ZLB: A Blockchain to Tolerate Colluding Majorities

Alejandro Ranchal-Pedrosa
University of Sydney and Protocol Labs
Sydney, Australia
alejandro.ranchalpedrosa@sydney.edu.au

Vincent Gramoli
University of Sydney and Redbelly Network
Sydney, Australia
vincent.gramoli@sydney.edu.au

***Abstract*—In general, consensus cannot be solved if an adversary controls a third of the system. Yet, blockchain participants typically reach consensus "eventually" despite an adversary controlling a minority of the system. Exceeding this $\frac{1}{3}$ cap is made possible by tolerating transient disagreements, where distinct participants select distinct blocks for the same index, before eventually agreeing on the same block. Until now, no blockchain could tolerate an attacker controlling a majority of the system.**

In this paper, we present Zero-Loss Blockchain (*ZLB*), the first blockchain that tolerates an adversary controlling more than half of the system. ZLB is an open blockchain that combines recent theoretical advances in accountable Byzantine agreement to exclude undeniably faulty replicas. Interestingly, ZLB does not need a known bound on the delay of messages but progressively reduces the portion of alive but corrupt replicas below $\frac{1}{3}$, and reaches consensus. Geo-distributed experiments show that ZLB outperforms HotStuff and is almost as fast as the scalable Redbelly Blockchain that cannot tolerate $n/3$ faults.

***Index Terms*—Byzantine, State Machine Replication**

I. Introduction

Blockchain systems [57] promise to track ownership of assets without a central authority and thus rely heavily on distributed nodes agreeing on a unique block at the next index of the chain. An attacker can exploit a disagreement to *double spend* by simply inserting conflicting transactions in competing blocks.

Some solutions [10], [37], [3], [25] to this problem avoid forks by guaranteeing that no disagreement can ever occur, even transiently. Such solutions typically adopt an open permissioned model where permissionless clients can issue transactions that n permissioned servers (or *replicas*) encapsulate in blocks they agree upon. These solutions typically assume partial synchrony [29] or that there exists an unknown bound on the time it takes to deliver any message. Unfortunately, it is well-known [62] that consensus cannot be solved as soon as $\frac{1}{3}$ of these replicas experience a Byzantine fault. Specifically, in these blockchains an attacker can exploit a disagreement to double spend if it controls $\frac{1}{3}$ of these replicas.

Other solutions, popularized by classic blockchains [57], [74], [34], [15], assume that the adversary controls only a minority of the replicas, typically expressed as computational power or stake. The tolerance to an adversary controlling more than $\frac{1}{3}$ but less than $\frac{1}{2}$ of the replicas is made possible by accepting forks and tolerating transient disagreements that eventually get resolved in an "eventual" consensus. Unfortunately, as soon as the adversary controls a majority of the system, then safety gets violated: This was recently illustrated by the losses of $\$70,000$ and $\$18$ million in Bitcoin Gold [64], [43] and $\$5.6$ million in Ethereum Classic [78].

In this paper, we propose the *Zero-Loss Blockchain* (or *ZLB* for short), the first blockchain that tolerates more than a majority of failed replicas while assuming partial synchrony. The problem ZLB solves is not simple for two reasons.

First, if a majority of the permissionned replicas stop participating, we cannot guarantee availability as implied by the CAP theorem [36]. On the bright side, however, blockchain systems incentivize replicas to be eventually actively participating: Typical replicas are either carefully monitored to generate financial rewards to their owner [57], [74] or incentivized to send misinformation to deceive honest replicas and steal assets [46], [33]. This departs significantly from the failure models found in closed distributed systems (e.g., datacenters, cloud services or distributed databases) where it is acknowledged [21], [51], [42] that most faults are omissions and rare commissions are due to unlucky events (e.g., disk errors [21]). So to cope with a possible liveness issue, we make the reasonable assumption that less than $n/3$ of the blockchain participants commit benign faults (omitting to send messages or sending unintelligible messages) forever. To this end, we adopt a slowly-adaptive version of the *alive-but-corrupt (a-b-c)* failure model [54]: at the beginning of each epoch there are f failing replicas among which t byzantine replicas can attack safety and liveness but where $f - t$ a-b-c replicas attack only safety.

Second, if a majority of faulty replicas collude then they can ensure that honest replicas disagree on the next block. To cope with this safety issue we rely on recent theoretical advances in the field of accountability [66], [18], [19] that guarantee that faulty replicas leave a cryptographically signed trace at an honest replica when trying to influence the decision of this replica. Honest replicas can then combine these traces to build undeniable proofs of fraud. In ZLB, honest replicas exploit these proofs of fraud to replace detected faulty replicas by new replicas until consensus is reached. Once honest replicas gather enough proofs of fraud, they start a membership change, consisting of an exclusion protocol followed by an inclusion protocol, during which replicas agree on the replicas to exclude and to include. The convergence of this membership change is guaranteed by honest replicas updating their committee at runtime by removing all replicas for which they receive a proof of fraud. Note that if no such proofs of fraud can be generated, then it implies that no disagreement can happen.

To demonstrate the efficiency of ZLB we implement it

with Bitcoin transactions and the formally verified blockchain consensus protocol, DBFT [6], and compare its performance to modern blockchain systems. We show that, on 90 machines spread across distinct continents, ZLB outperforms by 5.6 times the HotStuff [76] state machine replication that inspired Facebook Libra's [3], and obtains comparable performance to the recent Redbelly Blockchain [25], [69] that also builds upon DBFT. Our empirical results also show an interesting phenomenon in that the impact of the attacks decreases rapidly as the system size increases, due to the increased message delays.

We also develop a Zero-Loss Payment application on top of ZLB. As opposed to classic blockchain payment systems [57], [74], [53], [32] that recover from forks a posteriori, our payment application guarantees deterministic agreement—no forks—when the number f of failures is lower than $n/3$. As opposed to more recent Byzantine fault tolerant payment solutions [35], [10], [37], [3], [25], our payment application recovers eventually from a state with a majority of faults. More generally, our system solves consensus tolerating $t < n/3$ Byzantine faults, or it instead resolves disagreements to converge to a state where consensus can be solved again for $f < 5n/9$ total faults, of which $t < n/3$ are Byzantine faults and the remaining $f - t$ are a-b-c faults.

We present the background (§II), our LLB problem (§III), our ZLB solution (§IV), our evaluation (§V) and our zero-loss payment application (§VI) before presenting the related work (§VII) and concluding (§VIII).

II. Background and Preliminaries

A blockchain system [57] is a distributed system maintaining a sequence of blocks that contains *valid* (cryptographically signed) and non-conflicting transactions indicating how assets are exchanged between *accounts*.[1]

A. Byzantine state machine replication

A Byzantine State Machine Replication (SMR) [14], [45] is a replicated service that accepts deterministic commands from clients and totally orders these commands using a consensus protocol so that, upon execution of these commands, every honest replica ends up with the same state despite *Byzantine* or faulty replicas. The instances of the consensus execute in sequence, one after the other, starting from index 0. We refer to the consensus instance at index i as Γ_i.

Traditionally, given that honest replicas propose a value, the Byzantine consensus problem [62] is for every honest replica to eventually decide a value (consensus termination), for no two honest replicas to decide different values (agreement) and for the decided value to be one of the proposed values (validity). In this paper, we consider however a variant of Byzantine consensus (Def. 1) useful for blockchains [55], [28], [25] where the validity requires the decided value to be a subset of the union of the proposed values, hence allowing us to commit more proposed blocks per consensus instance.

Definition 1 (Set Byzantine Consensus)**.** *Assuming that each honest replica proposes a set of transactions, the* Set Byzantine Consensus *(SBC) problem is for each of them to decide on a set in such a way that the following properties are satisfied:*

- *SBC-Termination: every honest replica eventually decides a set of transactions;*
- *SBC-Agreement: no two honest replicas decide on different sets of transactions;*
- *SBC-Validity: a decided set of transactions is a non-conflicting set of valid transactions taken from the union of the proposed sets;*
- *SBC-Nontriviality: if all replicas are honest and propose a common valid non-conflicting set of transactions, then this set is the decided set.*

SBC-Termination and SBC-Agreement are common properties to many Byzantine consensus definition variants, while SBC-Validity states that transactions proposed by Byzantine proposers could be decided as long as they are valid and *non-conflicting* (i.e., they do not withdraw more assets from one account than its balance); and SBC-Nontriviality is necessary to prevent trivial algorithms that decide a pre-determined value from solving the problem. As a result, we consider that a consensus instance Γ_i outputs a set of enumerable decisions $out(\Gamma_i) = d_i,\ |d_i| \in \mathbb{N}$ that all n replicas replicate. We refer to the state of the SMR at the i-th consensus instance Γ_i as all decisions of all instances up to the i-th consensus instance.

B. Accountability

The replicas of a blockchain system are, by default, not accountable in that their faults often go undetected. For example, when a replica creates a fork, it manages to double spend after one of the blockchain branches where it spent coins vanishes. This naturally prevents other replicas from detecting frauds and from holding this replica accountable for its misbehavior. Recently, Polygraph [17], [18] introduced accountable consensus (Def. 2) as the problem of solving consensus if $f < n/3$ and eventually detecting $f_d \geq n/3$ faulty replicas in the case of a disagreement.

Definition 2 (Accountable Consensus)**.** *The problem of* accountable consensus *is: (i) to solve consensus if the number of Byzantine faults is $f < n/3$, and (ii) for every honest replica to eventually output at least $f_d \geq n/3$ faulty replicas if two honest replicas output distinct decisions.*

C. Solving the Set Byzantine Consensus (SBC)

A classic reduction [4], [5], [23], [18] of the problem of multi-valued consensus, which accepts any ordered set of input values, to the problem of binary consensus, that accepts binary input values, proved promising to solve the SBC problem (Def. 1) when $f < n/3$ [25]. The idea consists of executing an all-to-all reliable broadcast [8] to exchange n proposals: any delivered proposal is stored in an array *proposals* at the

[1] Note that in §IV we will implement Bitcoin's transactions where "valid" implies "non-conflicting" as requested transactions cannot be valid if their UTXOs are already consumed.

index corresponding to the identifier of the broadcaster. A binary consensus at index k is started with input value 1 for each index k where a proposal has been recorded. Once $n - f$ proposals are delivered locally, a binary consensus at the remaining indices $0 \leq \ell < n$ where $\ell \neq k$ is started with input value 0. The results of these concurrent binary consensus instances is stored into a $bitmask$ array. Hence, applying the $bitmask$ to the $proposal$ array yields a sequence of proposals whose content is the output of consensus. Polygraph [16], [17], [18] is the accountable variant of this algorithm where replicas broadcast *certificates*, sets of $2n/3$ messages signed by distinct replicas, each time they reliably broadcast or decide a binary value.

III. The Longlasting Blockchain Problem

We consider the classic distributed system model [14], [45] where messages are delivered within bounded but unknown time (i.e., partial synchrony [29]). Solving the longlasting blockchain problem is to solve consensus when possible ($f < n/3$), and to recover from a situation where consensus is violated ($n/3 \leq f < 2n/3$) by excluding faulty replicas, resolving this violation, and preventing future ones ($f' < n'/3$).

A. Longlasting Blockchain

A Longlasting Blockchain (LLB) is a Byzantine fault tolerant SMR that allows for its k^{th} *consensus instance*, denoted Γ_k, to reach a disagreement before fixing the disagreement by merging the branches of the resulting fork and deciding the union of all the past decisions using SBC (Def. 1). In particular, the j^{th} *consensus attempt* of consensus instance Γ_k to reach agreement is denoted Γ_k^j. More formally, an SMR is an LLB if it ensures termination, agreement and convergence:

Definition 3 (Longlasting Blockchain Problem)**.** *An SMR is an LLB if all the following properties are satisfied:*

1) **Termination:** *For all $j, k > 0$, the consensus attempt Γ_k^j terminates, either with agreement or disagreement.*

2) **Agreement:** *For all $k > 0$, if $f < n/3$ when Γ_k starts, then honest replicas executing consensus attempt Γ_k^1 reach agreement.*

3) **Convergence:** *There is a finite number of disagreements after which every consensus attempt of every instance solves consensus.*

Termination does not imply agreement among honest replicas whereas agreement is the classic property of consensus. Convergence guarantees that there is a limited number of disagreements (0 if $f < n/3$) before reaching agreement.

B. Threat model

In this section, we show how our adaptive adversary assigns Byzantine, alive-but-corrupt and honest roles dynamically.

a) Failures: We adopt the alive-but-corrupt (a-b-c) fault model [54], that is a refinement of the Byzantine failure model [46]. In particular, each replica is either Byzantine, a-b-c or honest. An *a-b-c replica* can take any action that violates the agreement property of the consensus protocol but if it cannot violate agreement then it acts correctly. As the a-b-c replica acts correctly if it cannot violate agreement, it never hampers termination. A *Byzantine replica* behaves abitrarily [46]. As opposed to an a-b-c replica, a Byzantine replica can hamper liveness as it can, for example, omit sending messages. Finally, any replica that is neither a-b-c nor Byzantine is called an *honest replica*. Note that each replica is assigned a specific (a-b-c, Byzantine or honest) role by an adversary as described below hence even if a Byzantine replica commits no other fault but agreement violations, it remains a Byzantine replica until the adversary assigns it a new role.

b) Slowly-adaptive adversary: As other blockchains that cope with bribery attacks [53], [44], [77], we consider a *slowly-adaptive* adversary in that the adversary assigns one of the three (a-b-c, Byzantine or honest) roles to each replica that remains unchanged for a time period we refer to as a *static period* of the adversary. As opposed to the static alive-but-corrupt fault model [54] our adversary is adaptive in that it can change roles at the beginning of each static period. Each static period p is assigned a consensus instance Γ_k and p starts when Γ_k starts. To cope with pipelined consensus instances, p may not end exactly when Γ_k ends, as there can be Γ_ℓ, ..., Γ_m instances running at this time, in which case p ends (and static period $p+1$ starts) as soon as Γ_ℓ, ..., Γ_m and Γ_k have all ended. Let n be the initial number of replicas in our system when period p starts, we assume that there are at most $f < 5n/9$ faulty replicas during p, among which $t < n/3$ are Byzantine replicas and $f - t$ replicas are a-b-c. Finally, we refer to $\delta = f/n$ as the *fault ratio*.

c) $f < 5n/9$ bound: The bound $f < 5n/9$ on the number of total replicas stems from the fact that honest replicas will identify $f_d \geq n/3$ provably faulty replicas through accountability, to then run a special membership change that executes consensus to have honest replicas agree on the replicas to exclude and the new replicas to include. Honest replicas start the membership change by locally excluding the f_d replicas that they know are faulty, even if they do not yet agree on the same set of excluded replicas. As we will explain (§IV-A1③) that all honest replicas exclude progressively all faulty replicas of the same static period of the adversary, the number of faulty replicas not excluded for the membership change f' may not exceed the bound on the number of faulty replicas $n'/3$. As $n' \leq n - f_d$ and $f' = f - f_d$, we have that $f' < n'/3$. Since $n'/3 \leq (n - f_d)/3$ and $f_d \geq n/3$, this means that $n'/3 \leq n/3 - n/9$, thus $f' < n'/3$ is equivalent to $f' < 2n/9$. Given that $f - f_d < 2n/9$ to ensure consensus of the exclusion phase of the membership change (i.e. $f' < n'/3$), and this amounts for $f' < 2n/9$, and since $f' = f - f_d$ for at least $f_d \geq n/3$, then the total number of tolerated faulty replicas must be $f = f' + f_d = 5n/9$. We refer to previous work [63] for a more detailed explanation and proofs of both bounds. The bound $t < n/3$ on the number of Byzantine replicas is immediate since without that bound it is impossible to solve consensus in the first place.

d) Relation with rational behaviors: A-b-c faults define a stronger adversary than rational players, in that a-b-c faults

are always trying to cause a disagreement, whereas rational players only deviate if deviating benefits them. Recent solutions for consensus in the presence of rational players [40] do not adhere to the a-b-c model, whereas our solution holds its properties when replacing a-b-c faults with rational players. We also denote the *fault ratio* f/n as δ. A replica that is not faulty is *honest*. In order to cope with the dynamism of our network, we need to be more precise about the number of participants. Let n be the initial number of replicas in our system, we assume a number of faulty replicas that satisfies either $t < n/3$ Byzantine replicas for consensus, or $f < 5n/9$ total faults of which at most $t < n/3$ are Byzantine for eventual consensus, where f is the total number of faults.

e) Pool of replica candidates: To model all nodes that can join as replicas in the system, we assume that there exists a large pool of m nodes among which at least $2n/3$ are honest nodes (m can be much greater than n) and the rest are a-b-c. This pool simply indicates that among the entire world of replicas that will ever be proposed to be included, at least $2n/3$ honest ones will eventually be proposed by honest replicas. Notice this is a significantly weaker assumption than assuming, for example, that honest replicas always propose other honest replicas to be included. For simplicity and w.l.o.g., we assume that no replica from this pool is proposed twice if it has been included before, within the same static period of the adversary.

Finally, we assume a public-key infrastructure (PKI) that associates replicas' identities with their public-keys, and that is common to all replicas. We also assume a computationally bounded adversary as can be found in [13], [12]. Note that in order to offer a zero-loss payment application in Section VI we will add a series of additional assumptions including an upper-bound on the amount transferred per block.

IV. The Zero-Loss Blockchain

In this section we detail our system. Its two main ideas are (i) to replace a-b-c replicas undeniably responsible for a fork by new replicas to converge towards a state where consensus can be reached, and (ii) to fund conflicting transactions (without rolling back any of them). We will show that ZLB solves the Longlasting Blockchain problem. As depicted in Figure 1, we present below the components of our ZLB system, namely the ASMR (§IV-A) and the Blockchain Manager (BM) (§IV-B) but we defer the zero-loss payment application (§VI).

As long as new requests are submitted by a client to a replica, the payment system component of the replica converts them into payments that are passed to the BM component. As depicted in Fig. 1, when sufficiently many payment requests have been received, the BM issues a batch of requests to the ASMR that, in turn, proposes it to the consensus component, which exchanges messages through the network for honest replicas to agree. If a disagreement is detected, then the accounts of the a-b-c replicas are slashed. Consider that Alice (A) attempts to double spend by (i) spending her \$1M with both Bob (B) and Carol (C) in tx and tx', respectively, and that (ii) p_k is faulty and commits a-b-c faults to produce a disagreement. Once the ASMR detects the disagreement, BM

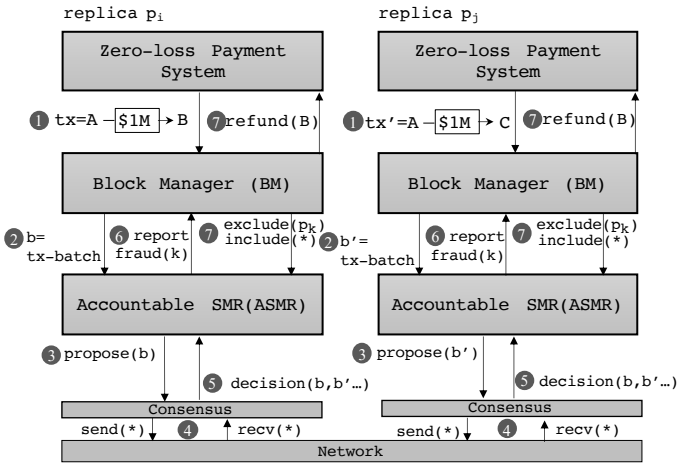

Fig. 1: The distributed architecture of our ZLB system relies on Accountable SMR (ASMR), BM and the payment system. Consider that Alice has \$1M initially and attempts to double spend by modifying the code of a replica p_k and issuing ❶ conflicting transfers tx and tx' of \$1M from Alice's account (A) to Bob's (B) and Carol's (C). Upon reception the replicas batch this transaction in distinct blocks b and b' ❷ and let us assume that p_k convinces some honest replicas to decide b whereas others decide b'. ❸–❺ The ASMR component detects the a-b-c replica p_k that tried to double spend, the associated transactions tx and tx' and account A with insufficient funds. It uses A's balance to fund transaction tx, ❻ notifies BM that ❼ excludes or replaces replica p_k and ❼ funds tx' with p_k's slashed deposit.

is notified, replica p_k is excluded or replaced and tx' is funded with p_k's slashed deposit.

A. Accountable SMR (ASMR)

In order to detect faulty replicas, we now present, as far as we know, the first accountable state machine replication, called ASMR. ASMR consists of running an infinite sequence of five actions: ① the accountable consensus (Def. 2) that tries to decide upon a new set of transactions, ② a confirmation that aims at confirming that the agreement was reached, ③–④ a membership change that aims at replacing a-b-c replicas responsible for a disagreement by new replicas and ⑤ a reconciliation phase that combines all the decisions of the disagreement, as depicted in Figure 2.

1) The phases of ASMR: For each index, ASMR first executes the accountable consensus (§II) phase ① for a membership (typically stored in the blockchain [72]) to try to agree on a set of transactions then it runs four subsequent phases ②–⑤ to recover from a possible disagreement.

① **ASMR consensus:** Honest replicas propose a set of transactions, which they received from clients, to the accountable consensus (Def. 2) in the hope to reach agreement. Note that as our solution builds upon DBFT, we implemented its accountable variant called Polygraph [16], [17], [18] that we also evaluated in Section V. When the consensus terminates,

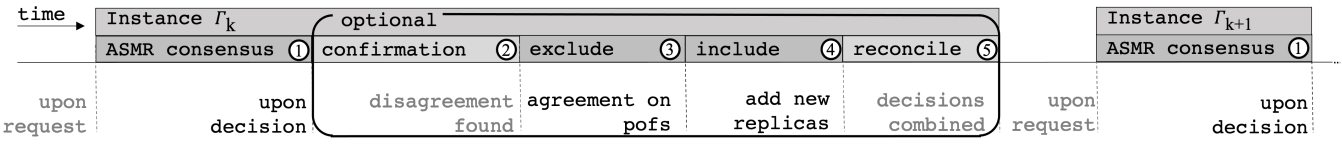

Fig. 2: If there are enqueued requests that wait to be served, then a replica starts a new instance Γ_k by participating in an ASMR consensus phase ①; a series of phases may follow: ② the replica tries to confirm this decision to make sure no other honest replica disagrees, ③ it invokes an exclusion protocol if faulty replicas caused a disagreement, ④ it then includes new replicas to compensate for the exclusion, and ⑤ merges the two batches of decided transactions. Some of these phases complete upon consensus termination (in black) whereas other phases terminate upon simple notification reception (in grey). The replica starts a new instance Γ_{k+1} without waiting for phases ②-⑤ to terminate, as this is not always guaranteed.

all honest replicas agree on the same decision or some honest replicas disagree: they decide distinct sets of transactions.

② **Confirmation:** As honest replicas could be unaware of the other decisions, they enter a confirmation phase waiting for messages coming from more distinct replicas than what consensus requires. If faulty replicas caused a disagreement, then the confirmation terminates and leads honest replicas to detect disagreements, i.e., honest replicas receive certificates supporting distinct decisions. Otherwise, this phase may not terminate, as an honest replica needs to deliver messages from more than $(\delta + 1/3) \cdot n$ replicas, where δ is the ratio of a-b-c replicas $\delta = f/n$. In particular, and due to the number of 'conflicting histories' [67], in order to guarantee that no disagreement was possible in the presence of $f < 5n/9$ faulty replicas, honest replicas need to receive agreeing messages from $n - x$ replicas solving $\lfloor (n-x)/(f-x+1) \rfloor = 1$, which translates into at least $8n/9$ replicas to guarantee that no disagreement was possible by a fault ratio δ. However, Γ_k always terminates, as it proceeds in parallel with the confirmation without waiting for its termination. If the confirmation phase terminates, it either confirms that a block is irrevocably final (no replica disagreed), or a membership change starts.

③-④ **Membership change:** Our membership change (Alg. 1) consists of two consecutive consensus algorithms: one that excludes faulty replicas (line 22), and another that adds newly joined replicas (line 42). We separate inclusion and exclusion in two consensus instances to avoid deciding to exclude and include replicas proposed by the same replica. Replica p_i maintains a series of variables: the current consensus instance Γ_k, the a-b-c replicas among the whole set C of current replica ids, a set C' of replica ids that is updated at runtime for the exclusion protocol, the pool of replicas $pool$, a set of certificates $certificates$, a set of proofs of fraud (PoFs) $pofs$ and of new PoFs new_pofs, a local threshold f_d of detected a-b-c replicas, a set $cons\text{-}exclude$ of decided PoFs and a set $cons\text{-}include$ of decided new replicas.

③ **Exclusion protocol:** Honest replicas identify faulty replicas by cross-checking received certificates. These cross-checks produce undeniable PoFs with conflicting signed messages from the same replica, indicating equivocation. If replicas detect $f_d = \lceil n/3 \rceil$ faulty replicas (via distinct PoFs), they stop their pending ASMR consensus (line 19) before restarting it with the new set of replicas (line 49). Then, honest replicas start the membership change ignoring messages from these f_d replicas by using instead an updated committee C' that excludes these replicas (lines 20-22). Honest replicas propose in line 22 their set of PoFs at the start of the exclusion protocol ex-propose by invoking the Polygraph accountable consensus algorithm we mentioned in §II-C.

The key novelty of our exclusion protocol is for replicas to exclude other replicas, and thus update their committee C', at runtime upon reception of new valid PoFs (lines 23-25). Hence, upon delivering a certificate (line 31), honest replicas verify that the certificate contains a threshold $\lceil 2|C'|/3 \rceil$ of signatures from non-excluded replicas (line 35) and decide the proposals that the certificate justifies at line 36. Upon updating their committee, honest replicas re-check all their certificates (line 27) and re-broadcast their PoFs (line 26). As the cardinality of C' decreases at runtime, the threshold $\lceil 2|C'|/3 \rceil$ is guaranteed to be eventually met. As our exclusion protocol solves the SBC problem (cf. Lemma IV.1), it maximizes the number of excluded replicas by deciding at least $\lceil 2|C'|/3 \rceil$ proposals at once. Note that instead of waiting for f_d PoFs (line 17), replicas could start Alg. 1 as soon as they detect one fault. However, waiting for at least f_d PoFs guarantees that a membership change is necessary and will help remove many faulty replicas from the same coalition at once. Moreover, waiting for f_d PoFs allows us to guarantee agreement of the exclusion protocol.

④ **Inclusion protocol:** To compensate for the excluded replicas, an inclusion protocol inc-propose (line 42) adds new candidate replicas taken from the pool of candidates (§III-B) in line 41. This inclusion protocol is also an instance of Polygraph, like the exclusion protocol, except that it differs in the format and verification of the proposals: each proposal contains as many new replicas as the number of replicas excluded (lines 41-42). By contrast with the exclusion protocol, the inclusion protocol does not update its committee at runtime, but uses the updated committee (C from line 40 onward), where honest replicas already excluded at least f_d replicas from C (line 40). Since the union of the $\lceil 2|C|/3 \rceil$ proposals contains more than enough replicas to include, we apply a deterministic function choose (line 44) to the union of all decided proposals. This function restores the committee size to n by selecting the replicas evenly from all decided proposals.

Algorithm 1 Membership change at replica p_i, consensus Γ_k

```
1: State:
2:   Γ_k, k^th instance of ASMR consensus p_i participates to.
3:   C, set of replicas forming the committee
4:   C', updated set of replicas, initially C' = C
5:   certificates, received certificates during exclusion, initially ∅
6:   pofs, the set of proofs of fraud (PoFs), initially ∅
7:   new_pofs, set of newly delivered PoFs, initially ∅
8:   cons-exclude, the set of PoFs output by consensus, initially ∅
9:   cons-include, the set of new replicas output by consensus, initially ∅
10:  pool, the pool of replicas from which to propose new replicas
11:  a-b-c ∈ I, the identity of an agreed a-b-c replica, initially ∅
12:  f_d, the threshold of proofs of fraud to recover, ⌈n/3⌉ by default
---
13: Upon receiving a list of proofs of fraud _pofs:
14:   if (verify(_pofs)) then                          ▷ if PoFs are correctly signed
15:     new_pofs ← _pofs\pofs
16:     pofs.add(_pofs)                                ▷ add PoFs on distinct replicas
17:     if (ex-propose not started) then
18:       if (size(pofs) ≥ f_d) then                   ▷ enough to change members
19:         if (Γ_k started and not finished) then Γ_k.stop()
20:         C' ← C'\new_pofs.replicas()
21:         ex-propose.update_committee(C')            ▷ update committee
22:         ex-propose.start(pofs)                     ▷ exclusion consensus
23:     else if (new_pofs ≠ ∅ and ex-propose not finished) then
24:       C' ← C'\new_pofs.replicas()
25:       ex-propose.update_committee(C')              ▷ update committee
26:       broadcast(new_pofs)                          ▷ broadcast new PoFs
27:       ex-propose.check_certificates(certificates)  ▷ recheck certificates
---
28: Upon receiving a certificate ex-cert of the exclusion protocol:
29:   if (ex-cert ∉ certificates and verify_certificate(ex-cert)) then
30:     certificates.add(ex-cert)
31:   ex-propose.check_certificates({ex-cert})         ▷ check certificate with current C'
---
32: function ex-propose.check_certificates(certs):
33:   for all cert ∈ certs do
34:     if (verify_certificate(cert)) then
35:       if (|cert.replicas() ∩ C'| ≥ 2|C'|/3) then   ▷ current threshold
36:         ex-propose.cert_decide(cert)               ▷ decide certificate's decision
---
37: Upon deciding a list of proofs of fraud cons-exclude in ex-propose:
38:   detected-fraud(cons_exclude.get_replicas())      ▷ application punishment
39:   pofs ← pofs \ cons-exclude.get_pofs()            ▷ discard the treated pofs
40:   C ← C \ cons-exclude.get_a-b-cs()                ▷ exclude a-b-c
41:   inc-prop ← pool.take(|cons-exclude|)             ▷ take replicas from the pool
42:   inc-propose.start(inc-prop)                      ▷ inclusion cons.
---
43: Upon deciding a list of replicas to include cons-include in inc-propose:
44:   new_replicas ← choose(|cons-exclude|,cons-include)  ▷ deterministic
45:   for all new_replica ∈ new_replicas do            ▷ for all new to inc.
46:     set-up-connection(new_replica)                 ▷ new replica joins
47:     send-catchup(new_replica)                      ▷ get latest state
48:   C ← C ∪ new_replicas
49:   if (Γ_k stopped) then goto ① of Fig. 2           ▷ restart cons.
```

This guarantees (i) a fair distribution of inclusions across all decisions, and (ii) that the fault ratio does not increase even if all included replicas are a-b-c. At the end, excluded replicas are punished by the application layer (l.38) and new replicas are included (l.44-49).

Honest replicas from different partitions might find themselves at different consensus instances at the moment they execute the membership change. For this reason, even after the membership change terminates, there is a transient period where honest replicas may receive blocks with certificates containing excluded replicas, that were decided and broadcast by other honest replicas in a different partition before they executed the membership change. Note, however, that all certificates contain at least 1 honest replica by construction (since all certificates contain at least $\lceil 2n/3 \rceil$ signatures and $f < 2n/3$), and thus all honest replicas eventually update their committee and stop generating new certificates with excluded replicas.

⑤ **Reconciliation:** Upon delivering a conflicting block with an associated valid certificate, the reconciliation starts by combining all transactions that were decided by distinct honest replicas in the disagreement. These transactions are ordered through a deterministic function, whose simple example is a lexicographical order but can be made fair by rotating over the indices of the instances. Note that, once all honest replicas have reconciled all the conflicting blocks with valid certificates, honest replicas obtain the same totally ordered sequence of transactions. Although there could be conflicting transactions in this agreed transaction sequence, this sequence guarantees that the blockchain state across all honest replicas is consistent. For example, consider two conflicting transactions tx and tx' where tx comes first before tx' in the sequence. Upon execution, if tx executed successfully, then the later execution of tx' will fail due to the conflict. This also means that the state needs to be recomputed any time a new conflicting block is being discovered. The total order is a direct consequence of the ASMR, but we will explain in Section VI how this ASMR can be extended into a zero-loss payment system by reimbursing the recipient of an unsuccessful transaction tx' with the assets deposited by the guilty replicas.

Once the current instance Γ_k terminates, another instance Γ_{k+1} can start, even if it runs concurrently with a confirmation or a reconciliation at index k or at a lower index.

B. Blockchain Manager (BM)

We now present the Blockchain Manager (BM) that builds upon ASMR to merge the blocks from multiple branches of a blockchain when forks are detected. Once a fork is identified, the conflicting blocks are not discarded as it would be the case in classic blockchains when a double spending occurs, but they are merged. Upon merging blocks, BM also copes with conflicting transactions, as the ones of a payment system, by taking the funds of excluded replicas to fund conflicting transactions. We defer to §VI the details of the amount replicas must have on a deposit to guarantee this funding.

Similarly to Bitcoin [57], BM accepts transaction requests from a permissionless set of users. In particular, this allows users to use different devices or wallets to issue distinct transactions withdrawing from the same account—a feature that is not offered in payment systems without consensus [22]. In contrast with Bitcoin, but similarly to recent blockchains [35], [69], our system does not incentivize all users to take part in trying to decide upon every block, instead a restricted set of permissioned replicas have this responsibility for a given block. This is why ZLB offers what is often called an open permissioned blockchain [25]. Nevertheless, ASMR can offer a permissionless blockchain with committee sortition [35] without substantial modifications.

1) Guaranteeing consistency across replicas: By building upon the accountability of the underlying ASMR that resolves disagreement, BM features a block merge to resolve forks by

replacing faulty replicas by new replicas. A consensus may reach a disagreement if $f \geq n/3$, resulting in the creation of multiple branches or blockchain forks. BM builds upon the membership change of ASMR in order to recover from forks. In particular, since ASMR excludes f_d faulty replicas each time a disagreement occurs, the ratio of faulty replicas $\delta = f/n$ converges to a state where consensus is guaranteed. The maximum number of branches that can result from forks depends on the number t of Byzantine faults and the number f of total faults [67], with $\lfloor (n-t)/(f-t+1) \rfloor = 3$ branches or less for $f < 5n/9$, $t < n/3$.

Algorithm 2 Block merge at replica p_i

```
1: State:
2:   Ω, a blockchain record with fields:
3:     deposit, an integer, initially 0
4:     inputs-deposit, a set of deposit inputs, initially in the first deposit
5:     punished-acts, a set of punished account addresses, initially ∅
6:     txs, a set of UTXO transaction records, initially in the genesis block
7:     utxos, a list of unspent outputs, initially in the genesis block

8: Upon receiving conflicting block block:                    ▷ merge block
9:   for tx in block do                                       ▷ go through all txs
10:    if (tx not in Ω.txs) then                             ▷ check inclusion
11:      CommitTxMerge(tx)                                   ▷ merge tx, go to line 17
12:      for out in tx.outputs do                            ▷ go through all outputs
13:        if (out.account in Ω.punished-acts) then          ▷ if punished
14:          PunishAccount(out.account)                      ▷ punish also this new output
15:  RefundInputs()                                          ▷ refill deposit, go to line 24
16:  StoreBlock(block)                                       ▷ write block in blockchain

17: CommitTxMerge(tx):
18:   toFund ← 0
19:   for input in tx.inputs do                              ▷ go through all inputs
20:     if (input not in Ω.utxos) then                       ▷ not spendable, need to use deposit
21:       Ω.inputs-deposit.add(input)                        ▷ use deposit to refund
22:       Ω.deposit ← Ω.deposit − input.value                ▷ deposit decreases in value
23:     else Ω.consumeUTXO(input)                            ▷ spendable, normal case

24: RefundInputs():
25:   for input in Ω.inputs-deposit do                       ▷ go through inputs that used deposit
26:     if (input in Ω.utxos) then                           ▷ if they are now spendable
27:       Ω.consumeUTXO(input)                               ▷ consume them
28:       Ω.deposit ← Ω.deposit + input.value                ▷ and refill deposit
```

2) Protocol to merge blocks: As depicted in Alg. 2, the state of the blockchain Ω consists of a set of inputs *inputs-deposit* (line 4), a set of account addresses *punished-acts* (line 5) that have been used by a-b-c replicas, a *deposit* (line 3), that is used by the protocol, a set *txs* of transactions and a list *utxos* of UTXOs. The algorithm propagates blocks by broadcasting on the network and starts upon reception of a valid block that conflicts with a known block of the blockhain Ω by trying to merge all transactions of the received block with those of the blockchain Ω (line 11). This is done by invoking the function CommitTxMerge (lines 17–23) where the inputs get appended to the UTXO table and conflicting inputs are refunded with the deposit (line 22) of a-b-c replicas. We explain in §VI how to build a payment system with a sufficient deposit to remedy successful disagreements.

3) Cryptographic techniques: ZLB is a blockchain that inherits the same *Unspent Transaction Output (UTXO)* model of Bitcoin [57]. To provide authentication and integrity, transactions are signed using the Elliptic Curves Digital Signature Algorithm (ECDSA) with parameters `secp256k1`, as in Bitcoin [57]. Each honest replica assigns a monotonically increasing sequence number to its transactions. The network communications use gRPC between clients and replicas and raw TCP sockets between replicas, but all communication channels are encrypted through SSL. Finally, the exclusion protocol (Alg. 1) uses ECDSA for authenticating the sender of messages responsible for disagreements (i.e., for PoFs). Unlike ECDSA, threshold encryption cannot be used to trace back the faulty users as they are encoded in less bits than what is needed to differentiate users, and message authentication codes (MACs) are insufficient to provide this transferrable authentication [20].

C. Fault tolerance and proofs

In this section, we show that ZLB solves the Longlasting Blockchain problem (Def. 3) hence recovering from a majority of failures. To show that ZLB solves LLB (Def.3) we first need to show that both the exclusion and inclusion protocols solve consensus.

Lemma IV.1. *The exclusion protocol solves SBC-consensus.*

Proof. SBC-Validity is immediate from the fact that honest replicas only decide at least $f_d \geq n/3$ valid PoFs proposed by another replica. The same occurs with SBC-Nontriviality. We consider now agreement and termination.

SBC-Agreement. By construction, honest replicas only start the membership change if they gather at least $f_d \geq n/3$ PoFs from distinct replicas, which they exclude at the start of the exclusion protocol and propose to exclude permanently during the exclusion consensus. This means that there are not enough faulty replicas left to cause a disagreement, because $(f - f_d) < (n - f_d)/3$ for $f < 5n/9$, regardless of whether honest replicas initially exclude the same or different sets of $f_d \geq n/3$ faulty replicas.

SBC-Termination. $f_d \geq n/3$ must be faulty for the membership change to even start. If all honest replicas start the exclusion consensus having excluded the same set of faulty replicas, then termination is guaranteed because $f - f_d < n'/3$ for $n' = n - f_d$ and $f < 5n/9$. We consider instead the case that honest replicas start having excluded different faulty replicas. W.l.o.g. suppose two partitions A and B of honest replicas at the start of the exclusion consensus. Since $f - f_d < n'/3 \leq 2n/9$ for $n' = n - f_d$, $f_d \geq n/3$ and $f < 5n/9$, this means that if $|A| + (f - f_d) \geq 2n'/3$ then $|B| + (f - f_d) < 2n'/3$. As a result, either none or only one of the two partitions can progress (while the partition lasts), even if the remaining faulty replicas try to cause a disagreement. Let $p_i \in A$ and $p_j \in B$ start the exclusion protocol excluding each different sets of $f_d \geq n/3$ faulty replicas. These sets differ in at most $x \in [0, 2n/9]$ faulty replicas. Suppose no partition of honest replicas progresses (because remaining faulty replicas omit). Eventually, both partitions exclude the union of the faulty replicas $f_d + x \in [n/3, 5n/9)$, agree on the excluded set and can terminate after updating the committee. Suppose instead that $p_i \in A$ terminates with certificate $cert_{p_i}$ that contains $x \in [0, 2n/9)$ replicas that have been excluded by

$p_j \in B$. Then, after p_j receives the PoFs from p_i and excludes at runtime $f_d + x$ replicas, then $|cert_{p_i}| - x \geq 2(n - f_d - x)/3$ because $|cert_{p_i}| \geq 2(n - f_d)/3$ for p_i to terminate with such certificate. Thus, the exclusion consensus solves SBC-Consensus □

Lemma IV.2. *The inclusion protocol solves SBC-consensus.*

Proof. By Lemma IV.1 all honest replicas start the inclusion consensus after agreeing on excluding $x \in [n/3, 5n/9)$ faulty replicas from the original committee C, $|C| = n$, and such that the remaining $f - x$ faulty replicas are $f - x < n'/3$ for $n' = n - x$. Since the inclusion protocol is an instance of Polygraph, $f' < n'/3$ for $f' = f - x$ and the adversary is slowly adaptive, the inclusion protocol solves consensus. □

In Theorem IV.3 we prove that ZLB solves LLB. The idea is that the pool of replicas will eventually decrease the fault ratio below $1/3$, once honest replicas start proposing to include other honest replicas (§III).

Theorem IV.3 (Convergence)**.** *ZLB solves LLB.*

Proof. First, agreement and termination follows from the correctness of the underlying consensus [23], [70]. Thus, if $f < n/3$ there is no disagreement and thanks to Polygraph there is consensus (agreement). If instead $f \geq n/3$, since $t < n/3$ and a-b-c replicas do not affect termination, there is still termination, either with agreement or with disagreement. If there is a disagreement, then there are at least f_d faulty replicas of which at most $t < n/3$ are Byzantine and the rest a-b-c, which implies that there will be termination of this iteration either with or without a disagreement (since a-b-c replicas do not prevent termination by definition), proving termination.

For convergence, Lemmas IV.1 and IV.2 show that the exclusion and inclusion satisfy consensus. Polygraph's accountability guarantees that every disagreement will lead to each honest replica identifying at least f_d faulty replicas. The inclusion consensus does not increase the fault ratio, since the inclusion consensus does not include more replicas than the number of excluded replicas by the exclusion consensus (thanks to the deterministic function) and all excluded replicas are faulty. As the inclusion consensus decides at least $2n'/3$ proposals where $n' \in [n - f_d, n - (f - t)]$ and the remaining faulty replicas are $f' \leq f - f_d < n/3 < 2n'/3$, it follows that some proposals from honest replicas will be decided, and these proposals contain replicas to include that have been taken from the pool of replica candidates. As this pool is finite and no replica is included more than once, it follows that the fault ratio will eventually decrease (in any sufficiently long static period of the adversary), after enough membership changes result in enough honest replicas added from the pool. Some inclusion consensus will thus eventually lead to a fault ratio $\delta < 1/3$ (i.e. $f < n/3$) and agreement is reached from then on. □

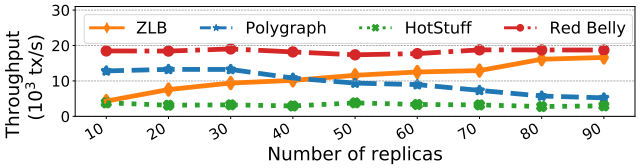

Fig. 3: Throughput of ZLB compared to that of Polygraph [18], HotStuff [76] and Redbelly Blockchain [25], [69].

V. Experimental Evaluation

This section answers the following: Does ZLB offer practical performance in a geo-distributed environment? How does ASMR perform compared to the HotStuff state machine replication that inspired Facebook Libra [3] and the recent fast Redbelly Blockchain [25]? What is the impact of large scale coalition attacks on the recovery of ASMR? We defer the evaluation of a zero-loss payment application to §VI.

a) Selecting the right blockchains for comparison: As we offer a solution for open networks, we cannot rely on the synchrony assumption made by other blockchains [35]. As we need to reach consensus, we have to assume an unknown bound on the delay of messages [29], and do not compare against randomized blockchains [55], [28], [38], [52] whose termination proof had some issues [70]. This is why we focus our evaluation on partially synchronous blockchains. We thus evaluated Facebook Libra [3], however, its performance was limited to 11 transactions per second, seemingly due to its Move VM overhead. Hence, we omit these results here and focus on its raw state machine replication (SMR) algorithm, HotStuff and its available C++ code that was previously shown to lower communication complexity of traditional Byzantine fault tolerance SMRs [76] (we use the unchanged original implementation in its default configuration [75]). We also evaluate the recent scalable Redbelly Blockchain [25], and the Polygraph protocol [18] as it is, as far as we know, the only implemented accountable consensus protocol. Nevertheless, this protocol does not tolerate more than $n/3$ failures as it cannot recover after detection.

b) Geodistributed experimental settings: We deploy the four systems in two distributed settings of c4.xlarge Amazon Web Services (AWS) instances equipped with 4 vCPU and 7.5 GiB of memory: (i) a LAN with up to 100 machines and (ii) a WAN with up to 90 machines. We evaluate ZLB with a number of failures f up to $\lceil \frac{2n}{3} \rceil - 1$, however, when not specified we fix $f = d = \lceil 5n/9 \rceil - 1$. All error bars represent the 95% confidence intervals and the plotted values are averaged over 3 to 5 runs. All transactions are ~ 400-byte Bitcoin transactions with ECDSA signatures [57].

A. ZLB vs. HotStuff, Redbelly and Polygraph

Figure 3 compares the performance of ZLB, Redbelly Blockchain (RBB) and Polygraph deployed over 5 availability zones of 2 continents, California, Oregon, Ohio, Frankfurt and Ireland (exactly like the Polygraph experiments [18]), with $f = 0$. For ZLB, we only represent the decision throughput

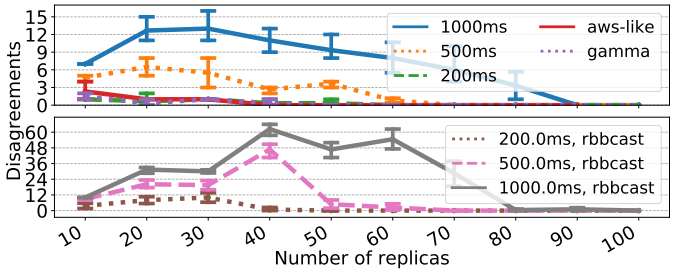

Fig. 4: Disagreeing decisions for various uniform delays and for delays generated from a Gamma distribution and a distribution that draws from observed AWS latencies, when equivocating while voting for a decision (top), and while broadcasting the proposals (bottom), for $f = \lceil 5n/9 \rceil - 1$.

that reaches $16,626$ tx/sec at $n = 90$ as the confirmation throughput is similar ($16,492$ tx/sec).

First, RBB offers the highest throughput. As expected it outperforms ZLB due to its lack of accountability: it does not require messages to piggyback certificates to detect PoFs. Both solutions solve SBC so that they decide more transactions (txs) as the number of proposals enlarges and use the same batch size of $10,000$ txs per proposal. As a result ASMR scales: the cost of tolerating $f \geq n/3$ failures even appears negligible at 90 replicas. Second, HotStuff offers the lowest throughput. Note that there are various variants of HotStuff and accountable algorithms have been proposed [66], we used HotStuff in its default configuration with its dedicated clients, they transmit the proposal to all servers to save bandwidth by having servers exchanging only a digest of each transaction. The performance is explained by the fact that HotStuff decides one proposal per consensus instance (i.e. one batch of $10,000$ txs), regardless of the number of submitted transactions, which is confirmed by previous observations [73]. By contrast, ZLB becomes faster as n increases to outperform HotStuff by $5.6\times$ at $n = 90$, thanks to the superblock optimization that allows ZLB to decide multiple proposals at once per instance of its multi-valued consensus [24]. Finally, Polygraph is faster at small scale than ZLB, because Polygraph's distributed verification and reliable broadcast implementations [18] are not accountable, performing less verifications. From 40 nodes, Polygraph is slower because of our optimizations: e.g., its RSA verifications are larger than our ECDSA signatures and consume more bandwidth.

B. Scalability of ZLB despite coalition attacks

To evaluate ZLB under failures, we implemented two possible coalition attacks: the reliable broadcast attack that causes a disagreement on delivered *proposals*, and the binary consensus attack that causes a disagreement on the decided *bitmask* (§II-C), with $f = \lceil 5n/9 \rceil - 1$ a-b-c replicas. To disrupt communications between partitions of honest replicas, we inject random communication delays between partitions based on the uniform and Gamma distributions, and the AWS delays obtained in previous measurements [56], [26], [25]. (A-b-c replicas communicate normally with each partition.)

Fig. 4(top) depicts the amount of disagreements as the number of distinct proposals decided by honest replicas, caused by the binary consensus attack. First, we select uniformly distributed delays between the two partitions of 200, 500 and 1000 milliseconds. Then, we select delays following a Gamma distribution with parameters taken from [56], [26] and a distribution that randomly samples the fixed latencies previously measured between AWS regions [25]. We automatically calculate the maximum amount of branches that the size of a-b-c faults can create (i.e., 3 branches for $f < 5n/9$), create one partition of honest replicas per branch, and apply these delays between any pair of partitions. Interestingly, we observe that our agreement property is scalable: the greater the number of replicas (maintaining the fault ratio), the harder for attackers to cause disagreements. This scalability phenomenon is due to an unavoidable increase of the latency between attackers as the scale enlarges, which gives relatively more time for the partitions of honest replicas to detect the faulty replicas, hence limiting disagreements. With more realistic network delays (Gamma distribution and AWS latencies) that are lower in expectation than the uniform delays, a-b-c replicas can barely generate a single disagreement. This confirms the scalability of our system.

Fig. 4(bottom) depicts the amount of disagreements under the reliable broadcast attack. The number of disagreements is substantially higher during this attack than during the binary consensus attack, however, it drops faster as the system enlarges, because the attackers expose themselves earlier.

C. Disagreements due to failures and delays

We now evaluate the impact of even larger coalitions and delays on ZLB, we measure the number of disagreements as we increase the fault ratios and the partition delays in a system from 20 to 100 replicas. These delays could be theoretically achieved with man-in-the-middle attacks, but are notoriously difficult on real blockchains due to direct peering between the autonomous systems of mining pools [30].

While ZLB is quite resilient to attacks for realistic but not catastrophic delays (Fig. 4), attackers can try to attack when the network collapses for a few seconds between regions. Our experiments show that attackers can reach up to 52 disagreeing proposals for a uniform delay of 10 seconds between partitions of honest replicas for the binary consensus attack, and up to 33 disagreements for a uniform delay of 5 seconds, with $n = 100$. Further tests showed that the reliable broadcast attack reaches up to 165 disagreeing proposals with a 5-second uniform delay.

D. Time to merge blocks and change members

To have a deeper understanding of the cause of ZLB delays, we measured the time needed to merge blocks and to change members by replacing a-b-c replicas by new ones. We show here the times to locally merge two blocks for different sizes assuming the worst case: all transactions conflict. This is the time taken in the worst case because replicas can merge proposals that they receive concurrently (i.e., without halting consensus). Our experiments show that the times to merge two

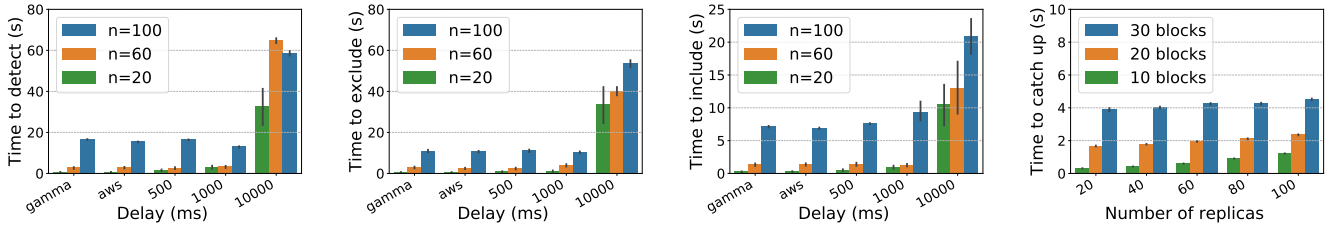

Fig. 5: (Left to right) Time to detect $\lceil \frac{n}{3} \rceil$ a-b-c replicas, exclude them, include new replicas, per delay distribution and number of replicas; and catch up per number of blocks and replicas, with $f = \lceil 5n/9 \rceil - 1$.

blocks of 100, 1000, and 10000 transactions are 0.55, 4.20 and 41.38 milliseconds, respectively. It is clear that this time to merge blocks locally is negligible compared to the time it takes to run the consensus algorithm.

Figure 5 shows the time to detect f_d a-b-c (left), and to run the exclusion (center-left) and inclusion (center-right) consensus, for a variety of delays and numbers of replicas. The time to detect reflects the time from the start of the attack until honest replicas detect the attack: If the first f_d a-b-c replicas are forming a coalition together and cause a disagreement, then the times to detect the first a-b-c and the first f_d a-b-c replicas overlap. (We detect all at the same time.) The time to exclude (57 seconds) is significantly larger than to include (21 seconds) for large communication delays, due to the proposals of the exclusion consensus carrying PoFs and leading replicas to execute a time consuming cryptographic verification. With shorter communication delays, performance becomes practical. Finally, Figure 5 (right) depicts the time to catch up depending on the number of proposals (i.e., blocks). This time increases linearly with the number of replicas, due to the catchup requiring to verify larger certificates, but it remains practical at $n = 100$ nodes.

VI. A Zero-Loss Payment Application

In this section, we describe how ZLB can be used to implement a *zero-loss payment system* where no honest replica loses any coin. The key idea is to request the consensus replicas to deposit a sufficient amount of coins in order to spend, in case of an attack, the coins of faulty replicas to avoid any honest replica loss.

a) Assumptions: In order to measure the expected impact of a coalition attack succeeding with probability ρ in forking ZLB by leading a consensus to disagreement, we first need to make the following assumptions:

1) **Fungible assets.** We assume that users can transfer assets (like coins) that are *fungible* in that one unit is interchangeable and indistinguishable from another of the same value. An example of a fungible asset is a cryptocurrency.

2) **Deposit refund.** To limit the impact of one successful double spending on a block, ZLB keeps the deposit for a number of blocks m, before returning it. A transaction should not be considered *final* (i.e. irreversible) until it reaches this blockdepth m. We call thus m the *finalization blockdepth*. Attackers can fork into a branches, and try to spend multiple times an amount $\mathfrak{G}$ (per block), which we refer to as the *gain*, obtaining a maximum gain of $(a-1)\mathfrak{G}$. Each correct replica can calculate the gain by summing up all the outputs of all transactions in their decided block. Additionally, replicas can limit the gain to an upper-bound by design, discarding blocks whose sum of outputs exceeds the bound, or they can allow the gain to be as much as the entire circulating supply of assets. Note that this assumption differs from the traditional blockchain model that cannot offer a zero-loss payment application: instead of upper-bounding the amount of instruction steps transactions of the same block can take, we limit the amount of assets transactions of the same block can transfer. In general, other mechanisms can provide zero-loss even for transactions whose spent funds greatly exceed the stake by, for example, delaying further the finalization (and the locking period of the stake) w.r.t. the transaction amount. We leave for future work the analysis of zero-loss result for variations of the model. The *deposit* $\mathfrak{D}$ is a factor of the gain, i.e., $\mathfrak{D} = b \cdot \mathfrak{G}$. The goal is for every coalition to have at least $\mathfrak{D}$ deposited, and since every coalition has at least size $\lceil n/3 \rceil$, this means that each replica must deposit an amount $3b\mathfrak{G}/n$.

3) **Network control restriction.** Once faulty replicas select the disjoint subsets (i.e., the partitions) of honest replicas to suffer the disagreement, we prevent Byzantine replicas from communicating infinitely faster than honest replicas in different partitions. More formally, let X_1 (resp. X_2) be the random variable of the time it takes for a message between two replicas within the same partition (resp. two honest replicas from different partitions). We have $E(X_1)/E(X_2) > \varepsilon$, for some $\varepsilon > 0$. Note that the definition of X_1 also implies that it is the random variable of the communication time of either two honest replicas of the same partition or any Byzantine replica with any other replica. This probabilistic synchrony assumption is similar to that of other blockchains (e.g. Bitcoin) that guarantee exponentially fast convergence, a result that also holds for ZLB under the same assumptions. In the following, we show an analysis focusing on the attack on each consensus iteration, considering a successful disagreement if there is a fork in a single consensus instance, even for a short period of time.

This zero-loss payment system can also work with non-fungible tokens (NFTs) and smart contracts, with the exception that one of the two recipients of the same NFT (or of disagreeing states) will see their NFT taken back (or their returned state reverted) in exchange for a previously agreed-upon reimbursement for the inconvenience.

b) Zero-loss: The goal of the deposited funds is to cope with the duplication of coins resulting from attempted double-spending (or triple-spending) attacks. We calculate zero-loss in expectation: with some probability, there can be periods of time where the stake taken from slashed replicas does not suffice to pay for the stolen funds from double-spent (or triple-spent) coins. In this case, ZLB simply mints extra coins (making every coin holder in the system implicitly pay for the successful attack). However, it is important to remember that, as zero-loss is guaranteed in expectation, an event in which there are not enough funds to pay for stolen coins would be unusual, since an attack is more likely to fail than to succeed, and so the reserve of slashed funds is more likely to grow with the number of attacks than it is to deplete (to a point in which the system can sustain even a catastrophic event leading to a large amount of funds being stolen). That is, our Blob system temporarily trades a disagreement and the resulting loss of funds and confidence in the system with temporary inflation. Thanks to zero-loss, this trade is only temporary, and inflation fades away as more attacks fail and pay for their own stolen coins along with those of old attacks that generated the inflation in the first place.

In this sense, ZLB also provides a mechanism for users to grow their confidence in the system as the number of failed attacks increases the reserve of slashed funds: ZLB can ensure that it has a reserve big enough to fight inflation and cope with attacks at the same time. The reserve growing can also allow for smaller deposits over time, or faster finalization. As a result, we argue that a temporary creation of coins (that will later be replaced for slashed funds) is a sensible approach for the described scenario.

c) Theoretical analysis: We show that attackers always fund at least as much as they steal. We consider that a membership change starts before a disagreement occurs or does not start, which is safer than the general case. Hence, the attack represents a Bernoulli trial that succeeds with probability ρ (per block) that can be derived from ε. Out of one attack attempt, the attackers may gain up to $(a-1)\mathfrak{G}$ coins by forking or lose at least $\mathfrak{D}$ coins as a punishment, which can be used to fund the stolen funds from successful attacks.

We introduce the random variable Y that measures the number of attempts for an attack to succeed and follows a geometric distribution with mean $E(Y) = \frac{1-\hat{\rho}}{\hat{\rho}}$, where $\hat{\rho} = 1-\rho$ is the probability that the attack fails. Thus, we define the expected gain of attacking: $\mathcal{G}(\hat{\rho}) = (a-1)\cdot(\mathbb{P}(Y > m)\cdot\mathfrak{G})$, and the expected punishment as: $\mathcal{P}(\hat{\rho}) = \mathbb{P}(Y \leq m)\cdot\mathfrak{D}$. We can then define the expected *deposit flux* per attack attempt as the difference $\Delta = \mathcal{P}(\hat{\rho}) - \mathcal{G}(\hat{\rho})$. Theorem VI.1 shows the values for which ZLB yields zero-loss.

Theorem VI.1 (Zero-Loss Payment System)**.** *Let ρ be the probability of success of an attack per block, $\mathfrak{D}$ the minimum deposit per coalition expressed as a factor of the upper-bound on the gain $\mathfrak{D} = b\mathfrak{G}$, and m the finalization blockdepth to return the deposit. If $g(a,b,\rho,m) = (1-\rho^{m+1})b - (a-1)\rho^{m+1} \geq 0$ then ZLB implements a zero-loss payment system.*

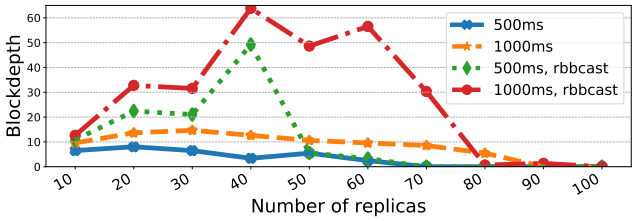

Fig. 6: Minimum finalization blockdepth m to obtain zero-loss for $\mathfrak{D} = \mathfrak{G}/10$, $f = \lceil 5n/9 \rceil - 1$.

Proof. Recall that the maximum gain of a successful attack is $\mathfrak{G}\cdot(a-1)$, and the expected gain $\mathcal{G}(\hat{\rho})$ and punishment $\mathcal{P}(\hat{\rho})$ for the attackers in a disagreement attempt are as follows:

$$\mathcal{G}(\hat{\rho}) = (a-1)\cdot(\mathbb{P}(Y > m)\cdot\mathfrak{G}) = (a-1)\cdot(\rho^{m+1}\cdot\mathfrak{G}),$$
$$\mathcal{P}(\hat{\rho}) = \mathbb{P}(Y \leq m)\cdot\mathfrak{D} = (1-\rho^{m+1})\mathfrak{D} = (1-\rho^{m+1})b\mathfrak{G}.$$

Thus the deposit flux $\Delta = \mathcal{P}(\hat{\rho}) - \mathcal{G}(\hat{\rho})$:

$$\Delta = \big((1-\rho^{m+1})b - (a-1)\rho^{m+1}\big)\mathfrak{G} = g(a,b,\rho,m)\mathfrak{G}.$$

If $\Delta < 0$ then a cost of $\mathcal{G}(\hat{\rho}) - \mathcal{P}(\hat{\rho})$ is incurred to the system, otherwise the punishment is enough to fund the conflicts. Since the gain is non-negative $\mathfrak{G} \geq 0$, it follows that $g(a,b,\rho,m) \geq 0$ for $\Delta \geq 0$, obtaining zero-loss. □

Setting $c = \frac{b}{a-1+b}$, we can either calculate the probability $\rho \leq c^{\frac{1}{m+1}}$ of success for an attack that ZLB tolerates given a finalization blockdepth m, or a needed finalization blockdepth $m \geq \frac{\log(c)}{\log(\rho)} - 1$ for a probability ρ to yield zero-loss, once we fix the deposit $\mathfrak{D}$ and upper-bound the gain $\mathfrak{G}$.

d) Finalization blockdepth and deposit size: A fault ratio $\delta = \frac{f}{n}$ holds a maximum number of branches a [67]. For example, for $\delta = 0.5$ then $a = 3$, and for a probability $\rho = 0.55$, a finalization blockdepth of $m = 4$ blocks guarantees zero-loss even if the deposit is a tenth of the maximum gain $\mathfrak{D} = \mathfrak{G}/10$, but with $\rho = 0.9$ then $m = 28$. Whereas a increases polynomially with ρ, it increases exponentially as the fault ratio δ approaches the asymptotic limit $2/3$, leading to $m = 37$ blocks for $\delta = 0.6$, while $m = 46$ for $\delta = 0.64$, or $m = 58$ for $\delta = 0.66$, with $\rho = 0.9$ and $\mathfrak{D} = \mathfrak{G}/10$.

e) Experimental evaluation of the payment system: Taking the experimental results of §V and based on our theoretical analysis, Figure 6 depicts the minimum required finalization blockdepth m for a variety of uniform communication delays for $\mathfrak{D} = \mathfrak{G}/10$, $f = \lceil 5n/9 \rceil - 1$. Again, we can see that the finalization blockdepth decreases with the number of replicas, confirming that the zero-loss property scales well. Additionally, small uniform delays yield zero-loss at smaller values of m, with all of them yielding $m < 5$ blocks for $n > 80$. The plot focuses on arguably catastrophic, uniform delays between partitions of correct replicas of 0.5s and 1s. More realistic delays, for example following Gamma distributions with realistic parameters, or AWS delays obtained in previously published measurements traces [56], [26], [25], barely achieve any disagreement in the first place (only 2 and 3 disagreements and only for $n = 20$), meaning that

the finalization blockdepth can be significantly smaller (a finalization blockdepth of 5 suffices for these cases). Despite the low likelihood of disagreements occurring in the presence of realistic latencies, ZLB is the first blockchain system that does not break in the presence of one or more disagreements. (as it does not need hard forks to resolve disagreements). Although omitted in the figure, our experiments showed that even for a uniform delay of 10 seconds, setting $m = 50$ blocks (resp. $m = 168$ blocks) yields zero-loss in the case of a binary consensus attack (resp. reliable broadcast attack). Nevertheless, if the network performs normally, ZLB will support large values of f, and will actually benefit from attacks, obtaining more than is lost.

VII. Related Work

Several works tried to circumvent the upper bound on the number of Byzantine failures [46] to reach agreement. As opposed to permissionless blockchains [57], some permissioned blockchains rotate the consensus participants to cope with an increasing amount of colluding replicas without perfect synchrony [72], [7]. Accountability has originally been applied to distributed systems in PeerReview [39] and to consensus in Polygraph [17], but not to recover from inconsistencies.

Slashing stakes aims at disincentivizing blockchain participants to misbehave. Casper [11] incurs a penalty in case of double votes but does not ensure termination even when $f < n/3$. Tendermint could become accountable with some transformation [9]. Balance [41] is a theoretical idea to adjust the size of the deposit to avoid over collateralizing. SUNDR [48] assumes honest clients that communicate directly to detect Byzantines. Polygraph [18] detect faults without excluding them. FairLedger [47] assumes synchrony for detection. Sheng et al. [66] consider a different definition of accountability that cannot be achieved with $2n/3$ faults. Freitas de Souza et al. [27] reconfigure replicas in a lattice agreement after detection. Shamis et al. [65] store messages in a ledger to punish replicas but requires $2n/3$ honest replicas to progress.

Traditionally, closed distributed systems consider that omission faults (omitting messages) are more frequent than commission faults (sending wrong messages) [21], [42], [51]. Zeno [67] guarantees eventual consistency by decoupling requests into weak (i.e., requests that may suffer reordering) and strong requests. ZLB could not be built upon Zeno because Zeno requires wrongly ordered transactions to be rolled back, whereas blockchain transactions can have irrevocable side effects like the shipping of goods to the buyer. BFT2F [49] offers fork* consistency, which forces the adversary to keep correct clients in one fork, while also allowing accountability. Stewart et al. [68] provide a finality gadget similar to our confirmation phase, however, it does not recover from disagreements.

The BAR model [1] is motivated by multiple administrative domains and corresponds better to the blockchain open networks, without distinguishing a-b-c faults. Upright [21] proposes a system that supports $n = 2u + r + 1$ faults, where u and r are the numbers of commission and omission faults, respectively. They can tolerate $n/3$ commission faults or $n/2$ omission faults. Flexible BFT [54] offers results to support $\lceil 2n/3 \rceil - 1$ alive-but-corrupt (abc) replicas. Flexible BFT requires clients to decide the exact fraction of abc and Byzantine replicas that they tolerate, tolerating for example $\lceil 2n/3 - 1 \rceil$ abc faults at the cost of no Byzantine replicas, or $\lceil n/3 \rceil - 1$ Byzantine replicas at the cost of no abc faults.

Some hybrid failure models tolerate crash failures and Byzantine failures but prevent Byzantine failures from partitioning the network [50]. Others aim at guaranteeing that well-behaved quorums are responsive [51] or combine crash-recovery with Byzantine behaviors to implement reliable broadcast [2]. Neu et al. [59] show an implementation of a system that provides availability in partial synchrony for $f < n/3$ and finality in synchrony for $f < n/2$. However, their system does not tolerate $n/3$ a-b-c failures in partial synchrony. They also extend this work to identify an accountability-availability dilemma that motivates the need for a-b-c replicas [60].

There have been various attacks against blockchains [58], [31], [61], especially leveraging forks. Most of the these attacks build upon disagreement and are usually the result of appending blocks before a unique block is chosen for the given index. Sometimes these errors are the results of flaws in the consensus protocols [31], sometimes it is due to the probabilistic nature of the consensus protocol [71] and sometimes it is just by design [57], [74], [11]. This is also the reason why ZLB builds upon a formally verified deterministic consensus protocol that does not assume synchrony [6]. In particular, ZLB is immune to the Balance Attack [58] because it tolerates partial synchrony and copes with other balancing attacks and nothing-at-stake attacks by eventually merging branches into a single branch.

VIII. Conclusion

In this paper, we proposed ZLB, the first blockchain that tolerates an adversary controlling a majority of failures by exploiting accountability and with alive but corrupt failures. ZLB outperforms HotStuff and achieves performance close to the Redbelly Blockchain.

Acknowledgments

We wish to thank the anonymous reviewers for their constructive feedback on an earlier version of this paper. This work is supported in part by the Australian Research Council Future Fellowship funding scheme (#180100496).

References

[1] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *SOSP*, 2005.

[2] M. Backes and C. Cachin. Reliable broadcast in a computational hybrid model with Byzantine faults, crashes, and recoveries. In *IEEE/IFIP DSN*, pages 37–46, 2003.

[3] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino. State machine replication in the libra blockchain. *The Libra Assn., Tech. Rep*, 2019.

[4] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC*, pages 52–61, 1993.

[5] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *PODC*, pages 183–192, 1994.

[6] N. Bertrand, V. Gramoli, I. Konnov, M. Lazic, P. Tholoniat, and J. Widder. Holistic verification of blockchain consensus. In C. Scheideler, editor, *DISC*, volume 246 of *LIPIcs*, pages 10:1–10:24, 2022.

[7] A. Bessani, E. Alchieri, J. ao Sousa, A. Oliveira, and F. Pedone. From byzantine replication to blockchain: Consensus is only the beginning. In *IEEE/IFIP DSN*, pages 424–436, 2020.

[8] G. Bracha. Asynchronous Byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[9] E. Buchman, R. Guerraoui, J. Komatovic, Z. Milosevic, D.-A. Seredinschi, and J. Widder. Revisiting tendermint: Design tradeoffs, accountability, and practical use. In *IEEE/IFIP DSN (Supplements)*, pages 11–14, 2022.

[10] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus. Technical Report 1807.04938, arXiv, 2018.

[11] V. Buterin and V. Griffith. Casper the friendly finality gadget. Technical Report 1710.09437v4, arXiv, Jan 2019.

[12] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

[13] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[14] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[15] B. Y. Chan and E. Shi. Streamlet: Textbook streamlined blockchains. In *ACM Advances in Financial Technologies (AFT)*, pages 1–11, 2020.

[16] P. Civit, S. Gilbert, and V. Gramoli. Polygraph: Accountable Byzantine consensus. In *Workshop on Verification of Distributed Systems (VDS)*, Jun 2019.

[17] P. Civit, S. Gilbert, and V. Gramoli. Brief announcement: Polygraph: Accountable byzantine agreement. In *DISC*, pages 45:1–45:3, 2020.

[18] P. Civit, S. Gilbert, and V. Gramoli. Polygraph: Accountable byzantine agreement. In *IEEE ICDCS*, Jul 2021.

[19] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, and J. Komatovic. As easy as ABC: Optimal (A)ccountable (B)yzantine (C)onsensus is easy! In *IEEE IPDPS*, 2022.

[20] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues. On the (limited) power of non-equivocation. In *PODC*, pages 301–308, 2012.

[21] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *SOSP*, pages 277–290, 2009.

[22] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y. A. Pignolet, D. Seredinschi, A. Tonkikh, and A. Xygkis. Online payments by merely broadcasting messages. In *IEEE/IFIP DSN*, pages 26–38, 2020.

[23] T. Crain, V. Gramoli, M. Larrea, and M. Raynal. DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In *IEEE NCA*, 2018.

[24] T. Crain, C. Natoli, and V. Gramoli. Evaluating the Red Belly Blockchain. Technical Report 1812.11747, arXiv, 2018.

[25] T. Crain, C. Natoli, and V. Gramoli. Red Belly: A secure, fair and scalable open blockchain. In *IEEE S&P*, 2021.

[26] M. E. Crovella and R. L. Carter. Dynamic server selection in the Internet. In *IEEE HPCS*, 1995.

[27] L. F. de Souza, P. Kuznetsov, T. Rieutord, and S. Tucci Piergiovanni. Brief announcement: Accountability and reconfiguration - self-healing lattice agreement. In *DISC*, pages 54:1–54:5, 2021.

[28] S. Duan, M. K. Reiter, and H. Zhang. BEAT: asynchronous BFT made practical. In *CCS*, pages 2028–2041, 2018.

[29] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[30] P. Ekparinya, V. Gramoli, and G. Jourjon. Impact of man-in-the-middle attacks on ethereum. In *IEEE SRDS*, 2018.

[31] P. Ekparinya, V. Gramoli, and G. Jourjon. The attack of the clones against proof-of-authority. In *NDSS*, 2020.

[32] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *USENIX NSDI*, pages 45–59, 2016.

[33] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, June 2018.

[34] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT*, pages 281–310, 2015.

[35] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68, 2017.

[36] S. Gilbert and N. Lynch. Perspectives on the CAP theorem. *Computer*, 45(2):30–36, 2012.

[37] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. SBFT: a scalable decentralized trust infrastructure for blockchains. In *IEEE/IFIP DSN*, pages 568–580, 2019.

[38] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous BFT protocols. In *CCS*, pages 803–818, 2020.

[39] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *SOSP*, 2007.

[40] I. Harel, A. Jacob-Fanani, M. Sulamy, and Y. Afek. Consensus in Equilibrium: Can One Against All Decide Fairly? In *OPODIS*, 2020.

[41] D. Harz, L. Gudgeon, A. Gervais, and W. J. Knottenbelt. Balance: Dynamic adjustment of cryptocurrency deposits. In *CCS*, 2019.

[42] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin. All about eve: Execute-verify replication for multi-core servers. In *USENIX OSDI*, pages 237–250, 2012.

[43] C. E. Kelso. Bitcoin gold hacked for $18 million. https://news.bitcoin.com/bitcoin-gold-hacked-for-18-million/, Accessed: 2021-01-31.

[44] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE S&P*, pages 583–598, 2018.

[45] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *SOSP*, 2007.

[46] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[47] K. Lev-Ari, A. Spiegelman, I. Keidar, and D. Malkhi. Fairledger: A fair blockchain protocol for financial institutions. Technical Report 1906.03819, arXiv, 2019.

[48] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *USENIX OSDI*, page 9, 2004.

[49] J. Li and D. Mazières. Beyond one-third faulty replicas in Byzantine fault tolerant systems. In *USENIX NSDI*, page 10, 2007.

[50] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic. XFT: practical fault tolerance beyond crashes. In *USENIX OSDI*, pages 485–500, 2016.

[51] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb. Fast and secure global payments with stellar. In *SOSP*, pages 80–96, 2019.

[52] Y. Lu, Z. Lu, Q. Tang, and G. Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous Byzantine agreement, revisited. In *PODC*, pages 129–138, 2020.

[53] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *CCS*, 2016.

[54] D. Malkhi, K. Nayak, and L. Ren. Flexible Byzantine fault tolerance. In *CCS*, pages 1041–1053, 2019.

[55] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *CCS*, pages 31–42, 2016.

[56] A. Mukherjee. On the dynamics and significance of low frequency components of Internet load. Technical Report MS-CIS-92-83, University of Pennsylvania, 1992.

[57] S. Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[58] C. Natoli and V. Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *IEEE/IFIP DSN*, 2017.

[59] J. Neu, E. Tas, and D. Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *IEEE S&P*, pages 446–465, 2021.

[60] J. Neu, E. N. Tas, and D. Tse. The availability-accountability dilemma and its resolution via accountability gadgets, 2021.

[61] J. Neu, E. N. Tas, and D. Tse. Two more attacks on proof-of-stake ghost/ethereum. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, 2022.

[62] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[63] A. Ranchal-Pedrosa and V. Gramoli. Basilic: Resilient optimal consensus protocols with benign and deceitful faults. In *IEEE CSF*, 2023.

[64] J. Redman. Bitcoin gold 51% attacked - network loses $70,000 in double spends. https://news.bitcoin.com/bitcoin-gold-51-attacked-network-loses-70000-in-double-spends/ Accessed: 2021-01-31.

[65] A. Shamis, P. Pietzuch, M. Castro, E. Ashton, A. Chamayou, S. Clebsch, A. Delignat-Lavaud, C. Fournet, M. Kerner, J. Maffre, et al. PAC: Practical accountability for CCF. Technical report, arXiv, 2021.

[66] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath. BFT protocol forensics. In *CCS*, 2021.

[67] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, P. Maniatis, et al. Zeno: Eventually consistent Byzantine-fault tolerance. In *USENIX NSDI*, pages 169–184, 2009.
[68] A. Stewart and E. Kokoris-Kogia. Grandpa: a byzantine finality gadget. Technical Report 2007.01560, arXiv, 2020.
[69] D. Tennakoon, Y. Hua, and V. Gramoli. Smart redbelly blockchain: Reducing congestion for web3. In *IPDPS*, pages 940–950. IEEE, 2023.
[70] P. Tholoniat and V. Gramoli. Formal verification of blockchain byzantine fault tolerance. In *FRIDA*, Oct 2019.
[71] P. Tholoniat and V. Gramoli. *Formal Verification of Blockchain Byzantine Fault Tolerance*, pages 389–412. Springer International Publishing, 2022.
[72] G. Vizier and V. Gramoli. ComChain: A blockchain with byzantine fault tolerant reconfiguration. *Concurrency and Computation, Practice and Experience*, 32(12), Oct 2020.
[73] G. Voron and V. Gramoli. Dispel: Byzantine SMR with distributed pipelining. Technical Report 1912.10367, arXiv, Dec. 2019.
[74] G. Wood. Ethereum: A secure decentralized generalized transaction ledger, 2015.
[75] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT consensus in the lens of blockchain. Technical Report 1803.05069, arXiv, July 2019. Version 6 (accessed 21 May 2019).
[76] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*, 2019.
[77] M. Zamani, M. Movahedi, and M. Raykova. RapidChain: Scaling blockchain via full sharding. In *CCS*, pages 931–948, 2018.
[78] T. Zimwara. $5.6 million double spent: Etc team finally acknowledges the 51% attack on network. https://news.bitcoin.com/5-6-million-stolen-as-etc-team-finally-acknowledge-the-51-attack-on-network/. Accessed: 2021-01-31.