have $\boldsymbol{\mu}_p^\rho[coord(r)][s] = v$, the condition of line 14 is true at c for process s, *i.e.*, $|\{q' \in \Pi : \boldsymbol{\mu}_c^\rho[q'][s] = received_c[s]\}| \geq 2t+1$. This means that at least $2t+1$ processes, including at least $t+1$ correct processes, have received from s in micro-round $\langle r, 1\rangle$ the same message that c received from s, namely v ($\star$). In micro-round $\langle r, 3\rangle$, these $t+1$ correct processes send *received* to all. Because $\mathcal{P}_{good}(\langle r, 3\rangle)$ holds, all these messages are received by q in round $\langle r, 3\rangle$ ($\star\star$). Consider now Algorithm 2 from the point of view of q, and again the loop at line 20 for process s. Since the coordinator is correct, it sends at line 18 the same message to p and to q, *i.e.*, at q we also have $\boldsymbol{\mu}_q^\rho[coord(r)][s] = v$. By ($\star$) and ($\star\star$), the condition $\left|\left\{i \in \Pi : \boldsymbol{\mu}_q^\rho[i][s] = \boldsymbol{\mu}_q^\rho[coord(r)][s]\right\}\right| \geq t+1$ is true. Therefore q executes line 22 with $\boldsymbol{\mu}_p^\rho[coord(r)][s] = v$. □

5 Achieving Consensus with WIC

In this section we show how to express the consensus algorithms of Castro-Liskov [8] and Martin-Alivisi [4] using WIC. The algorithm of Castro and Lisko solves a sequence of instances of consensus (state machine replication). For simplicity, we consider only one instance of consensus.

Consensus is defined by agreement, termination and a validity property. We consider two validity properties, weak and strong validity [1]:

Agreement. No two correct processes decide differently.
Termination. All correct processes eventually decide.
Weak Validity. If all processes are correct and if a correct process decides v, then v is the initial value of some process.
Strong Validity. If all correct processes have the same initial value v and a correct process decides, then it decides v.

Both, [8] and [4] achieve only weak validity. Weak validity allows correct processes to decide on the initial value of a Byzantine process. With strong validity, however, this is only possible if not all correct processes have the same initial value. We give algorithms for both, weak and strong validity, and show that strong validity is in fact easy to ensure.

5.1 On the Use of WIC

We express the algorithms of this section in the round model defined in Section 3. All rounds of MA and CL require $\mathcal{P}_{int}$ to hold. Some of the rounds require $\mathcal{P}_{cons}$ to eventually hold. These rounds can be simulated using, *e.g.*, Algorithm 1 or Algorithm 2. We explicitly mention those rounds of MA and CL as rounds "in which $\mathcal{P}_{cons}$ must eventually hold". The other rounds of MA and CL are ordinary rounds.

Algorithm 3. MA (weak validity)

Initialization:
  $x_p \leftarrow v_p \in V$
Round $r = 1$**:**
  S_p^r:
    if $p = coord$ **then**
      send x_p to all
  T_p^r:
    if $\boldsymbol{\mu}_p^r[coord] \neq \bot$ **then**
      $x_p \leftarrow \boldsymbol{\mu}_p^r[coord]$
Round $r = 2$**:**
  S_p^r:
    send x_p to all
  T_p^r:
    if $\exists \bar{v} \neq \bot : \#(\bar{v}) \geq \lceil (n + 3t + 1)/2 \rceil$ **then**
      Decide $\bar{v}$
Round $r \geq 3$**:**
  Same as Algorithm 4 without **Initialization**

Algorithm 4. MA (strong validity)

Initialization:
  $x_p \leftarrow v_p \in V$
Round $r = 2\phi - 1$**:**
  /* in which $\mathcal{P}_{cons}$ must eventually hold */
  S_p^r:
    send x_p to all
  T_p^r:
    if $\#(\bot) \leq t$ **then**
      $x_p \leftarrow \min \{ v : \nexists v' \in V \ s.t. \ \#(v') > \#(v) \}$
Round $r = 2\phi$**:**
  S_p^r:
    send x_p to all
  T_p^r:
    if $\exists \bar{v} \neq \bot : \#(\bar{v}) \geq \lceil (n + 3t + 1)/2 \rceil$ **then**
      Decide $\bar{v}$

5.2 MA Algorithm

The algorithm of Martin and Alvisi [4] is expressed in the context of "proposers", "acceptors" and "learners". For simplicity, we express here consensus without considering these roles.

We give two algorithms. The first solves consensus with weak validity and is given as Algorithm 3. In the first phase it corresponds to the "common case" protocol of [4]. All later phases correspond to the "recovery protocol" of [4] (cf. Algorithm 4). The second algorithm solves consensus with strong validity, and is even simpler: all phases are identical, see Algorithm 4. In both algorithms, the notation $\#(v)$ is used to denote the number of messages received with value v, *i.e.*, $\#(v) \equiv \left| \left\{ q \in \Pi \ : \ \boldsymbol{\mu}_p^r[q] = v \right\} \right|$.

For MA with weak validity, the first phase needs an initial coordinator, which is denoted by *coord*. Note that WIC is relevant only to rounds $2\phi - 1$, $\phi > 1$, of Algorithm 4. If rounds $2\phi - 1$ are simulated using Algorithm 1, we get the original algorithm of [4]. If rounds $2\phi - 1$ are simulated using Algorithm 2, we get a new algorithm. In this new algorithm, similarly to the algorithm in [4], fast decision is possible in two rounds; however, signatures are not used in the recovery protocol.

Both algorithms require $n \geq 5t+1$. Agreement, weak validity and strong validity hold without synchrony assumptions. Termination requires (i) one phase ϕ such that $\mathcal{P}_{cons}(2\phi - 1)$ holds, and (ii) one phase $\phi' \geq \phi$ such that $\mathcal{P}_{good}(2\phi')$ holds.

Theorem 1. *If $n \geq 5t + 1$ then Algorithm 3 (resp. Algorithm 4) ensures weak (resp. strong) validity and agreement. Termination holds if in addition the following condition holds:*

$$\exists \phi : \ \mathcal{P}_{cons}(2\phi - 1) \wedge \exists \phi' \geq \phi : \ \mathcal{P}_{good}(2\phi')$$

Algorithm 5. CL (weak validity)

1: **Initialization:**
2: $x_p \leftarrow v_p \in V$
3: $\mathit{pre\text{-}vote}_p \leftarrow \emptyset$ /* see Algorithm 6 */
4: $\mathit{vote}_p \leftarrow \bot$ /* see Algorithm 6 */
5: $\mathit{tVote}_p \leftarrow 0$ /* see Algorithm 6 */
6: **Round** $r = 3\phi - 2 = 1$**:**
7: S_p^r:
8: **if** $p = \mathit{coord}$ **then**
9: send $\langle x_p \rangle$ to all
10: T_p^r:
11: **if** $\boldsymbol{\mu}_p^r[\mathit{coord}] \neq \bot$ **then**
12: add $(\boldsymbol{\mu}_p^r[\mathit{coord}], \phi)$ to $\mathit{pre\text{-}vote}_p$
13: **Round** $r = 3\phi - 1 = 2$**:**
14: S_p^r:
15: **if** $\exists (v, \phi) \in \mathit{pre\text{-}vote}_p$ **then**
16: send $\langle v \rangle$ to all
17: T_p^r:
18: **if** $\#(v) \geq \lceil (n + t + 1)/2 \rceil$ **then**
19: $\mathit{vote}_p \leftarrow v$
20: $\mathit{tVote}_p \leftarrow \phi$
21: **Round** $r = 3\phi = 3$**:**
22: S_p^r:
23: **if** $\mathit{tVote}_p = \phi$ **then**
24: send $\langle \mathit{vote}_p \rangle$ to all
25: T_p^r:
26: **if** $\exists \bar{v} \neq \bot : \#(\bar{v}) \geq \lceil (n+t+1)/2 \rceil$ **then**
27: DECIDE $\bar{v}$
28: **Round** $r \geq 4$**:**
29: Same as Algorithm 6 without **Initialization**

Note that $n \geq 5t + 1$ is only needed for terrmination, while only $n \geq 3t + 1$ is needed for agreement and strong validity.

5.3 CL Algorithm

The algorithm of Castro and Liskov [8] solves a sequence of instances of consensus (state machine replication). For simplicity, we consider only one instance of consensus. As for MA, we give two algorithms.

The first solves consensus with weak validity and is given as Algorithm 5. In the first phase it corresponds to the "common case" protocol of [8]. All later phases correspond to the "view change protocol" of [8] (cf. Algorithm 6). The second algorithm solves consensus with strong validity, and is even simpler: all phases are identical, see Algorithm 6. In both algorithms, the notation $\#(v)$ is used to denote the number of messages received with value v, *i.e.*, $\#(v) \equiv \left|\left\{q \in \Pi \ : \ \boldsymbol{\mu}_p^r[q] = v\right\}\right|$.

For CL with weak validity, the first phase needs an initial coordinator, which is denoted by *coord*. In round 1 of this phase the coordinator sends its initial value to all. In round 2 every process that has received the initial value from the coordinator in round 1 resends this value to all. Every process p, upon receiving this value from at least $\lceil (n + t + 1)/2 \rceil$ processes, updates vote_p and tVote_p (lines 19 and 20), and then sends vote_p to all in round 3. A process receiving in round at least $\lceil (n + t + 1)/2 \rceil$ messages with the same value v, decides v. For CL with weak validity, WIC is relevant only to rounds $3\phi - 2$, $\phi > 1$ (cf. Algorithm 6). If rounds $3\phi - 2$, $\phi > 1$ are simulated using Algorithm 2, we get an algorithm close to the original algorithm of [8] (the differences are explained in [9]). If rounds $3\phi - 2$, $\phi > 1$ are simulated using Algorithm 1, we get a variant of PBFT with signatures.

CL with strong validity (see Algorithm 6) consists of a sequence of phases ϕ, where each phase ϕ has three rounds $3\phi - 2$, $3\phi - 1$ and 3ϕ. The role of the variables is explained in comments, see lines 2–5. WIC is needed only in round $3\phi - 2$. Rounds $3\phi - 1$ and 3ϕ are the same as rounds 2 and 3 of Algorithm 5.

Algorithm 6. CL (strong validity)

```
Initialization:
   x_p ← v_p ∈ V                                  /* v_p is the initial value of p */
   pre-vote_p ← ∅        /* set of (v, φ), where φ is the phase in which v is added to pre-vote_p */
   vote_p ← ⊥                                     /* the most recent vote */
   tVote_p ← 0                                    /* phase in which vote_p was last updated */

Procedure pre-vote_p.add(v, φ) :
   if ∃(v, φ') ∈ pre-vote_p then
      remove (v, φ') from pre-vote_p
   add (v, φ) to pre-vote_p

Round r = 3φ − 2 :                     /* round in which P_cons must eventually hold */
   S_p^r:
      send ⟨vote_p, tVote_p, pre-vote_p, x_p⟩ to all
   T_p^r:
      proposals_p ← ∅ ; I_p ← ∅                   /* temporary variables */
      if μ_p^r contains at least ⌈(n + t + 1)/2⌉ messages ⟨vote, tVote, pre-vote, x⟩ then
         for all m ∈ μ_p^r do
            if
            |{m' ∈ μ_p^r : (m'.tVote < m.tVote) ∨ (m'.tVote = m.tVote ∧ m'.vote = m.vote)}| ≥
            ⌈(n + t + 1)/2⌉ and
            |{m' ∈ μ_p^r : ∃(v, φ') ∈ m'.pre-vote s.t. φ' ≥ m.tVote ∧ v = m.vote}| ≥ t + 1 then
               proposals_p ← proposals_p ∪ m.vote
         if |proposals_p| > 0 then
            pre-vote_p.add(min(proposals_p), φ)
         else if exist at least ⌈(n + t + 1)/2⌉ messages m' ∈ μ_p^r : m'.vote = ⊥ then
            I_p ← {m.x s.t. m ∈ μ_p^r}
            x̄ ← min{v : ∄v' ∈ I_p s.t. #(v') > #(v)}
            pre-vote_p.add(x̄, φ)

Round r = 3φ − 1 :
   S_p^r:
      if ∃(v, φ) ∈ pre-vote_p then
         send ⟨v⟩ to all
   T_p^r:
      if #(v) ≥ ⌈(n + t + 1)/2⌉ then
         vote_p ← v
         tVote_p ← φ

Round r = 3φ :
   S_p^r:
      if tVote_p = φ then
         send ⟨vote_p⟩ to all
   T_p^r:
      if ∃v̄ ≠ ⊥ : #(v̄) ≥ ⌈(n + t + 1)/2⌉ then
         DECIDE v̄
```

Both algorithms (CL with weak validity and CL with strong validity) require $n \geq 3t+1$. Agreement, weak validity and strong validity hold without synchrony assumptions. Termination requires (i) one phase ϕ such that $\mathcal{P}_{cons}(3\phi - 2)$, $\mathcal{P}_{good}(3\phi - 1)$ and $\mathcal{P}_{good}(3\phi)$ hold.

Theorem 2. *If $n \geq 3t+1$ then Algorithm 5 (resp. Algorithm 6) ensures weak (resp. strong) validity and agreement. Termination holds if in addition the following condition holds:*

$$\exists \phi : \mathcal{P}_{cons}(3\phi - 2) \wedge \mathcal{P}_{good}(3\phi - 1) \wedge \mathcal{P}_{good}(3\phi).$$

6 Related Work

Unification. To the best of our knowledge, there is little work that has tried to unify algorithms for Byzantine faults that use signatures and algorithms that do not use signatures. We are only aware of the work of Skrikanth and Toueg [3] related to *authenticated broadcast* (as already mentioned in Section 1). Further there is the work of Neiger and Toueg [10] who have developed methods to automatically translate protocols tolerant of benign faults to ones tolerant of more severe faults, including Byzantine faults, in the context of synchronous systems. Abstractions introduced by Lampson in [11] are relevant only to PBFT [8], and its hard to see how these abstractions can be extended to other Byzantine consensus protocols. Orthogonal to our approach, [12] proposes a solution for implementing digital signatures using MACs (message authentication codes).

Byzantine consensus algorithms. Several models with Byzantine faults have been considered for solving consensus or closely related problems, such as Byzantine agreement or state machine replication. The early work of Lamport, Shostak and Pease [7,2] considers a synchronous system and proposes algorithms for Interactive Consistency and Byzantine agreement with and without signatures. A weaker system model, namely partial synchrony, has been considered by Dwork, Lynch and Stockmeyer [1]. This is also the model we consider in this paper. In [1], the authors propose two consensus algorithms for Byzantine faults: one that uses signatures, and one without signatures. In [13], the authors consider a system with less synchrony than provided by partially synchrony, and describe a consensus algorithm that does not use signatures. Randomized consensus can be solved in an asynchronous system with Byzantine faults, as shown first in [14]. In [15], the authors solve consensus with Byzantine faults assuming a system equipped with a *Trusted Timely Computing Base* (TTCB).

Our CL algorithm is a simplified version of PBFT. Other authors have tried to increase the efficiency of PBFT, *e.g.* [16]. Recently, [17] has proposed a consensus algorithm for Byzantine faults that ensures strong validity, in which the decision is possible in the first round.

7 Conclusion

The paper has introduced the *weak interactive consistency* (or *WIC*) abstraction, and has shown that WIC allows to unify Byzantine consensus algorithms with and without signatures. This has been illustrated on two seminal Byzantine consensus algorithm, namely on the FaB Paxos algorithm [4] and on the PBFT algorithm [8]. In both cases this leads to a very concise algorithm. Apart from these two algorithms, we also managed to express two other algorithms for Byzantine faults using WIC: the algorithms for Byzantine faults of [1] and a deterministic version of the algorithm for Byzantine faults of [14], which is the basis for the algorithm in [13]. Therefore, we conjecture that WIC is the abstraction that underlines all Byzantine consensus algorithms for partial synchronous systems.

Acknowledgements. We would like to thank Nuno Santos for his comments on an earlier version of the paper.

References

1. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM 35(2), 288–323 (1988)
2. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982)
3. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distributed Computing 2(2), 80–94 (1987)
4. Martin, J.P., Alvisi, L.: Fast byzantine consensus. Transactions on Dependable and Secure Computing 3(3), 202–214 (2006)
5. Lamport, L.: The weak byzantine generals problem. J. ACM 30(3), 668–676 (1983)
6. Doudou, A., Guerraoui, R., Garbinato, B.: Abstractions for devising byzantine-resilient state machine replication. In: SRDS (2000)
7. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980)
8. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems 20(4), 398–461 (2002)
9. Milosevic, Z., Hutle, M., Schiper, A.: Unifying byzantine consensus algorithms with weak interactive consistency (LSR-REPORT-2009-003)
10. Neiger, G., Toueg, S.: Automatically increasing the fault-tolerance of distributed algorithms. J. Algorithms 11(3), 374–419 (1990)
11. Lampson, B.: The abcd's of paxos. In: PODC, p. 13. ACM Press, New York (2001)
12. Aiyer, A.S., Alvisi, L., Bazzi, R.A., Clement, A.: Matrix signatures: From mACs to digital signatures in distributed systems. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 16–31. Springer, Heidelberg (2008)
13. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Consensus with byzantine failures and little system synchrony. In: DSN, pp. 147–155 (2006)
14. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: PODC, pp. 27–29. ACM, New York (1983)
15. Correia, M., Neves, N.F., Lung, L.C., Veríssimo, P.: Low complexity byzantine-resilient consensus. Distributed Computing 17(3) (2005)
16. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.L.: Zyzzyva: speculative byzantine fault tolerance. In: SOSP, pp. 45–58 (2007)
17. Song, Y.J., van Renesse, R.: Bosco: One-step byzantine asynchronous consensus. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 438–450. Springer, Heidelberg (2008)