Consensus Byzantin Responsable Optimal

Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic

Consensus Byzantin Responsable Optimal

Pierre Civit[1] et Seth Gilbert[2] et Vincent Gramoli[3,4] et Rachid Guerraoui [4] et Jovan Komatovic [4]

[1] *Sorbonne University, CNRS, LIP6*
[2] *National University of Singapore*
[3] *University of Sydney*
[4] *EPFL*

Il est connu que la propriété d'accord du problème du consensus byzantin entre n processus peut être violée dans un système non-synchrone si le nombre t de processus défectueux dépasse $t_0 = \lceil n/3 \rceil - 1$ [DLS88]. Dans cet article, nous étudions le problème du consensus byzantin *responsable* (*accountable*) qui se doit de (i) résoudre le consensus byzantin dans le scénario optimiste où $t \leq t_0$ et (ii) de permettre aux processus corrects d'obtenir une preuve indéniable de culpabilité de $t_0 + 1$ processus fautifs dans le pire des cas, où des processus corrects sont en désaccord. Nous présentons ainsi $\mathcal{ABC}$: une transformation simple mais efficace de tout protocole de consensus byzantin en une version responsable. $\mathcal{ABC}$ introduit un surcoût de (1) seulement deux tours de communication et $O(n^2)$ bits échangés supplémentaires dans le scénario optimiste, et (2) trois tours de communication et $O(n^3)$ bits échangés supplémentaires sinon. En appliquant notre transformation à un algorithme de consensus byzantin quadratique (il en existe et sont optimaux en complexité en bits échangés), le résultat obtenu est prouvé optimal.

Mots-clefs : responsabilité, détection, fautes Byzantines, incitatifs, systèmes distribués

1 Problem Description

Consensus problem [PSL80] is one of the most studied problem in distributed computing. This task consists in having a set $\Psi = \{P_1, ..., P_n\}$ of n processes, communicating by exchanging messages through a non-synchronous [†] reliable point-to-point network, that have to agree on a common value. That is, every process i proposes a value and eventually decides (at most once) a value s. t. the following properties hold:

— (Liveness) *Termination:* Every correct process eventually decides.
— (Safety) *Agreement:* All correct processes decide the same value.
— (Safety) *Validity:* [‡] If all processes are correct and propose the same value, only that value can be decided by a correct process.

It has been shown [DLS88] that no non-synchronous algorithm can ensure the conjunction of the aforementioned properties if the number t of Byzantine processes (that can perform arbitrary faults, deviating from their prescribed protocols) exceeds the threshold $t_0 = \lceil n/3 \rceil - 1$, even if an idealised PKI (digital signatures) is assumed, as we do, where each message sent by process p is properly authenticated. Furthermore, it is impossible [CGG21] to ensure a gracefully degraded version that would ensure (i) Safety and Liveness for $t \leq t_0$ and (ii) Safety only for $t > t_0$.

This limitation motivated researchers to investigate *accountable* variants of this problem [CGG21, SWN+21] that consist in (1) solving consensus when $t \leq t_0$, and (2) allowing all correct participants to eventually obtain undeniable proof of culpability against $t_0 + 1$ Byzantine processes in case of safety violation ($t > t_0$). This second property is slightly different from the *completeness*, ensured by Peer-Review [HKD07], where every Byzantine process is eventually suspected forever by at least one correct process. We say that a set

†. Non-synchronous systems include: asynchronous systems, where the bound on message delivery time does not exist, and partially synchronous systems [DLS88], where the (potentially unknown) bound holds only eventually after a (potentially unknown) global stabilization time (GST).

‡. this property is often called weak validity w.r.t. alternative properties like external validity or strong validity. The choice of such a validity property does not impact this work.

Base Consensus Protocol	Communication Complexity of the Base Consensus Protocol	Communication Complexity of the Accountable Variant in the Common Case	Accountability Threshold	Paper
HotStuff [YMR+19]	$O(n^3)$	$O(n^3)$	$2n/3$	[SWN+21]
Binary DBFT [CGLR18]	$O(n^3)$	$O(n^4)$	n	[CGG21]
Multivalue DBFT [CGLR18]	$O(n^4)$	$O(n^4)$	n	[CGG21]
Any	X	X	n	**this paper**

TABLE 1: Overview of the main properties of existing accountable Byzantine consensus protocols. We consider worst-case communication complexities (after Global Stabilisation Time (GST) [DLS88]) in all columns. Worst-case communication complexity (after GST) of a Byzantine consensus protocol is bounded by the $O(n^2)$ Dolev-Reischuk bound. The accountability threshold is the number of tolerated Byzantine processes to provide accountability.

$\mathcal{S}$ of properly authenticated messages sent by a process P_i is a *proof of culpability* of P_i if there does not exist an execution e of the system where (1) P_i sends all the messages from the $\mathcal{S}$ set, and (2) P_i is correct. Observe that a proof of culpability of a process contains messages signed by the process with its PKI private key. Indeed, the PKI private key of a correct process is *never* revealed

Accountability in distributed systems is important since it discourages bad behavior. If malicious behavior is guaranteed to result in apprehension and punishment, malicious processes are much less likely to carry out an attack in the first place, thus strengthening the security of the system.

All previous works employed sophisticated mechanisms for obtaining accountability instead of treating the base consensus protocol as a "black box", thus obtaining simpler and more efficient accountable Byzantine consensus protocols. table 1 compares accountable Byzantine consensus protocols obtained by $\mathcal{ABC}$ with the existing alternatives. The metric used in table 1 is the *communication complexity* which is the maximum number of signed messages sent by all correct processes after GST combined across all executions of the system.

The generic transformation $\mathcal{ABC}$ proposed by this paper owns its simplicity and efficiency to the observation that the composition presented in algorithm 1 solves the Byzantine consensus problem when $t \leq t_0$. Indeed, if the number of faults does not exceed t_0, all processes eventually broadcast and receive $n - t_0$ matching CONFIRM messages. However, the important mechanism illustrated in algorithm 1 is that faulty processes *must* send conflicting CONFIRM messages in order to cause disagreement. Indeed, a disagreement between two correct processes P_i and P_j that would respectively decide v_i and v_j with $v_i \neq v_j$, would imply the reception of $n - t_0$ [CONFIRM, v_i] messages by P_i and $n - t_0$ [CONFIRM, v_j] messages by P_j. So at least $2(n - t_0) - n = t_0 + 1$ Byzantine processes sent both [CONFIRM, v_i] and [CONFIRM, v_j]. Hence, whenever correct processes disagree, they only need to exchange received CONFIRM messages to obtain accountability in the form of $t_0 + 1$ pairs of conflicting signed CONFIRM messages with different values v_i and v_j from as many Byzantine processes. Indeed, a correct process never sends two conflicting CONFIRM messages since the original Byzantine consensus protocol allow him to decide at most only once.

Algorithm 1 Intuition Behind Transformation

```
function propose(v) do
    ▷ bc is any Byzantine consensus protocol
    v' ← bc.propose(v)
    broadcast [CONFIRM, v']
    wait for n − t0 [CONFIRM, v']
    return v'
```

Roadmap We devote remaining sections to our transformation. Specifically, we first introduce the novel accountable confirmer problem (section 2), the crucial building block of $\mathcal{ABC}$. Then, we present $\mathcal{ABC}$ and state its correctness and complexity (section 3). Finally, we conclude the paper in section 4.

2 Accountable Confirmer

The accountable confirmer problem is a distributed problem defined among n processes. The problem is associated with parameter $t_0 = \lceil n/3 \rceil - 1$ emphasizing that some properties are ensured only if the number

of faulty processes does not exceed t_0. Accountable confirmer exposes the following interface: (1) request *submit*(v) - a process *submits* value v; invoked at most once, (2) indication *confirm*(v') - a process *confirms* value v'; triggered at most once, and (3) indication *detect*$(F, proof)$ - a process *detects* processes from the set F such that *proof* represents a proof of culpability of all processes that belong to F; triggered at most once. The following properties are ensured:

— *Terminating Convergence:* If the number of faulty processes does not exceed t_0 and all correct processes submit the same value, then that value is eventually confirmed by every correct process.
— *Agreement:* If the number of faulty processes does not exceed t_0, then no two correct processes confirm different values.
— *Validity:* Value confirmed by a correct process was submitted by a correct process.
— *Accountability:* If two correct processes confirm different values, then every correct process eventually detects at least $t_0 + 1$ faulty processes and obtains a proof of culpability of all detected processes.

Terminating convergence ensures that, if (1) the number of faults does not exceed t_0, and (2) all correct processes submit the same value, then all correct processes eventually confirm that value [§]. Agreement stipulates that no two correct processes confirm different values if the system is not corrupted (even if submitted values of correct processes differ). Validity ensures that any confirmed value is submitted by a correct process. Finally, accountability ensures detection of $t_0 + 1$ faulty processes by every correct process whenever correct processes confirm different values.

Implementation We now give a naive and very informal implementation of the accountable confirmer problem that suffers from a $O(n^3)$ complexity even when $t \leq t_0$. An optimization obtaining a $O(n^2)$ complexity when $t \leq t_0$ is available in long version of the paper [CGG+]. [¶].

Each process initially broadcasts the value it submitted in a SUBMIT message: the SUBMIT message contains the value signed with the PKI private key of the sender.

Once a process receives such a SUBMIT message, the process (1) checks whether the message is properly signed, (2) verifies the signature, and (3) checks whether the received value is equal to its submitted value. If all of these checks pass, the process stores the entire message with its signature. Once a process stores signatures from (at least) $n - t_0$ processes, the process combine the received signatures into a *full certificate* which is simply the concatenation of the submitted value and the $n-t_0$ associated signatures. Then, the process confirms its submitted value and finally informs other processes about its confirmation by broadcasting the full certificate.

Finally, once a process receives two conflicting full certificates, the process obtains a proof of culpability of (at least) $t_0 + 1$ faulty processes, which ensures accountability.

Algorithm 2 $\mathcal{ABC}$ Transformation - Code For Process P_i

```
1:  Implements:
2:      Accountable Byzantine Consensus, instance abc
3:  Uses:
4:      ▷ Byzantine consensus protocol to be transformed
5:      Byzantine Consensus, instance bc
6:      Accountable Confirmer, instance ac
7:  upon event ⟨abc, Propose | proposal⟩ do
8:      trigger ⟨bc, Propose | proposal⟩
9:  upon event ⟨bc, Decide | decision⟩ do
10:     trigger ⟨ac, Submit | decision⟩
11: upon event ⟨ac, Confirm | confirmation⟩ do
12:     trigger ⟨abc, Decide | confirmation⟩
13: upon event ⟨ac, Detect | F, proof⟩ do
14:     trigger ⟨abc, Detect | F, proof⟩
```

Indeed, each full certificate contains $n-t_0$ properly authenticated messages: every process whose message is in the two conflicting full certificates is faulty and these messages represent a proof of its misbehavior (recall that no faulty process *ever* obtains the PKI private key of a correct process).

This algorithm (resp. its optimization in [CGG+]) solves the accountable confirmer problem with:

§. Note that it is *not* guaranteed that any correct process eventually confirms a value if correct processes submit different values (even if the number of faulty processes does not exceed t_0).

¶. The optimization takes benefits from $(n - t_0, n)$ threshold signature scheme where the cost of asynchronous distributed key generation (ADKG) is assumed to be amortized.

— $O(n^3)$ (resp. $O(n^2)$) communication complexity in the common case, and
— $O(n^3)$ communication complexity in the degraded case.

3 BC + Accountable Confirmer = ABC

We now define our $\mathcal{ABC}$ transformation (algorithm 2), the main contribution of our work. $\mathcal{ABC}$ is built on the observation that any Byzantine consensus protocol paired with accountable confirmer solves the accountable Byzantine consensus problem.

Theorem 1 *algorithm 2 solves the accountable Byzantine consensus problem with $O(n^2)$ additional bits of information exchanged when $t \leq t_0$.*

Finally, we note that $\mathcal{ABC}$ does not worsen the communication complexity of any Byzantine consensus protocol. It is well-known that any protocol that solves the Byzantine consensus problem incurs quadratic communication complexity due to the Dolev-Reischuk lower bound [DR85]. Given the fact that accountable confirmer has quadratic communication complexity in the common case, every Byzantine consensus protocol *retains* its complexity after our transformation.

Corollary 1 *Let Π be a Byzantine consensus protocol with the communication complexity X_Π. Let Π^A be a protocol obtained by applying $\mathcal{ABC}$ to Π. Then, Π^A solves the Byzantine consensus problem with the communication complexity X_Π.*

4 Conclusion

We presented the first generic transformation to obtain optimal Accountable Byzantine Consensus. This transformation can naturally be applied to randomized consensus, reliable-broadcast and consistent-broadcast.

References

[CGG+] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as abc: Optimal (a)ccountable (b)yzantine (c)onsensus is easy! In *36th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022*.

[CGG21] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS'21)*, Jul 2021.

[CGLR18] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. Dbft: Efficient leaderless byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the Association for Computing Machinery, Vol. 35, No. 2, pp.288-323*, 1988.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[HKD07] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. *SOSP'07*, 2007.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[SWN+21] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *Computer and Communication Security (CCS)*, Nov 2021.

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.