Brief Announcement: Revisiting Signature-Free Asynchronous Byzantine Consensus

Christian Cachin
University of Bern, Switzerland

Luca Zanolini
University of Bern, Switzerland

Abstract

Among asynchronous, randomized, and signature-free implementations of consensus, the protocols of Mostéfaoui et al. (PODC 2014 and JACM 2015) represent a landmark result, which has been extended later and taken up in practical systems. The protocols achieve optimal resilience and take, in expectation, only a constant expected number of rounds and have quadratic message complexity. Randomization is provided through a common-coin primitive. However, the first version of this simple and appealing protocol suffers from a little-known liveness issue due to asynchrony. The JACM 2015 version avoids the problem, but is considerably more complex.

This work revisits the original protocol of PODC 2014 and points out in detail why it may not progress. A fix for the protocol is presented, which does not affect any of its properties, but lets it regain the original simplicity in asynchronous networks enhanced with a common-coin protocol.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols; Software and its engineering → Distributed systems organizing principles

Keywords and phrases Randomized consensus

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.51

Related Version *Full Version*: `https://arxiv.org/pdf/2005.08795.pdf`

Funding This work has been funded by the Swiss National Science Foundation (SNSF) under grant agreement Nr. 200021_188443 (Advanced Consensus Protocols).

1 Introduction

Consensus is a fundamental abstraction in distributed systems. It captures the problem of reaching agreement among multiple processes on a common value, despite unreliable communication and the presence of faulty processes. Consensus in asynchronous networks requires randomization.

Mostéfaoui et al. [4] presented a signature-free round-based asynchronous consensus algorithm for binary values at PODC 2014. It had received considerable attention because it was the first asynchronous consensus protocol with optimal resilience, tolerating up to $f < \frac{n}{3}$ Byzantine processes, that did not use digital signatures. Hence, it needs only authenticated channels and remains secure against a computationally unbounded adversary. Moreover, it takes $O(n^2)$ constant-sized messages in expectation. (This description excludes the necessary cost for implementing randomization, for which the protocol relies on an abstract common-coin primitive, as defined by Rabin [6].) The algorithm represents a landmark result, and practical systems, such as "Honey Badger BFT" [3], have later taken it up, and many others have extended it.

However, this protocol, which we call the *PODC-14* version [4] in the following, suffers from a subtle and little-known problem. It may violate liveness, i.e., an adversary can prevent progress among the correct processes by controlling the messages between them and by sending them values in a specific order. This has explicitly pointed out by Tholoniat

© Christian Cachin and Luca Zanolini;
licensed under Creative Commons License CC-BY 4.0
35th International Symposium on Distributed Computing (DISC 2021).
Editor: Seth Gilbert; Article No. 51; pp. 51:1–51:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

source: https://doi.org/10.48350/165019 | downloaded: 12.10.2024

and Gramoli [7] in 2019. Indeed, the corresponding journal publication by Mostéfaoui et al. [5], to which we refer as the *JACM-15* version, only touches on the issue and goes on to present a modified, extended protocol. This fixes the problem, but requires also many more communication steps and adds considerable complexity.

In this work, we introduce an improved protocol that changes the PODC-14 version in a simple, but crucial way and thereby regains the simplicity of the original result. The reader may find detailed descriptions and arguments in the full version [2].

2 The Byzantine consensus algorithm of PODC-14

Let us briefly recall the consensus algorithm from the PODC-14 version (Alg. 1). A correct process may propose a binary value b by invoking *rbc-propose*(b); the consensus abstraction decides for b through an *rbc-decide*(b) event. The algorithm proceeds in rounds. In each round, an instance of *bv-broadcast* is invoked, a primitive introduced in the same paper [4, Figure 1]. A correct process p_i executes *bv-broadcast* and waits for a value b to be delivered, identified by a tag characterizing the current round. When such a bit b is received, p_i adds b to *values* and broadcasts b through an AUX message to all processes. Whenever a process receives an AUX message containing b from p_j, it stores b in a local set $aux[j]$. Once p_i has received a set $B \subseteq values$ of values such that every $b \in B$ has been delivered in AUX messages from at least $n - f$ processes, then p_i releases the coin for the round. Subsequently, the process waits for the coin protocol to output a binary value s through *output-coin*(s), tagged with the current round number.

Process p_i then checks if there is a single value b in B. If so, and if $b = s$, then it decides for value b. The process then proceeds to the next round with proposal b. If there is more than one value in B, then p_i changes its proposal to s. In any case, the process starts another round and invokes a new instance of *bv-broadcast* with its proposal.

Algorithm 1 Randomized binary consensus according to Mostéfaoui et al. [4] (code for p_i).

```
State
    round ← 0: current round
    values ← {}: set of bv-delivered binary values for the round
    aux ← [{}]^n: stores sets of values that have been received in AUX messages in the round
upon event rbc-propose(b) do
    invoke bv-broadcast(b) with tag round
upon bv-deliver(b) with tag r such that r = round do
    values ← values ∪ {b}
    send message [AUX, round, b] to all p_j ∈ P
upon receiving a message [AUX, r, b] from p_j such that r = round do
    aux[j] ← aux[j] ∪ {b}
upon exists B ⊆ values such that B ≠ {} and |{p_j ∈ P | B = aux[j]}| ≥ n − f do
    release-coin with tag round
    wait for output-coin(s) with tag round
    round ← round + 1
    if exists b such that B = {b} then                          // i.e., |B| = 1
        if b = s then
            output rbc-decide(b)
        invoke bv-broadcast(b) with tag round                   // propose b for the next round
    else
        invoke bv-broadcast(s) with tag round // propose coin value s for the next round
    values ← [⊥]^n; aux ← [{}]^n
```

3 The problem and a solution

In the problematic execution [7], the network reorders messages between correct processes and delays them until the coin value becomes known. Our crucial insight concerns the coin: In any full implementation, it is not abstract, but implemented by a protocol that exchanges messages among the processes. Based on this, our solution consists of two parts.

Our *first* change is to assume FIFO ordering on the reliable point-to-point links, including the messages exchanged by the coin implementation. FIFO-ordered links are actually a very common assumption. They are easily implemented by adding sequence numbers to messages [1]. Our *second* change is to allow the set B (Alg. 2, line 25) to dynamically change while the coin protocol executes. Alg. 2 implements these changes. More details and correctness proofs appear in the full version [2].

Algorithm 2 Randomized binary consensus (code for p_i).

1: **State**
2: $\quad$ *round* $\leftarrow 0$: current round
3: $\quad$ *values* $\leftarrow \{\}$: set of *bv-delivered* binary values for the round
4: $\quad$ *aux* $\leftarrow [\{\}]^n$: stores sets of values that have been received in AUX messages in the round
5: $\quad$ *decided* $\leftarrow []^n$: stores binary values that have been reported as decided by other processes
6: $\quad$ *sentdecide* $\leftarrow$ FALSE: indicates whether p_i has sent a DECIDE message
7: **upon event** *rbc-propose*(b) **do**
8: $\quad$ **invoke** *bv-broadcast*(b) with tag *round*
9: **upon** *bv-deliver*(b) with tag r **such that** $r = round$ **do**
10: $\quad$ *values* $\leftarrow$ *values* $\cup \{b\}$
11: $\quad$ send message $[\text{AUX}, round, b]$ to all $p_j \in \mathcal{P}$
12: **upon** receiving a message $[\text{AUX}, r, b]$ from p_j **such that** $r = round$ **do**
13: $\quad$ $aux[j] \leftarrow aux[j] \cup \{b\}$
14: **upon** receiving a message $[\text{DECIDE}, b]$ from p_j **such that** $decided[j] = \perp$ **do**
15: $\quad$ $decided[j] = b$
16: **upon exists** $b \neq \perp$ **such that** $|\{p_j \in \mathcal{P} \mid decided[j] = b\}| \geq f + 1$ **do**
17: $\quad$ **if** $\neg sentdecide$ **then**
18: $\quad\quad$ send message $[\text{DECIDE}, b]$ to all $p_j \in \mathcal{P}$
19: $\quad\quad$ *sentdecide* $\leftarrow$ TRUE
20: **upon exists** $b \neq \perp$ **such that** $|\{p_j \in \mathcal{P} \mid decided[j] = b\}| \geq 2f + 1$ **do**
21: $\quad$ *rbc-decide*(b)
22: $\quad$ **halt**
23: **upon exist** $|Q_i = \{p_j \in \mathcal{P} \mid aux[j] \subseteq values\}| \geq 2f + 1$ **do**
24: $\quad$ *release-coin* with tag *round*
25: **upon event** *output-coin*(s) with tag *round* **and** $\exists B \neq \{\}, \forall\, p_j \in Q_i : B = aux[j]$ **do**
26: $\quad$ $round \leftarrow round + 1$
27: $\quad$ **if exists** b **such that** $|B| = 1 \wedge B = \{b\}$ **then**
28: $\quad\quad$ **if** $b = s \wedge \neg sentdecide$ **then**
29: $\quad\quad\quad$ send message $[\text{DECIDE}, b]$ to all $p_j \in \mathcal{P}$
30: $\quad\quad\quad$ *sentdecide* $\leftarrow$ TRUE
31: $\quad\quad$ **invoke** *bv-broadcast*(b) with tag *round* // propose b for the next round
32: $\quad$ **else**
33: $\quad\quad$ **invoke** *bv-broadcast*(s) with tag *round* // propose coin value s for the next round
34: $\quad$ $values \leftarrow [\perp]^n$; $aux \leftarrow [\{\}]^n$

References

1. Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.
2. Christian Cachin and Luca Zanolini. From symmetric to asymmetric asynchronous byzantine consensus. *CoRR*, abs/2005.08795v3, 2021. `arXiv:2005.08795v3`.
3. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *Proc. ACM CCS*, pages 31–42, 2016.
4. Achour Mostéfaoui, Moumen Hamouma, and Michel Raynal. Signature-free asynchronous byzantine consensus with $t < n/3$ and $O(n^2)$ messages. In *Proc. PODC*, pages 2–9, 2014.
5. Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, 2015.
6. Michael O. Rabin. Randomized byzantine generals. In *Proc. FOCS*, pages 403–409, 1983.
7. Pierre Tholoniat and Vincent Gramoli. Formal verification of blockchain byzantine fault tolerance. In *6th Workshop on Formal Reasoning in Distributed Algorithms (FRIDA'19)*, 2019.