Proof-of-Decision: Achieving Fast and Timeout-Resistant Consensus in Asynchronous Byzantine Environments *

Aleksander Kampa
ak@sikoba.com

Sikoba Research

September 2019

Abstract

In Asynchronous Byzantine environments, the main difficulty is dealing with muteness failures. By spawning multiple single-sender consensus processes in parallel, and introducing the concept of *Proof-of-Decision* to prevent premature timeout messages form Byzantine nodes, we achieve efficient consensus that is very fast under favourable circumstances.

1 Introduction

The consensus problem is fundamental to distributed computing. It is not a single problem, but rather a class of related problems which can differ quite widely in their assumptions, constraints and goals. Reaching consensus when all nodes can propose different values is not the same as reaching binary consensus.

To solve consensus in the context of real-world distributed system, such as a blockchain, one approach is to use the full power of available cryptographic and consensus primitives. This is the path chosen in "The Honey Badger of BFT Protocols" [MXC+16], which makes heavy use of threshold cryptography to temporarily hide proposals, and also for its cryptographic common coin. Other primitives used are Reliable Broadcast and Common Subset Agreement. It is an impressive achievement, which comes at a cost of significant complexity as well as some limitations as to scalability.

The goal of this paper is to present a simpler approach, suitable for situations where the validity of proposals can easily be checked. The focus is on simplicity and on achieving fast consensus in favourable circumstances.

*Research supported by Fantom Foundation

2 Definitions

For the definitions of most terms used in this paper, please refer to [Kam18]. An additional definition is given below.

In a traditional setting, all nodes propose a value and the goal of the consensus is to agree on one of the proposed values. To deal with situations where only one node proposes values, we introduce the following definition:

Single-Sender Consensus Protocol A protocol allowing all correct nodes in a network to decide a single value when only one well-identified node (the *proposer*) can propose values which must be different from the special value 0. The value 0 is used to signal the decision to reject the proposal. Despite possible faults, the protocol satisfies the following conditions:

- *Agreement*: all correct nodes decide the same value;
- *Determinism*: if the proposer proposes only one value v, then this will be the decision value;
- *Validity*: a correct node can only decide a value proposed by the proposer, or 0;
- *Termination*: all correct nodes eventually (with probability 1) decide a value.

3 Model

3.1 Consortium Blockchain

We place ourselves in the context of a *Consortium Blockchain*, in which all nodes are well known to each other and communicate using cryptographically signed messages. Therefore, nodes cannot impersonate each other.

We assume a *Weakly Byzantine Environment* [Kam18]: there are n nodes, of which at most t can be faulty, the rest being honest. Of the faulty nodes, at most t' can be Byzantine.

The nodes communicate over a *Reliable Asynchronous Network*: all messages sent eventually get delivered, but with arbitrary delays.

Each node consists of a *Consensus Engine* and a *Deterministic State Machine* which are identical on all honest nodes. The role of the consensus engine is to communicate with other nodes and agree on ordered sets of transactions to submit to the state machine. It may also perform administrative tasks, such as seeking consensus on accepting new nodes and on rating or suspending nodes. The role of the state machine is to execute the transactions received from the consensus engine.

3.2 Valid transactions

A valid transaction is one that meets a set of criteria for being accepted for processing by the state machine. First of all, the transaction needs to be correctly formatted and signed. Other criteria may be related to the account sending the transaction, such as:

- Consistency of tx nonce;
- Minimum amount of tokens in sender account;

- Maximum transaction volume per time period not exceeded, based for example on tokens held.

A transaction set is valid if it contains only valid transactions, and invalid otherwise. An honest node can easily verify if a transaction or a transaction set is valid or invalid.

Note that after having been submitted to the state machine, a valid transaction may still fail to be executed. In other words, validity does not imply successful execution.

3.3 Block-Based Consensus and Proposed Values

We are in a context of block-based consensus, where the nodes aim to reach a common value in successive rounds, which are also called blocks. Rounds or blocks are identified by a unique sequential number.

During a round, one or more nodes are expected to submit proposals that contain a set of transactions to process. A proposal will typically be a data structure with the following content:

- p_k - sending node
- r - round or block number
- tr - proposal content, which is a (possibly empty) set of transactions
- Signature
- Hash

A proposal will be denoted $INIT_k(r, tr)$. When there is no risk of confusion, this may be written as $INIT_k(tr)$ or simply $INIT$.

4 Types of Failures

In our model, the adversary is given extensive powers over message delivery and over the behaviour of faulty nodes. Let's focus on three specific types of adversarial action.

4.1 Message Relay Failures

The adversary can make faulty nodes ignore certain incoming transactions, refuse to relay them and/or not include them in their proposal sets. There are several defenses against this:

- Users wishing to submit a transaction can send it to multiple nodes. For example, if at most 25% of nodes are faulty, then the probability that 5 randomly chosen nodes are all faulty is below 0.1%. Here, we assume an efficient gossip protocol such that if one honest node receives a transaction, we can be sure that all honest nodes eventually receive it.
- Users submitting transactions can request a signed acknowledgement of receipt: if none is received within a certain time period, they can resubmit to another node. This will ensure that transactions are not submitted to crashed nodes, and provides evidence that a transaction was submitted to a node.

- Users can forward acknowledgements of receipt to some other random nodes, or to special-purpose observer nodes, to monitor possible cheating attempts. If the node that acknowledged receipt then does not include this transaction in its proposal set, this may become visible by comparing node timestamps (this will require additional research).
- Users (or some trusted relays) can monitor if the transaction is included in a transaction set and, if that does not happen promptly, resubmit their transaction.

This is as much as we will say about message relay failures in this paper.

4.2 Multiple Proposals

A node may attempt to send different proposals to different nodes. Because all proposals are signed, two different proposals from the same node indicate either Byzantine behaviour, or that the node's private key was compromised. Whatever the reason, an expulsion procedure of the offending node should be initiated.

At certain stages of the protocol, a node may also send different messages to different nodes. If an honest node obtains proof of such a behaviour, this should also result in an expulsion procedure.

While any consensus protocol will need to guard against multiple proposals, it is the easiest Byzantine behaviour to identify, and thus the least likely.

4.3 Muteness Failures

The adversary has the power to prevent faulty nodes from sending out message or proposals, or to delay such sending. This is the most difficult attack to defend against, as malicious delays are hard to distinguish from simple network delays.

5 Reaching Consensus on a Single Proposal

Assume that the node p_k broadcasts $INIT(p_k, r, tr_k)$ to all nodes. Under this assumption, when an honest node receives $INIT$, it can simply verify if the proposal is valid or not and decide to accept or reject it accordingly, knowing that all other honest nodes will come to the same conclusion. No additional steps are needed to reach consensus in this case.

What this means is that once all honest nodes know what the proposal is, consensus has essentially been reached. The difficulty is to actually reach agreement as to what the proposal is. Note that a node may also delay sending out proposals, or crash, in which no proposals will be sent.

To problem of reaching consensus on the proposal of a single node p_k can thus be reduced to answering the following two questions:

Has a proposal been sent?

If yes, what is the proposal?

Let's first deal with the second question, which is the easier one.

6 Single-Sender Consensus Protocols

In this section, we present three Single-Sender protocols.

6.1 SSC-1: Single-Sender Consensus without Timeout

We aim to reach agreement on the proposal sent by node p_k, the *proposer*, which is expected to broadcast an $INIT$ value. The proposer can be Byzantine and sent different $INIT$ values to different nodes, or fail to send $INIT$ values to some nodes.

Each node p_i manages the following local variables:

- REC_i whose initial value is $\perp$ and which holds the first proposal value received, either directly from the proposing node, or via another node.
- SET_i which is the set of REC values received from other nodes, including its own REC_i value.

An $INIT$ value sent by p_k corresponds to REC_k. The REC values being are simply signed copies of REC_k and cannot therefore be forged. We will say that two REC values are identical if they correspond to the same $INIT$ value.

During the consensus process, nodes will be sending messages to each other. Whenever necessary, we assume that messages are not only signed, but also include a proof. For example: when a node sends a REC message, it includes the original INT or REC message received as proof; in the case of a $PROPOSE_i(0)$ message, it must contain two valid REC_k and REC'_k values as proof; etc.

The procedure executed by every node p_i is the following:

Protocol SSC-1 ($n > 2t + t'$)

Step 1: wait to receive a first valid REC_j value (either an $INIT$ from the proposer, or a copy of such an $INIT$ via another node). Set REC_i to this value, initialise SET_i and and broadcast REC_i.

Step 2: collect up to $(n - t)$ valid REC messages, updating SET_i accordingly.

If at any stage a REC value different from REC_i is received, broadcast $PROPOSE_i(0)$ and move to Step 3.

If $(n - t)$ identical REC messages have been received, broadcast $PROPOSE_i(REC_i)$ and move to Step 3.

Step 3: collect up to $(n - t)$ valid $PROPOSE$ values.

If at any stage $t + t' + 1$ proposals with a non-zero value v are received, **decide** for that value and forward that decision to Step 4.

If at least one nonzero value v is received (but less than $t + t' + 1$), propose that value in Step 4.

If all $n - t$ proposals received were 0, propose 0 in Step 4.

Step 4: enter **Underlying-Consensus**, treating votes for any value other than 0 as votes for 1.

Lemma 1. If the proposer acts honestly, broadcasting a single $INIT$ value, then all honest nodes decide for that value in Step 3.

Proof: Because messages cannot be faked, and the Adversary can delay but not drop messages, all honest nodes will set the same REC message in Step 1. As a result, in Step 2 all honest nodes will receive $(n-t)$ identical REC messages and thus propose the same value. In Step 3, all honest nodes will thus receive $(n-t)$ identical proposals and therefore decide for that proposed value, because $n > 2t + t'$ implies $n - t \geq t + t' + 1$.

What this shows is that when the sender is honest, consensus can be achieved quickly. Now let's look at situations when the sender can be Byzantine.

Lemma 2. If the proposer is Byzantine, the Adversary can cause any number of honest nodes to PROPOSE(0) at the end of Step 2.

Proof: There are many ways the adversary can achieve one or more PROPOSE(0) from honest nodes, here is an approach using an "isolated node". We divide the set of honest nodes into three subsets:

- I - a single "isolated node", which will propose 0
- HO - a (possibly empty) set of honest nodes which will propose 0
- HV - a (possibly empty) set of honest nodes which will propose a nonzero value

Initially, the adversary makes the proposer to send INIT to all nodes except I, and delays all messages sent to I. The other nodes are left to exchange messages, with the adversary using its powers to influence message delivery to obtain the following situation: all nodes in HV have proposed a nonzero value, and all nodes in $H0$ have received $INIT$ exactly once.

At that point, the adversary causes the proposer to send INIT' to J, makes J deliver that $INIT'$ value to all nodes in $H0$ and also makes J receive the $INIT$ value from some node. This achieves the desired result.

Lemma 3. If two honest nodes propose a non-zero value in step 2, then it is the same value.

Proof: Consider an honest node A proposing a nonzero value in Step 2. This node must have received a set R(A) of $n-t$ REC messages containing identical values. Among these messages, the set RH(A) of messages sent from honest nodes has at least $n-t-t'$ elements. Consider another honest node B that received a set R(B) of $n-t$ REC messages containing identical values. For the sets RH(A) and R(B) to necessarily overlap, we need to have $|RH(A)| + |R(B)| - n > 0$. This is the case, because $(n-t-t') + (n-t) - n = n - 2t - t' > 0$. This means that at least one of the REC messages received by B is identical with a message received by A from a honest node. Therefore, all values received by B must be equal to that value.

Corollary. Lemma 3 applies to any nodes, honest or Byzantine, as we require that any proposal be accompanied by proof. A proposal for a non-zero value in Step 2 must therefore come with a set of $(n-t)$ REC values.

Lemma 4. If an honest node decides for a value in Step 3, then any other honest node will either also decide on that value in Step 3, or propose that value in Step 4.

Proof: It is straightforward to see that if a node receives $n-t-t'$ proposals with a non-zero value v in Step 3, then any other node can receive at most $n-t-1$ proposals with value 0 in that Step, and must therefore receive at least one proposal with a non-zero value. By Lemma 3, that non-zero value is the same for all nodes.

We again note that this Lemma 3 applies to any nodes, honest or Byzantine. Because proof is

required with every proposal, the only malicious action available to a Byzantine node is to delay sending a proposal.

Theorem 1. The SSC-1 protocol is not a valid Single-Sender Consensus Protocol, as it does not meet the Termination criterion.

Proof: When the proposer sends a value, the protocol does meet the four conditions of a Single-Sender Consensus Protocol. This results directly from Lemma 3, and from the fact that Underlying-Consensus is a valid Binary Consensus Protocol. However, when the proposer does not broadcast any value, the protocol fails.

6.2 SSC-2: Single-Sender Consensus with Timeout

We now modify the previous protocol to attempt to deal with timeouts. Honest nodes wait for some predetermined time Δ. If no $INIT$ or REC are received, they will set REC to the timeout value $\boxtimes$.

Protocol SSC-2 ($n > 2t + t'$)

Step 1: wait to receive a first valid REC_j value (either an $INIT$ from the proposer, or a copy of such an $INIT$ via another node). Set REC_i to this value, initialise SET_i and and broadcast REC_i.

If no valid REC was received after a time Δ, set REC_i to $\boxtimes$.

Step 2: collect up to $(n - t)$ valid REC messages, updating SET_i accordingly.

If at any stage a REC value different from REC_i is received, broadcast $PROPOSE_i(0)$ and move to Step 3.

If $(n - t - t')$ or more identical messages v were received, and thus at most t' messages $\boxtimes$, broadcast $PROPOSE_i(v)$.

Else (strictly more than t' messages $\boxtimes$ received), broadcast $PROPOSE_i(0)$.

Step 2BIS (only applicable if $n \leq 2t + 2t'$): collect up to $(n - t)$ $PROPOSE$ values.

If all values are identical and equal to v, broadcast $PROPOSE2_i(v)$

Else broadcast $PROPOSE2_i(0)$

Step 3: collect up to $(n - t)$ valid $PROPOSE$ values (or PROPOSE2 values if $n \leq 2t + 2t'$).

If at any stage $t + t' + 1$ proposals with a non-zero value v are received, **decide** for that value and forward that decision to Step 4.

If at least one nonzero value v is received (but less than $t + t' + 1$), propose that value in Step 4.

If all $n - t$ proposals received were 0, propose 0 in Step 4.

Step 4: enter **Underlying-Consensus**, treating votes for any value other than 0 as votes for 1.

Let's start with a rather obvious remark: if the proposer does not send any message, all honest nodes will eventually propose 0 before entering Step 4, and will therefore decide 0 in Step 4.

Lemma 5. We assume the proposer is Byzantine. For any integer $k > 1$, if $n \leq \frac{k}{k-1}t + 2t'$ then honest nodes can propose up to k different nonzero values in Step 2.

Proof: First, we note that if $n - t - t' \leq t'$, or $n \leq t + 2t'$, then the adversary can make every single honest node propose a different value. This is done by having the Byzantine proposer send as many different messages as necessary to the other Byzantine nodes, after which $n-t-t'$ Byzantine nodes send different values to each honest node.

Let's define hc-nodes as the set of honest and crash-prone nodes. We will assume that all crash-prone nodes actually act like honest nodes. To start with, we look at one honest node receiving $n - t - t'$ messages v in the following configuration:

- $n - t - 2t'$ hc-nodes proposing v)
- t' Byzantine nodes proposing v) votes for $0 : n - t - t'$
- t' hc-nodes proposing $\boxtimes$) votes for $\boxtimes : t'$
- delayed: t hc-nodes proposing other values

If we want $(k - 1)$ additional honest nodes to decide on other values, we can only use the t' Byzantine nodes, as well as the t hc-nodes that were blocked. The Byzantine nodes can change their message for each honest node, but the hc-nodes cannot. To maximise the number of values, we therefore need to divide the values sent by the t hc-nodes into $k - 1$ sets. So the condition becomes:

$$\frac{1}{k-1}t + t' \geq n - t - t' \tag{1}$$

This yields the desired result.

So we see that if $n > 2t + 2t'$, then honest nodes can propose only one value in step 2. However, if $n \leq 2t + 2t'$, more than two values are necessary. That is why we need Step 2BIS in that case.

Lemma 6. If two honest nodes propose a non-zero value in Step 2BIS, it is the same value.

Proof: Similar to Lemma 3.

Lemma 7. If an honest node decides for a value in Step 3, then any other honest node will either also decide on that value in Step 3, or propose that value in Step 4.

Proof: Similar to Lemma 4.

Theorem 2. The SSC-2 protocol is not a valid Single-Sender Consensus Protocol, as it does not meet the Determinism criterion.

Proof: Even if the proposer is honest, the Adversary can cause any number of honest nodes to PROPOSE(0) at the end of Step 2. This is because by delaying messages in Step 1, the adversary can make any number of honest nodes set REC to $\boxtimes$. As a result, the honest nodes may decide on 0.

In other words, when going from SSC-1 to SSC-2, we have achieved Termination but have lost Determinism. If we introduce a limit on the delays that the adversary can cause, we can avoid timeout messages by honest nodes and obtain the following result:

Theorem 3. The SSC-2 protocol is a Single-Sender Consensus Protocol if the adversary cannot

delay messages more than Δ.

6.3 SSC-3: Single-Sender Consensus with Timeout and Proof

We have seen how the Adversary can trigger timeout messages from honest nodes in protocol SSC-2 to defeat its purpose. We now introduce the protocol SSC-3, which introduces the concept of **proof**. This proof is simply some data item without which nodes are not allowed to send $\boxtimes$ (timeout) messages. The definition of this proof is not part of the protocol and must be added when implementing this protocol. We therefore define SSC-3 as follows:

Protocol SSC-3

This protocol is identical with protocol **SSC-2**, except that in Step 1, the value $\boxtimes$ can be sent only together with a **proof**.

In the next section, we will show how such a proof can help us solve our consensus problem.

6.4 Evalutation of the SSC protocols

We have seen that under favourable conditions, Single-Sender consensus can be achieved quite quickly, without the need to trigger Underlying-Consensus. When the proposer is honest, the number of messages sent in SSC-1 are given below. This count includes the messages that nodes send to themselves.

The proposer:

- the proposer broadcasts n INIT messages

All nodes:

- Step 1: receives $n - t$ and broadcast n REC messages
- Step 2: receives $n - t$ and broadcast n PROPOSE messages
- Step 3: receives $\leq n - t$, decide and broadcast n decisions

Under favourable circumstances, in which the adversary does not trigger timeouts, SSC-2 will need the same number of messages if $n > 2t + 2t'$, and an additional n messages if $2t + t' < n \leq 2t + 2t'$. This is again a good performance.

All nodes - additional messages in REC-2 if $n \leq 2t + 2t'$:

- Step 2BIS: receives $n - t$ and broadcast n REC messages

7 Multiple-Sender Consensus

7.1 Introduction

To overcome the limitations of SSC-1 and SSC-2, we take an approach suggested in [CGLR17] which consists in launching multiple consensus processes in parallel. In this case, we will launch multiple SSC-3 processes with a specific proof.

At the beginning of each consensus cycle, we assume there is a process that randomly selects an ordered group of nodes that will be the proposers. In [CGLR17] all nodes propose values, but are ordered in some random way that cannot be predicted in advance. This set of nodes should contain a sufficiently high proportion of honest nodes with a probability of essentially 1.

The process of choosing the random set of k proposing nodes for each block must be deterministic and well known to all nodes, here we simply assume it as a given.

7.2 The MSC Multiple-Sender Consensus

We spawn a separate SSC-3 consensus process C_i for each proposing node p_i with $i \in [1, m]$.

Every proposing node initialises an internal variable D to $\perp$. This variable will be set to the smallest $i \in [1, m]$ for which C_i has reached a decision for a nonzero value.

We define a proof as a set of $t + t' + 1$ identical nonzero proposals received in Step 3 of some C_j. Obtaining a proof is therefore equivalent to deciding in Step 3 of the protocol, which ensures that protocol C_j will decide for that nonzero value. We therefore name this proof **"Proof-of-Decision"**. Note that this proof is valid in all of the ongoing processes.

All nodes participate in all of the m consensus processes. Upon obtaining its first proof from process C_j, an honest node:

- sets D to j;
- stops participating in any processes $C_i : i > j$;
- sends ⧖ (together with its proof) to any processes $C_i : i < j$ for which it has not yet received any INIT or REC value in Step 1;

After that, the node keeps participating in all consensus processes $C_i : i < D$ that have not yet terminated. Whenever it reaches a decision on a nonzero value for a process C_k with $k < D$, it sets D to k and stops participating in any processes $C_i : i > D$.

The consensus process ends in either one of the two following configurations:

- $D = 1$;
- $D > 1$ and all processes $C_i : i < D$ have terminated with value 0

The result of the consensus process is the proposal of C_D.

It is easy to see that MSC meets the required criteria of Agreement, Unanimity, Validity and Termination.

Theorem 4. MSC is a Consensus Protocol.

Proof Outline: The group of proposing nodes contains a majority of honest nodes. Each of these will broadcast an INIT value and because of the proof, Byzantine nodes cannot prevent at least one process to reach a decision for a nonzero value. The ordered subset of processes which reach a nonzero value is thus not empty and has a smallest element.

7.3 Removal Procedure

At any stage of the consensus process, if an honest node sees evidence that the proposing node has sent more than one proposal, or that a node has send different REC values to two or more nodes, or similar Byzantine behaviour, it initiates a removal procedure. There are various ways to this, one is simply to introduce special removal transactions which can be added to proposal sets together with other transactions.

7.4 MSC Performance under favourable conditions

Under favourable conditions, in which there are only few faulty nodes, it is likely that process C_1 will often terminate with a nonzero value. In these cases, consensus can be reached with limited overhead and in three or four communication steps.

8 Limitations and Improvements

A node may attempt to cheat by excluding some transactions in its proposal. This is mitigated by the fact that this will only cause a delay of at most a few blocks. Another way to mitigate this is to always complete the process for the first several (say 3 or 5) processes, and take the union of all nonzero results.

We have assumed that all the nodes in the network participate in the parallel consensus processes in MPC. In a large network, we could instead have only a random subset of the nodes run consensus. We would just need to make sure that the probability that the number of crash-prone and Byzantine nodes in that subset meets certain criteria with very high probability.

To accelerate the process, we could imagine focusing on a subset of the lowest-ranked parallel consensus processes first. The likelihood that one of them will yield a nonzero decision will generally be quite high.

For the choice of a random set of proposing nodes, we could for example use a cryptographic common coin based on a threshold signature scheme. This is used in the Honey Badger protocol [MXC+16] for another purpose, namely to achieve binary Byzntine agreement, based on previously published approaches.

9 Conclusion

The MSC protocol presented in this paper is a simple, efficient and scalable protocol for blockchains in which the set of validator nodes is known at all times. It introduces the concept of "Proof-of-Decision" to defend against spurious timeout messages from Byzantine node.

It assumes the existence of a Binary Consensus Protocol, whose existence in the Weakly Byzantine scenario has not yet been proven. Therefore, Proof-of-Decision can currently only be applied in the traditional Byzantine scenario.

References

[CGLR17] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. *CoRR*, abs/1702.03068, 2017.

[Kam18] Aleksander Kampa. One-step consensus in weakly byzantine environments. `http://research.sikoba.com/`, 2018.

[MXC+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. *IACR Cryptology ePrint Archive*, 2016:199, 2016.