Fork Accountability in Tenderbake

Antonella Del Pozzo
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Thibault Rieutord
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Abstract

This work investigates the Fork Accountability problem in the BFT-Consensus-based Blockchain context. When there are more attackers than the tolerated ones, BFT-Consensus may fail in delivering safety. When this occurs, Fork Accountability aims to account for the responsible processes for that safety violation.

As a case study, we consider Tenderbake when the assumption on the maximum number of Byzantine validators – participants involved in creating the next block – does not hold anymore. When a fork occurs, there are more than one-third of Byzantine validators, and we aim to account for the responsible validators to remove them from the system. In this work, we compare three different approaches to implementing accountability in the case of a fork. In particular, we show that in the case of a fork, if we do not modify Tenderbake or we enrich it with a reliable broadcast communication abstraction, then we can account Byzantine processes only in particular scenarios. Contrarily, if we change Tenderbake such that the exchanged messages also carry extra information (which size is proportional to the duration of the current consensus computation), then we can account for Byzantine processes in all kinds of scenarios; however, at the cost of unbounded message size and unbounded local memory.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Blockchain, BFT-Consensus, Fork Accountability

Digital Object Identifier 10.4230/OASIcs.FAB.2022.5

Funding This work was founded by Nomadic Labs.

Acknowledgements The authors warmly thank Lăcrămioara Aştefănoaei, Eugen Zălinescu and Sara Tucci-Pieriogvanni for all the insightful discussions that improved the quality of this work.

1 Introduction

A Blockchain, as the name suggests, is a chain of blocks. Current Blockchain solutions are divided into blockchains with probabilistic finality and blockchains with immediate finality. The most known blockchains, Bitcoin [18] and Ethereum v1.0 [20], are based on the Proof-of-Work mechanism to decide on the next block to append, and in that case, they provide probabilistic finality. Once a block appears in the $i-th$ position of the blockchain, it will stay there with a probability that exponentially grows proportionally to the length of the chain extending it [14]. In the case of immediate finality, as in the case of Tendermint [6] and Tenderbake [1], we have that a new BFT Consensus instance is run to decide on the next block to append. Hence, once a block appears in the $i-th$ position it stays there forever. However, BFT Consensus works as long as, given a set of n committee members (or validators) in charge to decide for the next block, at most $f = n/3 - 1$ are affected by Byzantine failures (validators showing arbitrary behaviors). As long as this assumption holds, we guarantee that precisely one block is decided for each consensus instance. Contrarily, if the assumption is violated, the blockchain can experience forks (loss of safety) or interruption of block production (loss of liveness).

Nevertheless, it is ambitious to assume that at most f validators are affected by Byzantine failures in a long-running system. Especially if we think that in the current solutions, we potentially select a new committee for each new block, and there could be mistakes in this selection process from time to time. On the contrary, if we do not frequently change the committee members, then bribery attempts might occur. To this end, since most of the current selection mechanisms are based on the stake held and bonded by validators, the bonded stake of misbehaving validators can be slashed in case of provable misbehavior. It plays as a disincentive to misbehave. In such a context, producing proof of misbehavior is a key aspect in putting in place actions to disincentivize validators from misbehaving. Once we have them, such proofs can be used to perform slashing actions. The procedure of deriving non-reputable proofs of misbehavior is known as accountability [16].

In this work, we consider accountability when more than f processes in a committee are Byzantine faulty and produce a fork. In this case, we refer to it as the fork accountability problem. Similarly, as in [12], in case of loss of safety, we aim to identify at least $f + 1$ of the byzantine processes responsible for the fork that occurred. We consider as a case of study Tenderbake [1], a BFT-Consensus protocol for the Tezos Blockchain [15, 3] designed for an eventually synchronous system in which processes are equipped with bounded buffer and can experience messages loss before the beginning of the period of synchrony.

Our contribution is the following. First, we organize the different scenarios that can induce a fork in Tenderbake in a fork taxonomy. Secondly, we design and compare three accountability approaches: the first considers *Tenderbake* as it is and investigates in which scenarios we can achieve accountability; in the second approach, we show how to modify Tenderbake to enrich messages with the necessary information to achieve accountability but at the price of unbounded message size; in the third approach, consider an approach in the middle of the previous two. We add reliable communications (contrarily to the previous two approaches) and some extra information in the messages. Interestingly, we obtain the same results as in the first approach. In the rest of the paper, we replace the f standard notation with T to design the maximum number of tolerated Byzantine processes. We use f_i to design the actual number of Byzantine processes during the consensus instance i, such that $f_i < n$, where n is the number of processes in the committee.

Related Work. Distributed system research areas are more focused on failure detectors [10, 5] than accountability. While the failure detector aims to provide each system process with the identities of faulty processes to help them make progress in the computation, accountability aims to provide non-repudiable proofs that can be shared with other processes. It plays a crucial role in putting in place countermeasures against faulty processes in a distributed system. As pointed out in [2], if processes are accountable for their behavior, then rational processes have an incentive to behave correctly.

To the best of our knowledge, PeerReview [16] is one of the first works that proposed a general solution to have accountability in distributed systems. They developed a system called PeerReview that implements accountability as an add-on feature for any distributed system. Each process in the system records messages in tamper-evident logs; an auditor can challenge a process, retrieve its logs, and simulate the original protocol to ensure that the process correctly behaved. They show that in doing so, it is always possible to identify at least one Byzantine process (if some process acts in a detectable Byzantine way). The main issue is that an auditor, to prove that a process is Byzantine, must receive a response from it. If no response is received, then the auditor cannot determine whether the process is Byzantine or whether the network has not yet stabilized. It follows that the Byzantine will

only be suspected forever but not proved guilty. This limitation is common to distributed protocols that are not designed to provide accountability. That is also true in the blockchain context, we observe that protocols as Tenderbake [1] and Tendermint [17] suffers from the same problem. Indeed, even though it is possible to observe deviation from the protocols, it might be necessary to prove the absence of messages to prove a party is guilty of bad behavior. For example, in a protocol, a message m is triggered on the reception of message m' from a quorum of processes. If m is triggered without the reception of sufficiently many occurrences of m', then the only way to prove that m is an expression of bad behavior is to prove that the causally dependent messages m' do not exist.

Polygraph [12] is the first work that manages to have accountability in the case of Byzantine behaviors, circumventing this issue by using message justifications. Contrarily to PeerReview, they do not use an accountability module as an add-on feature for a distributed protocol; they designed a distributed protocol that provides accountability features by design. Intuitively, messages sent during the computation carry the necessary information to have accountability in the case of forks. In case of failures, there is no need to query processes to obtain proof of their innocence.

Unlike adding extra accountability information in the protocol's messages, as Polygraph does, another accountability approach is to add extra information directly into the blockchain itself. While not done for accountability purposes, Streamlet [9] creates blocks in each round of computation while providing finality only when particular conditions are met. In other words, in Streamlet processes add blocks to a blocktree, and the fact that a block is added to the tree does not mean that such block is final. This allows the blocktree to be formed of non-finalized concurrent branches keeping track of the evolution of the computation. For finalization, Streamlet applies a rule to the tree to select the finalized branch. As a side result, it has been showed that Streamlet provides accountability without the need to add extra information, either on chain or in the exchanged messages [4]. Indeed, if two concurrent branches manage to be finalized, then the faulty behavior can be traced back to conflicting blocks up to the responsible processes. The work in [4] provides a specification of Accountability for Streamlet. When we compare the information that processes keep locally, we observe that by default, nodes keep intermediary unconfirmed blocks contrarily to Tenderbake. This provides an interesting trade-off between the amount of information and the Accountability capabilities.

In the Proof-of-Stake blockchain context, accountability represents an important goal. The idea is that users that violate the protocol can be punished by confiscating their deposited stake (e.g., Casper [7]). Tendermint as well is going in that direction [1]. However, there is not yet a solution specification that exhaustively performs fork accountability, encompassing all scenarios. The work of Sheng et al. [19] aims at modifying existing blockchain BFT consensus protocols to enforce them with fork accountability. In particular they consider PBFT [8], HotStuff [21] and Algorand [11]. Interestingly, a very recent work, [13] presents how to turn any consensus protocol into an accountable one with two additional all-to-all communication steps. It is important to stress that in the case of Tenderbake, we have to face an extra difficulty given by the lossy nature of channels, which poses an extra challenge in collecting proofs in case of misbehavior compared to the existing works.

[1] The fork accountability was described in the Cosmos blog `https://v1.cosmos.network/resources/whitepaper#fork-accountability` as in Tendermint discussions `https://github.com/tendermint/tendermint/issues/4189`. In the Tendermint documentation, we can find an analysis of the kind of Byzantine misbehavior that we can observe with respect to the protocol execution `https://docs.tendermint.com/master/spec/light-client/accountability/#on-chain-attacks`.

Our contribution. In this document, we adapt the misbehavior analysis in the Tendermint context to Tenderbake [1], and we derive a fork taxonomy. Furthermore, we establish that with the current version of Tenderbake, we can only account for Byzantine processes for a subset of the possible fork typologies. Nevertheless, if we consider an extended version of Tenderbake enriched with message justifications, we can derive accountability conclusions with any kind of forks. The price to pay is the requirement of unbounded local memory due to the unbounded size of messages. This result makes a step forward in understanding the cost of Fork Accountability depending on the kind of protocol.

Paper organization. The work is organized as follows, in Section 2 we define the system model as an extension of the system model for Tenderbake [1]. Section 3 recalls how Tenderbake works, and Section 4 organizes the different kinds of forks in a fork taxonomy and discusses the difficulty of having accountability in different cases. In Section 5.1 we define the Fork Accountability problem and we discuss how to solve it in the following cases: if we can collect all exchanged messages before a fork (Section 5.2); if we consider Tenderbake as it is (Section 3); if we enrich Tenderbake with message justifications (Section 5.4) and finally; if, contrarily to Tenderbake in [1], we have reliable communications (Section 5.5). In Section 5.6 we compare and discuss the trade-offs between the accountability and the message costs of the presented approaches, and finally, in Section 6 we discuss future directions.

2 System model and definitions

In this work, we refer to Tenderbake as defined in [1]. Briefly, Tenderbake solves Dynamic Repeated Consensus, i.e., it executes consensus instances in sequence to produce an infinite output of decided values. Each consensus instance is executed by a (possibly) different committee of n processes. We assume that in the committee of processes running a consensus instance, T is the maximum number of Byzantine processes, and the number of committee members is at least $n = 3T + 1$. For each committee c_i, we refer to f_i as the number of Byzantine faulty processes present in such a committee. Additionally, for each committee c_i, if $f_i \leq T$ then we say that c_i is a *correct committee*, otherwise c_i is a *Byzantine-faulty committee*. Processes have access to digital signing and hashing algorithms. For simplicity, we assume that cryptography is perfect: digital signatures cannot be forged, and there are no hash collisions. Each process has an associated public/private key pair for signing, and processes can be identified by their public keys.

We consider the same assumptions as defined in the system model of Tenderbake [1]. In particular, we assume a partially synchronous system, in which after some unknown time τ (the global stabilization time, GST) the system becomes synchronous and channels reliable. That is, there is a finite *unknown* bound δ on the message transfer delay. Before τ, the system is asynchronous, and channels are lossy. We assume the presence of a best-effort broadcast primitive used by processes participating in a consensus instance. Broadcasting messages is done by invoking the primitive broadcast. This primitive provides the following guarantees: (i) integrity, meaning that each message is delivered at most once and only if some process previously broadcasts it; (ii) validity, meaning that after τ if a correct process broadcasts a message m at time t, then every correct process receives m by time $t + \delta$. This communication primitive is built on top of point-to-point channels, where exchanged messages are authenticated. When specified, we consider a reliable broadcast abstraction built on top of no lossy channels, which provides integrity, validity, and agreement. The

integrity property does not change, and contrarily to the best-effort broadcast, the validity became, if a correct process broadcasts a message m at time $t > \tau$, then it receives it. The agreement property guarantees that if a correct process receives m at time $t > \tau$, every correct process eventually receives it by time $t + \delta$.

In the following, we consider that processes are executing a protocol for solving Dynamic Repeated Consensus (DRC) as defined in [1]. Informally, given that processes have an infinite sequence of input values, DRC guarantees the following three properties: (i) progress: each correct process has an infinite sequence of output values; (ii) validity: for each correct process, the sequence of output values satisfies a predetermined predicate $\mathsf{isValid}()$; and (iii) agreement: at any time, for any two sequences of outputs at correct processes, one is the prefix of the other. In the rest of the document, we use the terms agreement and safety interchangeably, and the same with the terms progress and termination.

3 Tenderbake and Round-based BFT-Consensus protocols

In this section, we present the main structure of Tenderbake. We start with a general overview of a round-based BFT-Consensus protocol, and we give more details on the specifics of Tenderbake, which is at the base of DRC. For a complete description of Tenderbake we refer to [1]. Round-based BFT-Consensus protocols are executed by a set of n processes, such that one process per round is selected as proposer. The proposer is in charge to drive all the other correct processes to agree on the same value at the end of that round. If this is not the case, a new round begins with a new proposer. To avoid safety violation between one round and another, processes carry certain information from one round to another. Given communication delay, we could have that only some correct process decides for a value in a round r, which implies that all the other correct processes have to agree on that value in the subsequent rounds. In most cases such as Tenderbake and Tendermint, safety is preserved thanks to the locking mechanism.

In particular, Tenderbake works as follows. Each round is divided into three sequential phases: PROPOSE, PREENDORSE, and ENDORSE. During the PROPOSE phase, only the unique designated proposer proposes to the committee members a single value b (a block proposal), either a new value or one inherited from a previous round. During the PREENDORSE phase, a process preendorses b if b comes from the designated proposer and if the process is not already locked or if it is locked on an outdated value. During the ENDORSE phase, if processes receive a preendorsement from a quorum for b, they lock on it and endorse it. Finally, if processes receive an endorsement from a quorum for b, they decide b. Let us stress the role of the lock variable. Such variable is set to a value b when potentially there could be some processes about to decide on it, and it is set to another value $b' \neq b$ only when a process observes evidence that no correct process might have decided on b. Each process signs the messages it sends, and each message carries a value b associated with the round and the phase. In the same spirit, each proposed block b is labelled with the round and the proposer that proposes it and became decided when there exists a Quorum Certificate (QC) of messages labelled with the same round r and the phase ENDORSE from $2T + 1$ different processes that refers to b. Each block b is composed of a block header and a block payload. In this context, we are not interested in the content of a block payload. Each block has the pointer to its (unique) predecessor block in the block header. If there are two different decided blocks with the same predecessor, we have a fork (safety violation).

4 Fork Taxonomy

This section aims at analyzing how forks can occur when committees are byzantine. The result is a fork taxonomy whose purpose is twofold. First, it helps in designing an accountability module: to collect evidence of Byzantine activities and make them accountable for their actions (if possible). Second, it helps in understanding the impact of Byzantine committees. Nevertheless, it is also the first step to designing fork recovery strategies, which is out of the scope of this document.

4.1 Fork Taxonomy

In the following, we define the kind of forks that can occur when Tenderbake runs under the hypothesis of more than T Byzantine failures in some committee c_i, in particular, $T < f_i$. Notice that the same can be applied to other repeated consensus protocols based on the locking mechanism and rotating coordinator (proposer) paradigm. Let us briefly recall that each consensus instance proceeds in rounds. Each round has a different proposer, and specific information is carried by processes from one round to another, such as the locking variable, to prevent the safety violation. In such context, we distinguish two kinds of forks, Intra-round forks, when two or more valid blocks[2] are produced during the same round, and Cross-round forks, when two or more valid blocks are produced across different rounds due to the violation of the locking mechanism. Finally, we present the Cross-committee fork, which occurs when we allow multiple committee selections for the same height (for instance, to deal with the absence of valid block production). Let us remark that this fork cannot occur with the current version of Tenderbake, and we discuss it for completeness. Given a Byzantine committee c_i, we define the following kinds of forks that can occur. In particular, if we have $T < f_i \leq 2T$ then only safety can be violated (or liveness, but there are no forks), while if $2T < f_i \leq n$, then both safety and validity can be violated.

- **Intra-Round (IR) fork ($T < f_i \leq 2T$)**: the fork is produced during the same round, i.e., there are at least two valid blocks under the same proposer. All the blocks in this fork share the same committee, proposer, round, and the same height;
- **Full Byzantine Intra-Round (FBIR) fork ($2T < f_i \leq n$)**: the fork is produced during the same round without any needed participation from correct committee members. Notice that we could also have a non-compliant block with the application level in this case even though it is valid in the sense that it carries a CQ, e.g., a block proposed by a Byzantine that is not the current proposer for that round. (Validity property violation)
- **Cross-Round (CR) fork ($T < f_i \leq 2T$)**: the fork is produced during different rounds, i.e., there are at least two valid blocks proposed during two distinct rounds and potentially distinct proposers. All the blocks in this fork share the same committee and height.
- **Full Byzantine Cross-Round (FBCR) fork ($2T < f_i \leq n$)**: the fork is produced during different rounds without any needed action from correct committee members. Notice that in this case, as before, we can have the Validity property violation.

[2] A block is said to be valid if it comes with a Quorum Certificate (CQ) of $2T + 1$ different endorsement (votes, or precommit – depending on the protocol vocabulary) messages from the same round and height.

4.1.1 Discussion on accountability

Let us recall that if there is a fork induced under the same committee, then we can have two scenarios, IR and CR forks. Interestingly, let us consider the first version of Tenderbake [1], in the case of (FB) IR forks. We can produce accountability proofs for any IR forks, considering the blocks' information. However, in the case of CR, the information in the blocks is not enough. More details are below.

- IR-Fork. In this case, the proposer for the round is Byzantine and proposes more than one block such that correct processes endorse only the first one they are aware of, and Byzantines endorse both. Any block in the fork comes with a QC of $2T + 1$ endorsements. It follows that any pair of blocks share at least $T + 1$ endorsements and up to $2T$. Those are the accountability proof for at least $T + 1$ Byzantine committee members. The proposer is also trivially accountable, and any pair of blocks give the proof of the fork.
  - FBIR-Fork, in this case, we can apply the same reasoning as for the IR-Fork but contrarily to it, two blocks can share the totality of endorsements, up to $2T + 1$. In this case, one of the proposers might be correct (the round proposer if it proposed only one of the two blocks).
- CR-Fork. The fork is composed of blocks produced during different rounds. In this case, the fork is due to the locking mechanism violation, i.e., there is some correct process that locks for different valid blocks (instead of at most one), not being aware that a previously preendorsed and endorsed block was decided (collecting $2T + 1$ endorsements). In this case, accountability is not possible by solely using the block's information. Indeed, we have that pairs of valid blocks can share the validators that signed the endorsements in their QC. For such a reason, Tenderbake has to be modified to gather enough information to distinguish between correct committee members that endorsed multiple times from Byzantine committee members. We discuss those modifications in the next sections.
  - FBCR-Fork, in this case, the lock does not have to be violated. Byzantine processes can directly produce two valid blocks by endorsing a proposal (from a valid proposer or not). This case inherits the same limitation as the previous one.

5 Fork Accountability

In this section, we define the Fork Accountability problem. Hereafter, we provide a pedagogical solution with all the available information (i.e., messages). Later we move to the specific case of the restricted information available with Tenderbake to discuss the limitation of the accountability accuracy and completeness that we can get. We further design modifications to Tenderbake to satisfy Fork Accountability. Finally, we investigate how improved communication primitives impact Fork Accountability.

5.1 Fork Accountability problem definition

An Algorithm A (Tenderbake in our case) is modeled as a collection of n deterministic automata, where $A(i)$ specifies the behavior of process i. Computation proceeds in steps of this automata. In each step (i, m, A) a process i first receives a message m or accepts an input (internal or external event) and after it changes its states according to $A(i)$. Finally, i sends a message specified by $A(i)$ for the new state to processes or produces an event. Let $E(A)_i$ be an execution of A at process i as the sequence of steps executed by i. Finally, let $M(E(A)_i)$, for conciseness M_i, be the set of messages m received by process i during an execution of A.

We define the accountability module for Tenderbake in terms of completeness and accuracy properties. Such module takes as input the messages M_i received by a correct process i during the execution of Tenderbake. If a fork occurs, it outputs the faulty participants and proof of their responsibility in producing such a fork.

- (**Completeness**) if a fork occurs then at least $T + 1$ committee members are accountable as faulty;
- (**Accuracy**) no correct process is ever accountable as faulty.

The Completeness property is similar to the Accountability property of the Accountable Byzantine Agreement problem as defined in [12] which merges the Fork Accountability and the BFT-Consensus problems.

5.2 Fork Accountability with all messages

For pedagogical purposes, in the following, we describe how to perform accountability if a correct process has access to the whole set of messages exchanged during an execution i of Tenderbake, M_i. More in detail, given the occurrence of a fork, a process can collect all the messages exchanged between correct processes before that fork occurs, as if they have been transmitted after τ by reliable and timely communication abstractions. Let us remember that the protocol proceeds in rounds and each round is composed of three steps. During each step, each correct committee member sends at most one message.

In the case of an IR fork, the proposer is necessarily Byzantine. Indeed, for each round, processes consider only the proposer's proposed values, which is supposed to propose a single value. Hence if there is an IR fork, there are not sufficiently many Byzantine processes to issue a fork by themselves, then the proposer proposed at least two different values. Moreover, the committee is Byzantine as well because there must have been sufficiently many Byzantine committee members who endorsed twice on different block proposals. In that case, given a fork at round r, the Byzantine members are detected by selecting from M_i all the members that sent more than one message labeled with round r and phase PROPOSE (more than one block was proposed), and that sent more than one message labeled with round r and phase ENDORSE (Byzantine validators endorsed twice and for different blocks). In the case of an FBIR fork, it can happen that the selected proposer did not propose a block in the fork for that round. In such a case, it means that all processes in the QC are byzantine.

Before digging into the CR fork, let us first describe how it may occur, detailing the faulty flip-flopping scenario [3] which provokes a violation of the locking mechanism and may result in a fork. Interestingly, with such a fork, the information carried in the QC of the decided blocks is not enough to perform fork accountability. Let us first make some observations about Tenderbake:

- A correct committee member decides for a block b associated to a round r if it receives a QC, $2T + 1$ endorsements, for it.
- Each correct committee member endorses at most once during a round r, while byzantine-faulty committee member can endorse for an unbounded number of different blocks.
- When a correct committee member i endorses for block b at round r, it also locks for block b at round r.

[3] `https://docs.tendermint.com/master/spec/light-client/accountability/#flip-flopping`

- The same correct committee member i re-locks (and endorses) for another block b' if i receives a proposal for b' at round r_2 and i already received $2T + 1$ preendorsements for b' produced at round r_1, $r < r_1 < r_2$ (cf. Tenderbake [1]).

This means that, if i decided for b but the committee is Byzantine then it can exist some round $r_1 > r$ where there are $2T + 1 - f_i$ correct committee members that preendorse b' along with f_i Byzantine committee members, then, at round r_2, i can re-lock on b' and decide for it. Hence, the locking mechanism is violated. In that scenario, blocks b and b' can share endorsements signed by the same committee members, either correct or Byzantine. In that case, looking only at the QC of b and b' it is not possible to distinguish correct from Byzantine processes and to perform any fork accountability.

Let us now generalize the flip-flop scenario. Let us consider an execution $E(A)$ of Tenderbake and let B the sequence of all blocks that obtains a QC during $E(A)$. Let S_B be the set of $T + 1$ Byzantine committee members and let S_L be a set of T correct committee members that lock and re-lock on all the blocks in B, and finally, let S_U be the set of correct committee members that never lock.

- At round r_k block b is proposed. S_B and S_L pre-endorse b. Processes in S_L, contrarily to processes in S_U, receive the $2T + 1$ pre-endorsements for b and then endorse and lock on it. Processes in S_B can assemble the QC for the decided block b with the $2T + 1$ endorsements (T from processes in S_L and $T + 1$ from processes in S_B) and delay its diffusion for as long as they wish.
- At round r_{k+1} block b_1 is proposed. S_B and S_U pre-endorse it and processes in S_L do not, because already locked. Processes in S_L receive all the $2T + 1$ endorsements for b_1.
- At round r_{k+2} block b_1 is proposed again. This time, processes in S_L can pre-endorse b_1 along with S_B (because they get the $2T + 1$ pre-endorsements for b_1 during the previous round). Processes in S_L receive the $2T + 1$ pre-endorsements for b_1 and then endorse and lock it. Processes in S_B can assemble a valid block b_1 with the $2T + 1$ votes (T from processes in S_L and $T + 1$ from processes in S_B) and delay its diffusion for as long as they wish.

In this case, in M_i there might not be any committee member that sent more than one message during the same step. A correct committee member does flip-flopping from b to b_1 when it receives enough pre-endorsements for b_1 from a previous round (still greater than the round in which it locked on b). On the contrary, a Byzantine committee member can flip-flop without needing those messages that justify its action. Thus, in order to account Byzantine committee members processes must look for unjustified flip-flopping, i.e., a committee member that endorsed a block b at round r_k and preendorsed b_1 at round r_{k+2} without the existence of $2T + 1$ preendorsements for b_1 at a round r_{k+1}, with $r_k < r_{k+1} < r_{k+2}$. In the described scenario, the withheld QC for b at round r_k (that contains the endorsement messages labeled with r_k issued by processes in S_B) plus the preendorsement for block b_1 at round r_{k+1} issued by processes in S_U constitute a proof of processes in S_U performing a faulty flip-flop (as we discuss, a legal flip-flop needs at least three rounds). The origin of the faulty flip-flop stands in the lock violation of Byzantine processes. However, it might not always be so direct to detect it. In the next Sections, we will discuss how to generalize this approach such that it is always possible to find a proof. This analysis would be possible combining the QCs of the decided blocks and M_i if M_i contains all messages ever exchanged during the protocol execution. Unfortunately, this is not possible with Tenderbake given the message lossy nature of the communication channels before τ, the reason why after that, we discuss how information can be added to the QC to perform CR Fork Accountability.

5.3 Partial Fork Accountability with Tenderbake

In the case of Byzantine committee, Tenderbake can incur (FB)IR and (FB)CR forks. In the first case, as we already discussed, accountability is straightforward. The information that we are interested in a block b_i is the proposer and the quorum certificate (CQ) signatures that come with the round in which they were issued. We use the following notation: $\mathsf{sign}(block) = \langle proposer, \{member_1, \ldots, member_{2T+1}\}, round\rangle$. Concerning forks, we use the following terminology: we say that two[4] blocks b_i, and b_j is a fork if they are two blocks of the blockchain with the same parent.

In case of an invalid block in an FBIR or FBCR fork, that is, with a round that does not correspond with the block proposer, then the proposer and all committee members that endorsed the block are byzantine processes, that is, $\mathsf{sign}(b_j).\mathsf{members}$ and $\mathsf{sign}(b_j).\mathsf{proposer}$.

In the case of an IR fork, each correct committee member sends only one message per phase in each round. It follows that, for each two pairs of blocks b_i, b_j in a fork, the faulty processes results from the intersection of $\mathsf{sign}(b_i).\mathsf{members}$ and $\mathsf{sign}(b_j).\mathsf{members}$. Thus the proposer and at least f committee members are accountable for the fork.

In the case of a CR fork there are no blocks in the same fork sharing the same round and the intersection of $\mathsf{sign}(b_i).\mathsf{members}$ and $\mathsf{sign}(b_j).\mathsf{members}$ is different than $\emptyset$. However, this gives us no clue about the faulty processes. Let us consider the scenario described in the previous section concerning the faulty flip-flopping. This scenario originates in blocks having QC sharing the same $2T + 1$ signatures. We cannot distinguish among them which signatures belong to correct committee members (if any) and which to Byzantine ones.

To discern correct from Byzantine, we need to combine the information carried by blocks with the exchanged messages. Indeed, a correct committee member does flip-flop from b_i to b_j when it receives enough pre-endorsements for b_j from a previous round (still greater than the round in which it locked on b_i). In contrast, a Byzantine committee member flip-flops without having those messages that justify its action. Moreover, let us recall that no pre-endorsement messages are recorded in the block. Thus those messages need to be kept locally to perform accountability in case of a fork. Therefore, ideally, all messages exchanged during the computation need to be saved. However, committee members do not rely on reliable communication channels, and not all messages are diffused reliably before τ.

In the following, we describe different approaches to provide the accountability module with the necessary information.

5.4 Full Fork Accountability with piggyback Tenderbake

The idea of the solution is relatively simple. Processes produce justifications when locked on a value b and pre-endorse or endorse a different value $b' \neq b$; for shortness in the following, we say that processes do not follow their lock. Notice that being locked on a value b is a local event. However, in the case of a fork involving b we might observe an endorsement for b in a QC. Indeed, a faulty flip-flopping occurs after some processes ignored their lock and preendorsed another value incorrectly. The issue is determining which processes ignored their lock incorrectly from those who followed the protocol. To distinguish between the two, we propose to provide justifications for these preendorsements votes.

When a process is locked but preendorses another value that was later proposed, if correct, it furnishes a quorum of preendorsement messages for the given value. It forms a justification when a process is not following a lock value. All these justification sets are kept in memory

[4] We consider for simplicity the case in which two blocks compose forks, but it can be easily generalized.

and are added to preendorsement and endorsement messages. That is if a process p is locked at round i and does not follow its lock at round $i + k$, then the justifications set of the preendorsement (endorsement) message contains the first preendorsement quorum certificate for a round between $i+1$ and $i+k-1$. Given a fork composed of two blocks, the justification information and the two blocks' endorsement messages should make it possible to account for at least $f + 1$ byzantine processes. However, this information alone is not enough when the processes that signed the second block were not directly involved in the flip-flopping. Thus, the justifications must also contain transitive justifications. More into details, if p is induced to flip-flop thanks to the messages from q, then p re-transmits also the justifications for the message from q. Appendix A provides a detailed explanation of the intuition behind our solution. In particular, Appendix A.1 describes a first approach of piggybacking, and Appendix A.2 details which case the previous approach does not work and introduces the transitive justifications.

5.4.1 Description of the modification to Tenderbake

Figures 1–2 detail the Tenderbake specification as described in [1] plus the modifications for the piggybacking (in red in the pseudo-code). It is out of the scope of this work to explain the Tenderbake functioning. In the following, we provide details useful for the fork accountability. Nevertheless, an interested reader can find more details in [1]. Processes have an extra variable $justification_p$, a list of triples of the following type $\{LR : int, LV : prop, peQC : set\ of\ messages\}$. The purpose is to keep for each Locked Value LV and Locked Round LR at process p the justification $peCQ$ that allows the flip-flopping. $peCQ$ is compounded of $2T + 1$ pre-endorsement messages for the new locked value received after round LR and before the new locked round. Moreover, pre-endorsement and endorsement messages have an extra field $justify_p$ that is composed of a list of $peQC$ extracted from the third element of the triples in the list $justification_p$. Figure 3 describes the Fork Accountability module that given two decided blocks, returns the set of faulty processes accountable for that. The proof of their accountability is given by the blocks themselves. As auxiliary functions we define $\mathsf{round}(peQC)$ and $\mathsf{value}(peQC)$ that return respectively round and the value associated to the $peQC$ provided as input.

This solution pays the cost of coping with a lossy channel. This cost is in terms of space complexity as message size becomes unbounded with the justifications. This set increases each time a lock occurs during the execution, which can, unfortunately, happen an infinite number of times (it depends on the Byzantine strategy, they can make processes flip-flop infinitely many times).

For detection, we have that when two distinct blocks are produced, processes can compare the endorsements quorums to detect at least $f + 1$ byzantine processes. It is done as follows: Let b_1 be the block with the smallest associated round, and let b_2 be the one associated with the greatest round. The endorsement of b_1, i.e., $\mathsf{sign}(b_1).\mathsf{members}$, are then compared with a refinement of the endorsement of b_2, i.e., $\mathsf{sign}(b_2).\mathsf{members}$. The refinement consists of taking endorsements and replacing them with the justification corresponding to the smallest round greater than $\mathsf{sign}(b_1).\mathsf{round}$, if any.

In the particular case of an FBCR fork, we can have the case in which no correct committee members are involved in the fork and thus in any information carried by each block. In this case, indeed, each endorsement comes with a justification produced by other Byzantine committee members. The detection is then performed in the same way as for CR fork, looking at the first justification in b_2 inconsistent with the CQ information of b_1.

```
1  var justification_p = empty list

2  proc handleConsensusMessage(msg)
3    let type_q(ℓ, r, h, payload) = msg
4    if ℓ = ℓ_p ∧ h = h_p ∧ (r = r_p ∨ r = r_p + 1) then
5      if isValidMessage(msg)
6        messages_p := messages_p ∪ {msg}
7        updateEndorsable(msg)
8    else if ℓ > ℓ_p then
9      pullChain

10 proc updateEndorsable(msg)
11   if |preendorsements()| ≥ 2f + 1 then
12     endorsableValue_p := proposedValue()
13     endorsableRound_p := r_p
14     preendorsementQC_p := preendorsements()
15   else if type(msg) ≠ Preendorse then
16     (eR, eV, pQC) := endorsableVars(msg)
17     if eR > endorsableRound_p then
18       endorsableValue_p := eV
19       endorsableRound_p := eR
20       preendorsementQC_p := pQC

21 proc endorsableVars(msg)
22   let pQC = match msg with
23   | Propose_p(ℓ_p, r_p, h_p, (eQC, hu, eR, pQC)) → pQC
24   | Preendorsements(ℓ_p, r_p, h_p, pQC) → pQC
25   return (roundQC(pQC), valueQC(pQC), pQC)

26 proc filterMessages()
27   messages_p := messages_p \ {type(ℓ, r, h, payload) ∈ messages_p | r ≠ r_p}
```

Figure 1 Message management for process p during single-shot Tenderbake. In red the new lines added with respect to Tenderbake [1].

5.4.2 Correctness proofs

▶ **Lemma 1.** *Let b_1 and b_2 two decided blocks, if b_1 and b_2 are in the same fork, then* detection(b_1, b_2) *returns at least $T + 1$ processes accountable as faulty.*

Proof. First, consider that b_1 and b_2 are two blocks in an Intra-round fork, i.e., those blocks are decided at the same round. In this case Algorithm in Figure 3 returns the intersection of the endorsement quorums (line 64). Since $2T + 1$ distinct signatures compose each QC and correct processes sing just once per round, then the intersection of two quorums contains at least $T + 1$ distinct processes.

Now, let us consider that we have a Cross-round fork composed of two blocks decided during different rounds. Let us assume, w.l.o.g., that b_1 is the block associated with the smallest round. In this case Algorithm in Figure 3 returns as accountable the intersection of the quorum of endorsement messages from b_1 and the refinement (line 65) of b_2 relatively to the round of b_1 (line 60). All we need to show is that: the refinement also returns a quorum of processes; hence, the intersection with the QC of b_1 contains at least $T + 1$ processes.

```
28  PROPOSE phase:
29   if proposer(ℓ_p, r_p) = p then
30     u := if endorsableValue_p ≠ ⊥ then endorsableValue_p
31          else newValue()
32     payload := (headCertificates_p, u,
33                 endorsableRound_p, preendorsementQC_p)
34     broadcast Propose_p(ℓ_p, r_p, h_p, payload)
35   handleEvents()

36  PREENDORSE phase:
37   if ∃q, eQC, u, eR, pQC :
38      Propose_q(ℓ_p, r_p, h_p, (eQC, u, eR, pQC)) ∈ messages_p ∧
39      (lockedValue_p = u ∨ lockedRound_p ≤ eR < r_p) then
40      if ∃(LR_p, LV_p, ∅) in justification_p then
41        replace (LR_p, LV_p, ∅) in justification_p by (LR_p, LV_p, pQC)
42     broadcast Preendorse_p(ℓ_p, r_p, h_p, hash(u), justification_p.peQC)
43   else if lockedValue_p ≠ ⊥ then
44     broadcast Preendorsements(ℓ_p, r_p, h_p, preendorsementQC_p)
45   handleEvents()


46    ENDORSE phase:
47   if |preendorsements()| ≥ 2f + 1 then
48    if ∃(LR_p, LV_p, ∅) in justification_p then
49        replace (LR_p, LV_p, ∅) in justification_p by (LR_p, LV_p, preendorsements())
50     u := proposedValue()
51     lockedValue_p := u;  lockedRound_p := r_p
52     add (lockedRound_p, lockedValue_p, ∅) to justification_p
53     broadcast Endorse_p(ℓ_p, r_p, h_p, hash(u), justification_p.peQC)
54     broadcast preendorsementQC_p
55   handleEvents()
56   advance(getDecision())
```

Figure 2 Piggyback version of Tenderbake for process p. In red, the new lines added with respect to Tenderbake [1].

Hence, let us show that the refinement procedure returns a quorum (line 65). The procedure starts with a quorum of endorsements and replaces processes with their justification (as long as the justification is associated with a greater round than b_1). Hence, we replace a process in a quorum with a quorum of processes forming the justification. This ensures that we still possess a quorum after each modification and, therefore, that at the end, the refinement procedure returns a quorum. Consequently, the intersection of the quorum returned by the refinement procedure with the quorum of endorsement messages returns at least $T + 1$ distinct processes. ◀

In the next Lemma we show that the returned processes by Algorithm in Figure 3 are never correct, hence those are Byzantine.

▶ **Lemma 2.** *Given two blocks b_1 and b_2 being a fork, then* detection(b_1, b_2) *never returns a correct process.*

```
58 proc detection(b_1, b_2)
59   if sign(b_1).round < sign(b_2).round
60     return sign(b_1).members ∩ refinement(sign(b_2).members, sign(b_1).round)
61   else if sign(b_2).round < sign(b_1).round
62     return sign(b_2).members ∩ refinement(sign(b_1).members, sign(b_2).round)
63   else
64     return sign(b_1).members ∩ sign(b_2).members

65 proc refinement(QC, round)
66   if ∃e ∈ QC, ∃q ∈ justification(e), round(q) > round
67     return q such that ∃e ∈ QC, q ∈ justification(e), round(q) > round∧
68                        ∀e ∈ QC, ∀q' ∈ justification(e), round(q') ≥ round(q) ∨ round(q') ≤ round
69   else return QC
```

Figure 3 Fork Accountability module at process p.

The intuition of the following proof in the case of CR forks is the following. Let us observe that the refinement procedure takes as input the QC associated to b_2 and r_1 the decision round of b_1. It returns the smallest justification present in messages in b_2 QC associated with a round $r > r_1$ if any, and returns QC associated with b_2 otherwise. Intuitively, a process that endorsed a block cannot preendorse or endorse for later blocks without having a valid justification attached to its message (which differentiates a flip-flopping from a faulty flip-flopping). Therefore, such a process should never be returned by the refinement procedure. Indeed, a correct process being always justified cannot be in the smallest justification as it itself has a smaller justification attached to itself.

Proof. Let us also start with the more straightforward case of an Intra-round fork. In this case, Algorithm in Figure 3 returns the intersection of two endorsement quorums for the same round. It implies that we return processes that sent two distinct endorsement messages for the same round, necessarily a byzantine failure. A single endorsement message can be sent per round.

Now let us look at the case in which we have a Cross-round fork. Let us assume, w.l.o.g., that b_1 is the block associated with the smallest round. Assume now by contradiction that a correct process p is returned in the intersection. That is p endorsed for b_1 and was returned in the refinement of b_2 relatively to the round of b_1.

Since p endorsed for b_1 it must have added a justification item (that can be empty) for the round and value of b_1 in its justification set (line 52). From the protocol, we see that p can only send preendorsement or endorsement messages that are justified (with a non-empty set) by a quorum that does not include itself and is associated with a round greater r_1. Indeed, if p sends a prendorsements, then either it already has a justification or it adds the preendorsement quorum from the proposer that is associated with a round greater than r_1 (line 41). Similarly, if p sends an endorsement message then it adds the set of received preendorsements justifying its vote to its justification set if not already justified (line 49). Therefore, p cannot be returned by the refinement procedure of b_2 relatively to the round of b_1 as it always has a justification attached to it – A contradiction. ◀

Combining the two preceding Lemmas we obtain the following Theorem about the completeness and accuracy of our accountability detector:

Table 1 Comparison of the different presented approaches.

Approach	IR Fork	FBIR Fork	CR Fork	FBCR Fork	Extra message space complexity
Tederbake	Yes	Yes	No	No	No extra costs
Tenderbake fully just.	Yes	Yes	Yes	Yes	Unbounded msg space and local memory
Tenderbake over rb	Yes	Yes	No	No	Unbounded local memory

▶ **Theorem 3.** *For each fork at least $T + 1$ committee members are accountable as faulty and no correct committee member is ever accountable as faulty.*

As we discussed, the solution is quite communication-intensive. So far, it does not seem that it can be much improved without revisiting Tenderbake more deeply. A possible idea was to keep these justifications information locally at each process. The problem is that there is a need for synchronous assumptions to provide accountability for forks. Indeed, when the fork is observed, then processes must be able to provide their justification within a known delay to be ensured not to be wrongly suspected. In that case, since τ is unknown, it is not possible to distinguish a slow correct process from a byzantine process withholding its non-adequate justifications.

5.5 Accountability with Tenderbake over reliable broadcast

The previous approach suffers from unbounded size messages, which is necessary if we want to perform accurate and complete accuracy despite the unreliability of the communication mean. In this section, we investigate the advantages of leveraging reliable communications. Indeed, in this context, there is no need for messages to carry unbounded justifications. We consider light piggyback justification with parameterizable depth d. Committee members justify the last d pre-endorsement issued after the d previous endorsement (if any). The justification is a set of $2T + 1$ pre-endorsement that allowed them to flip-flopping. Those justifications are further carried by the *peQC* associated with each endorsement message.

In this case, we can detect Byzantine processes in a few cases:

- IR-Fork and FBIR-Fork: always.
- CR-fork: only if a process endorses for a value at round r for a block b_1 and pre-endorses for a value $b_2 \neq b_1$ at round $r + d$.

If $d = \infty$ then we boil down in the previous approach.

The main limitation of this approach is that even though we eventually receive all messages observed by correct processes, in case of missing justification for a flip-flopping, we do not know if those messages will arrive (exonerating the baker) or they do not exist at all (incriminating the baker).

5.6 Discussion

Table 1 depicts the different Fork Accountability approaches. The first four columns refer to the four kinds of forks that can occur with Tenderbake when more than T Byzantine are present in the committee: Intra-Round Forks; Fully Byzantine Intra-Round Forks; Cross-Round Forks; and Fully Byzantine Cross-Rond Forks. For each of those fork kinds, we

state if we are able to have Fork Accountability with the given approach. Finally, the last column presents the accountability solution's complexity to the Tenderbake complexity. Before digging into that, let us recall that if we have less than T Byzantine committee members, then we know that, after τ, the consensus instance terminates in $f + 2$ rounds [1]. Contrarily, termination property can be violated, Byzantine committee members can let the consensus run endlessly or end it with a fork at their will (violating the agreement property). Thus, there is no upper bound on the consensus instance duration. Indeed, the number of times a correct committee member can perform a flip-flopping is unbounded, which explains why Tenderbake with full justifications incurs the cost of unbounded message space and, consequently, unbounded local memory. However, if we consider that messages do not carry the total amount of justifications (thus, message space complexity is bounded), we still have that the total amount of messages is unbounded (due to the computation duration unbounded). It follows that Tenderbake with weak justifications, even though it gets rid of the unbounded message space, it still suffers the extra cost of the unbounded local memory. Unsurprisingly, reducing the message space complexity comes at the price of reducing the accountability capability. As a result, we obtain the same results as with Tenderbake unmodified, which is, on the light side, free of any extra costs. To complete this discussion, let us consider the Streamlet [9] case. As shown in [4], with few modifications to the protocol (in the way blocks are decided, with no impact on the message and space complexity), we obtain an Accountable protocol. In a nutshell, differently than Tenderbake, processes when observing that a new proposed block b collects a quorum of pre-endorsement messages, rather than locally locking on it, they add such block to their local structure (a block-tree rather than a chain). In that case, we say that b is notarized, which is different than decided. After, new blocks are proposed. When b is followed by other notarized blocks such that it meets a particular condition, b is decided. Intuitively, in Streamlet the lock mechanism is explicit on chain, so there is no need to add extra information to the blocks as in Tenderbake, and that would also be true if with lossy channels before τ (Indeed, after τ correct processes would successfully synchronize with other processes being able to retreive the missed notarized and decided blocks, yet losing the messages exchanged to get to the notarization of those blocks). For such a reason is, Streamelet we can perform Fork Accountability with no extra costs because those costs are already paid by the protocol as it is in terms of space occupied by the blockchain data structure.

6 Conclusion and Future Work

This work lays the groundwork for Fork Accountability solutions for BFT-like consensus algorithms, as Tenderbake. Even though the proposed solutions are either not fully solving the Fork Accountability problem or are impractical, we believe that is an essential step toward better understanding the Accountability possibilities and impossibilities in the Blockchain context. In future work, we aim to explore how to increase Accountability capabilities practically. A possible direction to explore is the redefinition of the BFT-Consensus algorithms with bounded buffers in the accountability lens.

References

1 Lăcrămioara Aştefănoaei, Pierre Chambart, Antonella Del Pozzo, Thibault Rieutord, Sara Tucci-Piergiovanni, and Eugen Zălinescu. Tenderbake - A Solution to Dynamic Repeated Consensus for Blockchains. In *4th International Symposium on Foundations and Applications of Blockchain 2021 (FAB 2021)*, pages 1:1–1:23, 2021.

2 Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, 2005.

3 Victor Allombert, Mathias Bourgoin, and Julien Tesson. Introduction to the Tezos Blockchain. In *Proc. High Performance Computing and Simulation*, 2019.

4 Emmanuelle Anceaume, Antonella Del Pozzo, Thibault Rieutord, and Sara Tucci-Piergiovanni. On Finality in Blockchains. In *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, pages 6:1–6:19, 2022.

5 Roberto Baldoni, Jean-Michel Hélary, and Sara Tucci Piergiovanni. A methodology to design arbitrary failure detectors for distributed protocols. *J. Syst. Archit.*, 54(7):619–637, 2008.

6 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, 2018.

7 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, 2017.

8 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 2002.

9 Benjamin Y Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains, 2020.

10 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

11 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 2019.

12 Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021.

13 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)*, 2022.

14 J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proc. EUROCRYPT International Conference*, 2015.

15 L.M. Goodman. Tezos – a self-amending crypto-ledger, 2014.

16 Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. In Thomas C. Bressoud and M. Frans Kaashoek, editors, *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*, 2007.

17 Jae Kwon and Ethan Buchman. Tendermint.

18 S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

19 Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1722–1743, 2021.

20 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

21 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proc. ACM Symposium on Principles of Distributed Computing*, 2019.

A Appendix – CR fork Scenarios

We describe two detailed scenarios dealing with the (faulty) flip-flopping case to justify the need to modify Tenderbake with justifications and their re-transmissions.

A.1 Scenario 1

In this first scenario, we consider a system composed of 7 processes, $p_0, \ldots, p_6$, among which p_4, p_5 and p_6 are byzantine. The execution comprises three rounds of communication and leads to a fork with the accountability of faulty processes.

Figure 4 depicts the first scenario. In the first round, p_0 proposes A (p A message at Round 1, phase p in Figure 4) and all processes pre-endorse A (pe A message at Round 1, phase pe in Figure 4) but only processes p_1 and p_2 endorse A among the correct processes (e A message at Round 1 represented by the lock image, phase e in Figure 4). As byzantine processes also endorse A, the block A is created with a Quorum Certificate composed by the endorsement messages from p_1, p_2, p_4, p_5 and p_6, but all such endorsements are received only by Byzantine processes (down-going black arrows Round 1, phase e in Figure 4) that withhold the block just created.

In the second round, p_3, which did not see the block A created, proposes a new block B. p_0 and p_3 pre-endorse it as they are not locked on A, along with byzantine processes that do not follow their lock. This set of pre-endorsements (that fulfills a quorum) is received by p_0 and p_2. The rounds end without receiving enough endorsements to form a quorum certificate.

In the last round, p_4 proposes B. This time B (being not new) has a valid round associated greater than 1, the round in which processes locked for A. Hence, p_2 can pre-endorse it along with p_0 (which is not locked) and all byzantine processes. This leads to a pre-endorsement quorum certificate that is received by p_1 and p_2. The round terminates by the endorsements of p_1 and p_2 along with the three endorsements for B by the byzantine processes, p_4, p_5, p_6, the same quorum certificate of block A. It leads to a fork once both blocks are diffused.

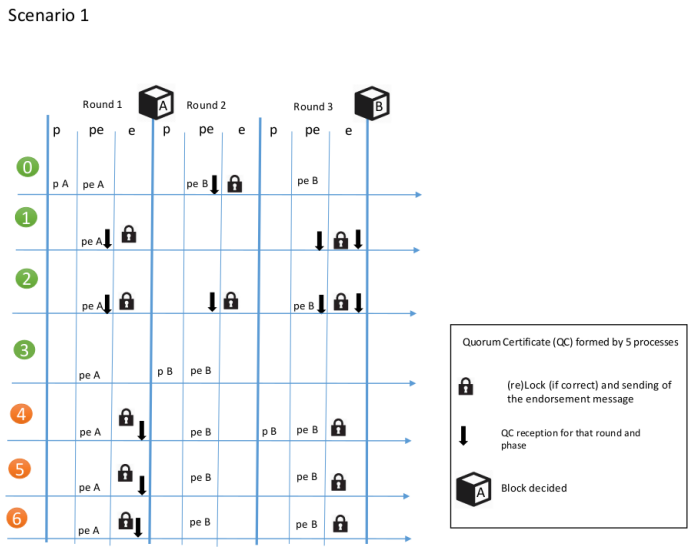

Figure 4 Scenario 1. We refer to the propose, pre-endorse and endorse phase with p, pe and e respectively. We refer to the proposal (pre-endorse) message for a value X with p (pe) X notation.

In this scenario, we can see that the byzantine misbehavior that led to the fork is that byzantine processes first endorsed for A at round 1 and pre-endorsed B just after, in round 2. It is highlighted in Figure 5. Indeed, a correct process locked on block A during round 1 would refuse to pre-endorse B in a later round $1 + k, k > 0$, unless it observes a quorum of pre-endorsement for B in a round $1 < r < 1 + k$. In this particular case, any correct process would never obtain a valid pre-endorsement quorum for B before round 2 to justify the pre-endorsement for B while locked on A.

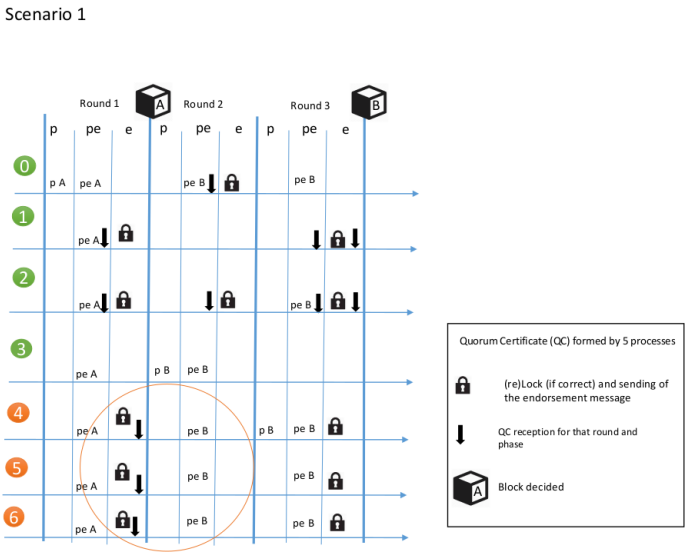

Figure 5 Example of how to detect Byzantine processes given a complete knowledge on the exchanged messages. In that case, p_4, p_5 and p_6 performs a faulty flip-flopping.

We consider a slightly modified version of Tenderbake. Processes produce justifications when locked on a value v and pre-endorse (endorse) a different value $v' \neq v$. That is, justifying their action by adding the pre-endorsement quorum certificate ($peQC$) for v' when sending a pre-endorsement message in the flip-flopping context. Note that endorsements carry all justifications issued during all rounds since the first lock round. Since those justifications chain are carried by endorsement messages then we collect them in decided blocks, that is, from processes which endorsed the block itself, p_1, p_2, p_4, p_5 and p_6 in this particular Scenario. Hence in this case, given the blocks A and B we collect information from correct processes p_1 and p_2 (Byzantine processes can omit justifications in their endorsement message).

- block A decided at round 1: such block comes with eQC, a quorum certificate of endorsements from p_1, p_2, p_4, p_5 and p_6 (first red rectangle in Figure 6), and none of them has justifications .
- block B decided at round 3: such block comes with a QC of endorsements from p_1, p_2, p_4, p_5 and p_6 (fourth red rectangle in Figure 6) and with the following justifications:
  - p_2 justifies its endorsement with $peQC$ for value B at rounds 2 (second red rectangle in Figure 6) and 3 (third red rectangle in Figure 6). Indeed, p_2 witnessed both the endorsement QC for block B in round 2 and round 3.

In this scenario, byzantine processes can be detected as their endorsement for A in round 1 (available from eQC for A round 1 in block A) implies that they should have set a lock for A at round 1. This, along with the pre-endorsement by the byzantine processes in round 2 (available from $peQC$ from B from round 2 and 3), implies a violation of the lock mechanism. Hence, we can detect $T + 1$ byzantine processes by simply gathering information available to the processes in the normal execution of Tenderbake once the blocks originating a fork are collected.

A.2 Scenario 2

In the following scenario, we first show that the previous scenario's accountability approach results in the violation of completeness and accuracy properties. We further show how to modify the accountability approach such that we are still able to detect and account for $T + 1$ Byzantine processes once a fork occurs.

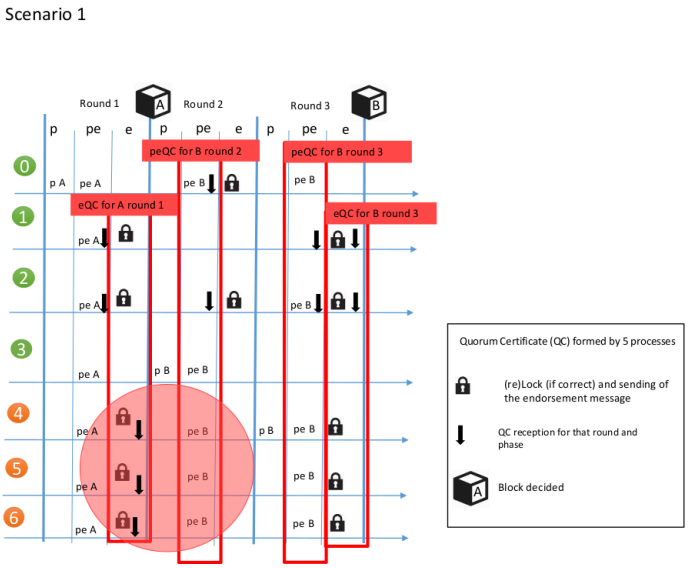

Figure 6 Justification gathered in decided block depicted using red rectangles along with highlight of detected byzantine process misbehaviour in light red.

The second scenario goes as follows and is depicted in Figure 7. During the first two rounds, we have the same execution as in Scenario 1. At the end of round 1, a block A is created with a Quorum Certificate composed of the endorsement messages from p_1, p_2, p_4, p_5, and p_6, but all such endorsements are received only by Byzantine processes that withhold the block just created. In the second round, p_3, which did not see the block A created, proposes a new block B. p_0 and p_3 pre-endorse it as they are not locked on A, along with byzantine processes that do not follow their lock. This set of pre-endorsements (that fulfills a quorum) is received by p_0 and p_2. The rounds end without receiving enough endorsements to form a quorum certificate.

Here, in the third and last round, we build the difference with the first scenario. p_4 proposes B as in Scenario 1. B has a valid round associated that is greater than 1 the round in which processes locked for A. Hence, p_2 can pre-endorse it along with p_0 (which is not locked at all) and all byzantine processes. This leads to a pre-endorsement quorum certificate that is received by p_1 and p_3 this time (in the first scenario it was p_2). The round terminates by the endorsements of p_1 and p_3 along with the three endorsements for B by the byzantine processes, p_4, p_5, p_6, a different quorum certificate of block A. It leads to a fork once both blocks are diffused.

If we use the same approach for the justifications as in Scenario 1, then we collect the information highlighted in Figure 8 and we might accuse p_2 of being Byzantine. Indeed, we get his endorsement at round 1 for A and his pre-endorsement for B at round 3, but not the justification for such pre-endorsement, that might look like a faulty flip-flopping. The same reasoning can be applied for the Byzantine processes p_4, p_5, p_6.

To solve this issue, we further modify Tenderbake. Processes produce justifications when locked on a value v and pre-endorse (endorse) a different value $v' \neq v$. This is done by adding the pre-endorsement quorum certificate ($peQC$) for v' when sending a pre-endorsement message in the flip-flopping context. Moreover, a process re-transmits all $peQC$ collected as justifications carried by pre-endorsement messages to other processes. That is, when sending a pre-endorsement message, it sends all justifications gathered so far. In this case,

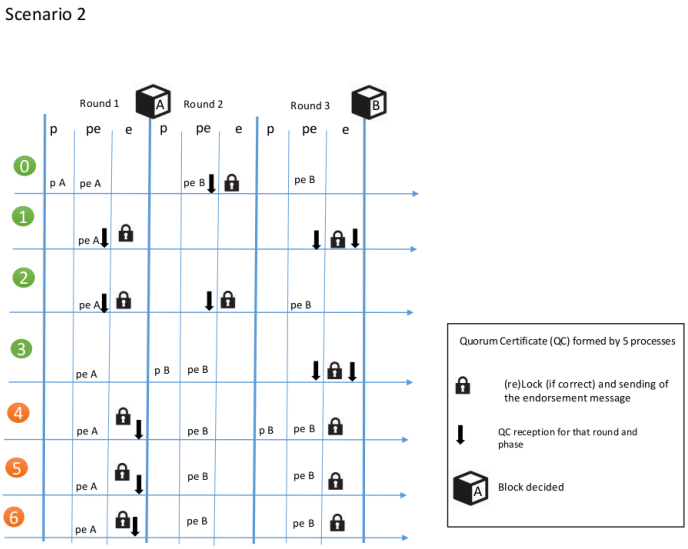

Figure 7 Scenario 2 representation. In particular we refer to the propose, pre-endorse and endorse phase with p, pe and e respectively. Moreover, when we refer to the proposal (pre-endorse) message for a value X with p (pe) X notation.

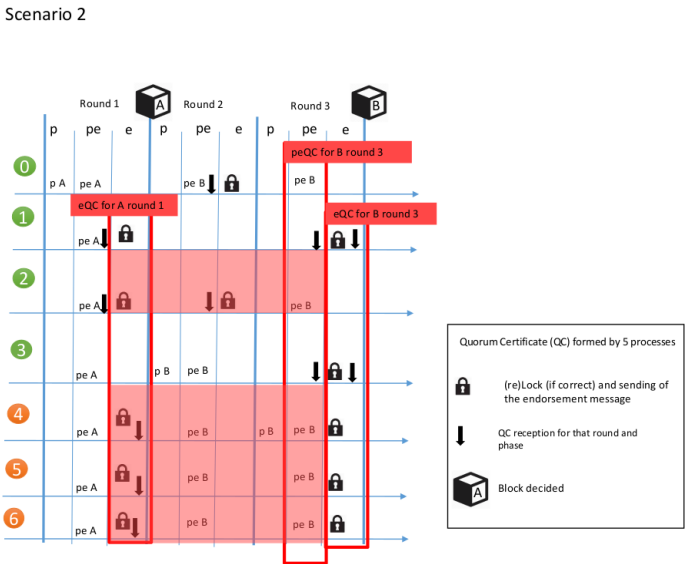

Figure 8 p_4, p_5 and p_6 performs a faulty flip-flopping, but their behaviour is indistinguishable from the p_2 given the subset of messages observed.

any endorsement message carries all justifications issued and collected following the same logic applied for the pre-endorsement messages justifications. Since those justifications chain are carried by endorsement messages then we collect them in decided blocks, that is, from processes which endorsed the block itself, p_1, p_3, p_4, p_5 and p_6 in this particular Scenario. Hence in this case, given the blocks A and B we collect information from correct processes p_1 and p_3 (Byzantine processes can omit justifications in their endorsement message).

- block A decided at round 1: such block comes with eQC from p_1, p_2, p_4, p_5 and p_6 (first red rectangle in Figure 6), and none of them has justifications.
- block B decided at round 3: such block comes with a eQC from p_1, p_3, p_4, p_5 and p_6 (fourth red rectangle in Figure 6) and with the following justifications:
  - p_1 and p_3 justifies directly its endorsement with $peQC$ for value B at rounds 3 (third red rectangle in Figure 9). Indeed, p_1 and p_3 witnessed eQC for block B in round 3.
  - p_1 and p_3 carries the justification from p_2, that is the pre-endorsement quorum certificate, $peQC$, for value B at round 2 (second red rectangle in Figure 9). Indeed, p_2 witnessed the endorsement QC for block B in round 2 and is part of the pre-endorsement quorum certificate for tound 3.

In this way, we have enough information to distinguish and account for the fork $T + 1$ Byzantine processes in the same way we did in the Scenario 1 context.

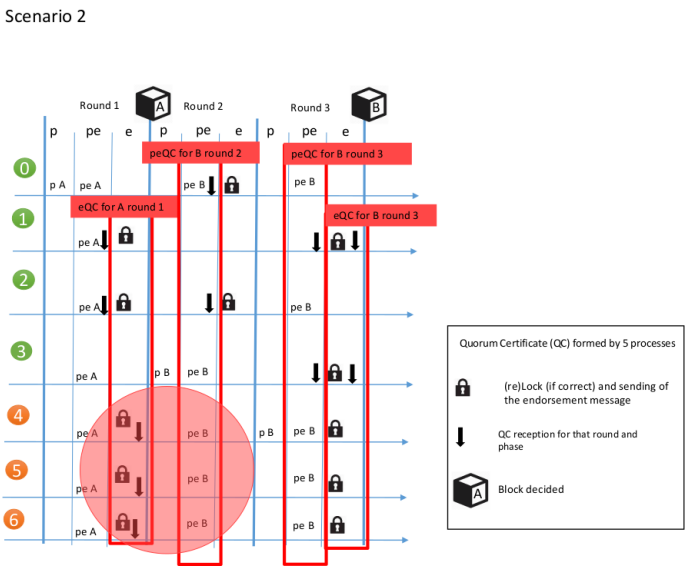

Figure 9 Justification gathered in decided block depicted using red rectangles along with highlight of detected byzantine process misbehaviour in light red.