Stabl: The Sensitivity of Blockchains to Failures

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

Rachid Guerraoui
EPFL
Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Andrei Lebedev
University of Sydney
Sydney, Australia
andrei.lebedev@sydney.edu.au

Gauthier Voron
EPFL
Lausanne, Switzerland
gauthier.voron@epfl.ch

Abstract

Blockchains promise to make online services more fault tolerant because they are replicated on a distributed system of nodes. Their nodes typically run different implementations of the same protocol across different geo-distributed regions, making the protocol supposedly tolerant to various failures including isolated crashes, transient failures, network partition or attacks. Unfortunately, their fault tolerance has never been compared.

In this paper, we provide the first fault tolerance comparison of blockchain systems. To this end, we introduce a novel *sensitivity* metric, interesting in its own right, as the responsiveness difference between a baseline environment and an adversarial environment. We inject various failures in controlled deployments of five modern blockchain systems, namely Algorand, Aptos, Avalanche, Redbelly and Solana. Our results show that (i) all blockchains except Redbelly are highly impacted by isolated failures, (ii) Avalanche and Redbelly benefit from the redundant information needed for Byzantine fault tolerance while others are hampered by it, and more dramatically (iii) Avalanche and Solana cannot recover from transient failures.

CCS Concepts: • **Computer systems organization → Reliability**.

Keywords: Dependability, Partition, Recovery, Sensitivity

ACM Reference Format:
Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, and Gauthier Voron. 2025. Stabl: The Sensitivity of Blockchains to Failures. In *Proceedings of 26th International Middleware Conference (MIDDLEWARE '25).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

MIDDLEWARE '25, December 15–19, 2025, Nashville, TN, US

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

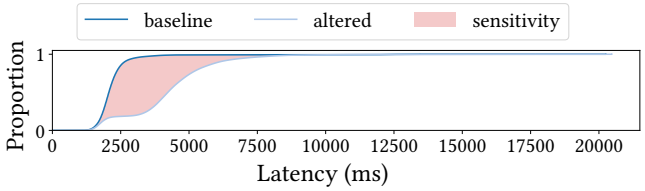

Figure 1. The *sensitivity* of Aptos to failures as the difference in latency distributions between a baseline environment without failure and the altered environment with failures.

1 Introduction

One may think that blockchains [55] are fault tolerant. They are distributed systems replicated across nodes in geodistributed regions making it unlikely to be affected by a single natural disaster. These nodes often run different implementations of the same protocol that would unlikely suffer the exact same bug [14]. This mitigates the risk of a unique bug leading to a global outage like the CrowdStrike one that affected 8.5 millions of machines recently [22]. Finally, the owners of these nodes are typically incentivized through cryptoassets to ensure their node run correctly [60].

However, blockchains experience frequent outages. Solana for example aims at running decentralized applications but experienced 9 outages between September 2021 and February 2023 for a cumulative outage duration of 154.5 hours [36]. In terms of service level agreement (SLA) this translates into offering a service whose availability ($< 99\%$) fails to reach two nines, while traditional cloud services offer three nines ($\geq 99.9\%$). This questions the ability of blockchains to tolerate failures. Unfortunately, blockchains have mostly been evaluated without failures [7, 34, 43, 54, 56].

In this paper, we provide the first fault tolerance comparison of blockchain systems. First, inspired by the supercumulative distribution function (SDF) used in economics [20], we introduce the *sensitivity* metric of a blockchain to adversarial environments. To illustrate the sensitivity metric,

consider Fig. 1 that depicts two *empirical Cumulative Distribution Functions (eCDFs)* of latencies of the Aptos blockchain (experimental settings are deferred to Section 3). The first distribution illustrated with the blue curve corresponds to a baseline environment without failures. The other distribution illustrated with the light blue curve corresponds to an altered environment where we injected failures. The sensitivity score of Aptos for this type of injected failures, represented as the pink area, is the difference of the areas under the two eCDFs.

Second, we implement Stabl (Sensitivity Testing and Analysis for BLockchain), pluggable in continuous integration (CI) pipelines to measure a blockchain sensitivity, and compare the fault tolerance of Algorand [40], Aptos [1], Avalanche [61], Redbelly [32] and Solana [11], selected for their ability to tolerate arbitrary (i.e., Byzantine) failures [48]. Given that consensus can only be solved between n nodes in the presence of $t < n/3$ permanent failures in an open network [35], we study the following properties that depend on the type and number t_B of failures we inject in blockchain $B \in \{Algorand, Aptos, Avalanche, Redbelly, Solana\}$: *(i)* Resilience: the insensitivity to $f = t_B$ definitive crash (or fail-stop) failures; *(ii)* Recoverability: the insensitivity to $f > t_B$ transient (or crash-recovery) failures; *(iii)* Partition tolerance: the insensitivity to the transient partition of $f > t_B$ nodes; and *(iv)* Byzantine node tolerance: the insensitivity to a mechanism to cope with $f = t_B$ Byzantine nodes.

Our results demonstrate that fault tolerance varies greatly with the choice of blockchain system. First, we confirm that all of these blockchains, except Redbelly, are significantly affected by failures. Second, we show that Avalanche and Solana cannot tolerate transient failures and stop working. Finally, we show how sending duplicated transactions to cope with Byzantine faults can reduce or improve the responsiveness of blockchain systems. Note that our intention is not to address any failures but to report the failures so that blockchain developers can improve the fault tolerance of their systems.

The paper is organised as follows. Section 2 presents the related work. Section 3 presents our solution. Sections 4, 5, 6 and 7 present respectively the resilience, the recoverability, the partition tolerance and the Byzantine node tolerance of the blockchains. Section 8 discusses our results. Finally, Section 9 concludes the paper.

2 Background and Related Work

We present the related work and the studied blockchains.

Related work. The impairments and remedies of dependability of software systems have been studied for more than four decades [18, 49]. A long series of work studied in particular the *Byzantine* fault tolerance [48] as the tolerance to arbitrary failures. It is more recently that blockchain security flaws [37, 66] were identified and that the research community started studying blockchain dependability [57]. A long series of blockchain security vulnerabilities can now be found in surveys and books [26, 42, 58].

Interestingly, two recent works [17, 63] observed vulnerabilities in the only two blockchains, Avalanche and Solana, that failed during our experiments. First, a theoretical analysis of Avalanche consensus protocols, Snowball and Snowflake, indicate that they do not offer a "decent" trade-off between security and performance [17]. Second, previous experiments showed that Solana could fork permanently [63], however, our observation is different as we noticed that all the nodes of Solana crash after an injection of transient communication delays.

Unfortunately, as of today there is no tool that allows to systematically compare the fault tolerance of blockchain systems. Few research works injected Byzantine failures [32, 60] or system call failures [73], others use fuzzing [52, 53, 69, 71], add network delays [39] or configuration parameters [28] for evaluating a particular blockchain. They cannot compare the fault tolerance of different blockchains on the same ground. Other results focus instead on injecting crash [68] or Byzantine failures [16, 21, 51, 62] in Byzantine Fault Tolerance (BFT) replicated state machines and ignore other components of the blockchain system. Actually, blockchain evaluation frameworks focus on performance in fault-free executions [7, 34, 43, 54, 56].

Algorand. Algorand [40] is a blockchain that leverages cryptographic sortition through Verifiable Random Functions (VRFs) to randomly select participants for specific roles in the consensus execution. Each participant independently computes a pseudo-random value and a proof, determining their selection for roles such as consensus participant. The Byzantine Agreement ($BA\star$) protocol then uses the consensus participants to propose and validate new blocks, reaching consensus even in the presence of Byzantine faults. This dynamic selection process ensures unpredictable and ever-changing committee membership, enhancing the blockchain security. We select $t_{Algorand}$ to be $\lceil n/5 - 1 \rceil$ as a coalition can fork Algorand if it controls 20% of its currency [40].

To optimize network performance, Algorand adjusts the consensus protocol's timing based on real-time network conditions using Dynamic Round Time [29], ensuring efficient block production while accommodating slower nodes. *Relay* nodes and *participation* nodes have distinct roles, with relay nodes handling data propagation and participation nodes focusing on transaction validation and consensus. However, a single node can fulfill both functions. Transaction propagation is managed through push and pull gossip methods, with push gossip actively broadcasting transactions while pull gossip enabling nodes to request missing transactions, ensuring efficient data synchronization across the network.

Avalanche. Avalanche is a blockchain based on the Snow binary consensus protocol family [61]. The Snowflake protocol uses three parameters: k, $\alpha > k/2$, and β. Initially, each processor starts with a color, either red or blue. The protocol proceeds in rounds, where in each round, a processor p randomly selects k other processors from the entire population and queries them about their current color. If at least α of the responses differ from p's current color, p switches to that opposite color. If p observes β consecutive rounds where at least α of the responses are red (resp. blue), then p decides on red as the final color (resp. blue). With the default parameter values, Avalanche requires at least 80% of stake to be online for consensus to operate. We select $t_{Avalanche}$ to be $\lceil n/5 - 1 \rceil$ as in the presence of 20% Byzantine nodes Avalanche safety is violated with a probability below 10^{-9} [61].

Avalanche offers throttling to limit its node resource usage. Message rate-limiting and connection rate-limiting [2] limits the amount of CPU, disk, bandwidth, and message handling a node consumes. In particular, the message rate-limiting can be configured based on CPU usage, disk reads/writes, bandwidth usage, and the size and number of unprocessed messages between validators and non-validators, the maximum burst size for bandwidth, and limits on the number of unprocessed messages. Finally, the connection rate-limiting controls the rate of inbound and outbound peer connections, including the maximum number of connections accepted per second and the frequency of connection attempts. We will discuss how throttling impacts recovery in Section 4.

Aptos. Aptos [1] is a blockchain based on a variant of the HotStuff consensus algorithm [72] called DiemBFT [65], then renamed AptosBFT. In particular, DiemBFT features a quadratic view-change mechanism instead of the linear approach used in HotStuff, and inherits the cubic communication complexity of the *Practical Byzantine Fault Tolerant (PBFT)* consensus protocol [27] that is reached when a leader is faulty or the network is unstable. It is thus a *leader-based* blockchain that tolerates up to a third of malicious participants in a partially synchronous environment and that requires view-changes in order to cope with faulty leaders. We select $t_{Aptos} = \lceil n/3 - 1 \rceil$ as Aptos tolerates a coalition of less than a third of validator nodes [33].

Interestingly, Aptos also features the *Block-STM* [38] design that optimizes the execution of blockchain transactions through Software Transactional Memory (STM), hence the name. In Block-STM, parallel execution leverages multiple threads to execute different transactions concurrently, provided they access distinct memory locations. Aptos executes transactions speculatively to dynamically manage conflicts based on a pre-determined order, but without pre-computing dependencies. When conflicts arise, transactions are aborted and re-executed with their write-sets used to predict and minimize future conflicts.

Redbelly. Redbelly Blockchain [32] is a scalable blockchain that builds upon the Democratic Byzantine Fault Tolerant (DBFT) consensus algorithm [31] that is *leaderless (non leader-based)* and deterministic, and works in a partially synchronous environment. DBFT has been formally verified with parameterised model checking [24], showing that it solves the consensus problem in all possible executions and for any system size. To enhance scalability further, Redbelly uses a collaborative approach, hence appending a superblock comprising as many valid proposed blocks as possible. This way the number of transactions per appended block can grow linearly with the number of nodes [32]. We select $t_{Redbelly} = \lceil n/3 - 1 \rceil$ as Redbelly tolerates a coalition of less than a third of validator nodes [32].

We used the latest version of Redbelly that features the Scalable version of the Ethereum Virtual Machine (SEVM) that runs *decentralised applications (dApps)* written in Solidity [64] that samples periodically a set of consensus participants among all participants [42]. This version was shown to perform well under realistic dApps particularly in a large geo-distributed environment when compared to other modern blockchains [64]. For the sake of security and Byzantine node tolerance, Redbelly features a library tolerating $f < n/3$ Byzantine nodes, called `credence.js`, for a read operation to return values that are replicated at at least $f + 1$ nodes.

Solana. Solana [11] is a blockchain that operates on a pre-determined leader schedule, assigning each validator a specific time *slot*, to produce a block within a larger time frame called an *epoch*. The leader schedule, computed in advance using a pseudo-random algorithm based on data from two epochs prior, ensures validators are chosen proportionally to their stake. This schedule is updated at the end of each epoch and communicated to validators beforehand. A core structure in Solana runtime is the *bank*, which represents the blockchain state at a specific slot, managing transactions, account states, and ensuring adherence to rules during transaction processing. Each bank processes transactions for its assigned slot and, upon completion, finalizes a frozen state that includes a cryptographic hash crucial for network consensus. We select $t_{Solana} = \lceil n/3 - 1 \rceil$ as Solana tolerates a coalition of less than a third of validator nodes [70].

Solana runtime includes a mechanism for calculating the *Epoch Accounts Hash (EAH)*, a hash of all accounts, to ensure consistency across validators during each epoch. The EAH is computed between the start and stop slots, typically from one-quarter to three-quarters into an epoch, and integrated into the bank's hash for consensus verification. Notably, Solana does not use a memory pool (or *mempool* for short), forwarding transactions directly to the current and upcoming leaders based on the known leader schedule [9]. If a leader cannot process a transaction in its assigned slot, it passes the responsibility to the next leader.

Table 1. Terminology and notation used in the paper.

Term	Description
Crash	node is halted and not restarted during the experiment
Transient failure	node is halted and restarted later during the experiment with the same identity
Partition	missing network connectivity between subsets of nodes
Leader	node responsible for proposing a block in the current consensus round
Sensitivity	a measure quantifying the change in transaction latencies in response to variations in the execution environment
Resilience	a measure quantifying the system latency under failures
Recoverability	ability to recover after a transient failure
f	number of failures in an experiment
t_B	maximum number of failures tolerated by a blockchain B
n	number of nodes in a blockchain network

3 Measuring Blockchain Sensitivity

In this section, we introduce the sensitivity score to measure the fault tolerance of blockchain systems and explain how we developed a tool called Stabl to measure it. We summarize the key terms in Table 1.

Sensitivity score. Previous works [44] rely on three metrics for their evaluation: the latency, the throughput and the downtime. On the one hand, the latency and throughput metrics quantify the magnitude of the impact of failures on a system and are therefore well suited for permanent failures of a portion of the distributed system. On the other hand, the downtime quantifies the duration of the effect of failures on the system and is better suited to transient failures. In order to compare the impact of different types of failures on the same blockchain, Stabl uses the *sensitivity score*, a metric that quantifies both the amplitude and the duration of an effect over a blockchain execution. We define the sensitivity score of a blockchain under a constant workload and in the face of some failures as a function of two latency distributions: one distribution measured in the absence of failures and one with failures. This function is a mapping from these two distributions to a number defined as the area between the eCDFs of the two distributions.

Let X be a random variable, which can take any value between a and b, with a CDF F. The super-cumulative distribution function, or simply super-cumulative [20], is defined as $S(x) = \int_a^x F(t)dt$.

In the setting of transaction latencies, let $(X_1, ..., X_m)$ be the values of a random variable. The eCDF is given by $\hat{F}(x) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{X_i \leq x}$, where the sum denotes the number of elements in the sample that are less than or equal to x. Then, we adapt the super-cumulative for the eCDF as $\hat{S}(x) = \sum_{i=a}^{x} \hat{F}(i)$.

Consider X_1 as the baseline latency measurements with values between a_1 and b_1, and X_2 as the latency measurements in the altered setting with values between a_2 and b_2, and their corresponding empirical super-cumulatives $\hat{S}_1$ and $\hat{S}_2$. The difference $\hat{S}_1(b_1) - \hat{S}_2(b_2)$ measures the change in the distribution of latencies from the baseline to the altered environment, as seen in Fig. 1. However, it is possible that the altered condition improves the performance of a blockchain and decreases transaction latencies [25, 45], in which case $\hat{S}_2(b_2)$ will be greater than $\hat{S}_1(b_1)$, producing a negative value of the difference. Since we are measuring sensitivity as the deviation from the baseline, we take an absolute value of the difference, so that the score is always positive, hence the sensitivity score is calculated as $|\hat{S}_1(b_1) - \hat{S}_2(b_2)|$.

The sensitivity score has the following valuable properties that make it an illustrative metric, as shown in Fig. 1:

- It measures both the amplitude and the duration of failure effects. Both factors skew the latency distribution of an experiment, resulting in an increased difference between the areas of two empirical super-cumulatives.
- It is resilient to outliers. A smaller fraction of particular latency values does not contribute significantly to the difference between the areas of two empirical super-cumulatives.
- It does not require a parameter for interpretation. For example, we do not need a sliding window to explain the score, which may be required to calculate throughput in transactions per second, because the block times might be greater than one second.
- It is an absolute metric. It allows direct comparison of scores between blockchains and experiments, since it is a function of transaction latencies.

Finally, notice that a blockchain that stops committing transactions after a failure event has an infinite sensitivity score, which indicates a liveness issue.

Stabl. To calculate sensitivity score, we developed Stabl, a benchmark suite to evaluate blockchains behavior in the presence of faulty processes. Stabl is built on top of Diablo [43], an open source software to assess the performance of blockchains under realistic but benign workloads. Stabl automatically evaluates and compares the ability of several blockchains to tolerate various types of faults. Specifically,

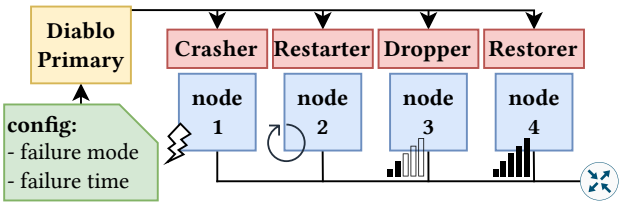

Figure 2. The architecture of Stabl passes fault configuration to observers deployed on each blockchain node.

Stabl evaluates the behavior of blockchains in the face of both permanent and transient failures. In order to accurately evaluate distributed systems, Diablo is itself a distributed system with two types of machines: the *primary* machine which acts as a central coordinator for the run and many *secondary* machines which simulate clients by submitting transactions to the blockchain processes and waiting for their response.

Observer nodes. Stabl extends the architecture of Diablo in order to control failures during the execution.

Fig. 2 depicts the architecture of Stabl. Unlike simulated clients, failure events take place on or between the blockchain machines. Therefore, neither the primary nor the secondary machines are suitable to trigger failures. Instead, Stabl uses *observer* processes which run on every blockchain machine and listen to a signal coming from the primary machine. When the primary decides to trigger a failure on one or many blockchain machines, it broadcasts a signal to the relevant observers. To implement crash faults, the observer processes simply kill the blockchain process running on their node. To implement a partition, the observer processes use the `netfilter` interface of their node to drop any IP packet coming from and going to other partitions. Observer nodes can also end the network partition by removing `netfilter` rules or reboot the blockchain process.

Dependability attributes. Achieving good performance despite faults, or the *resilience* metric has received some attention for Byzantine Fault Tolerant state machine replication systems [19, 23, 30, 41, 67]. The metric captures how the system performs with crashed nodes being present in the network, when up to f servers are non-responsive, compared to baseline execution, when all the servers behave correctly.

We measure *recoverability* of a blockchain as its ability to recover after a transient failure, where the number of failures is greater than the threshold, $f > t$.

From the user perspective, *partitions* display the same behavior as transient faults, as they both result in nodes not being able to exchange the messages. However there is a difference from the implementation perspective. While recovery from transient faults is active, since the restarted nodes immediately report their status to the rest of the network after being started, partition recovery can be considered passive, because the nodes cannot detect that the network connectivity was restored without constant polling.

Assessing fault tolerance. Blockchains are known to tolerate coalitions that represent a portion of the network, rather than an absolute number of faulty nodes. For example, for Avalanche to progress, the owners of 80% of the stake have to be online [15]. It is also well known that consensus cannot be solved in networks where the upper bound on the delay of messages is unknown and where a third of the participants are Byzantine [35]. This is why, in our experiments, we will set the number of permanent failures to a portion lower than the threshold t_B claimed to be tolerated by blockchain B and a number of transient failures greater than t_B. This will allow us to run experiments on small and medium size networks rather than large-scale networks for simplicity in making our experiments reproducible.

As well as simulating failures, Stabl differs from Diablo and previous evaluation platforms by implementing Byzantine node tolerance. Indeed, a common practice in blockchain client applications is to reach for a single blockchain node and trust it for transmitting the client transactions and relaying the responses from the network. For example 4 out of the 5 evaluated blockchains (with the exception of Redbelly as we will explain in Section 7) provide an SDK for client applications that connect to and trust a single blockchain node [4, 5, 10] even though this vulnerability has already been reported [46]. Trusting one specific node effectively brings the number of tolerated Byzantine faults to zero and can lead to devastating cyberattacks [47, 66].

A common solution is to send the same requests to many, randomly picked, blockchain nodes and compare their responses to detect any faulty response. Thanks to the deduplication mechanisms, legitimate transactions are executed only once while their results can be observed many times. This technique however puts an additional load on blockchain nodes as they must deduplicate redundant transactions. Moreover, this technique likely increases each transaction latency since clients must wait for the slowest of many blockchain nodes instead of one. We show in Section 7 that the effect of Byzantine node tolerance on transaction latency is twofold: it may benefit the transaction latency in mempool-based blockchains, and it may cause redundant transaction execution, even with transaction deduplication mechanisms.

Experimental settings. To deploy our blockchain networks, we used virtual machines spawned on a real network of physical machines. As our focus is on fault tolerance and not peak performance, we fixed the total sending rate to 200 TPS, a value low enough to guarantee that no blockchains would drop transactions in our baseline executions. In particular, Avalanche capacity is limited to about 357 TPS because its blocks are produced every 2 seconds and contain a maximum of 714 transactions (as its block limit is 15M gas while the transfer fee is 21K gas).

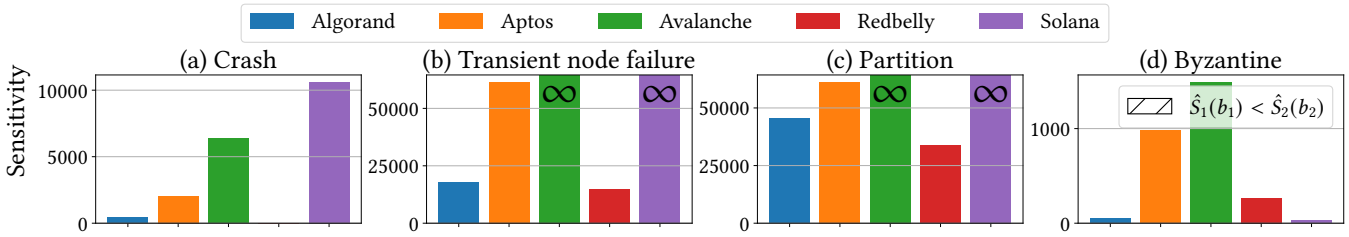

Figure 3. Sensitivity score of 5 blockchains with $f = t$ crashes, $f = t + 1$ transient node failures, transient network partition isolating $f = t + 1$ nodes and redundant requests to cope with Byzantine fault tolerance. Bars with stripes indicate blockchains benefiting from the altered environment.

As opposed to Diablo that measures blockchain performance at large scale, Stabl measures sensitivity to failures affecting a proportion of the blockchain network. This is why we did not reproduce the large-scale experiments of Diablo. Following recent observations [50] that blockchain networks behave similarly at large scale as at a much smaller scale, we deployed Stabl on a distributed system of 15 nodes. Each node runs as a virtual machine (VM) of 4 vCPUs and 8 GB of memory and each of the 5 client nodes sends at the same rate of 40 TPS to only one of the blockchain node. The setup consists of 5 client nodes and 10 blockchain nodes, each client sending native transfer transactions to one blockchain node. Failures are injected on the 5 remaining blockchain nodes that do not receive transactions from clients, this way, faulty nodes never receive transactions they would otherwise lose.

To assess the Byzantine node tolerance of blockchains in Section 7 we connected each client to 4 blockchain nodes such that each of 5 blockchain nodes has two clients connected to it. This duplication of requests increased the CPU consumed by the speculative execution of Aptos, which required us to allocate more resources. We thus used VMs with 8 vCPUs and 16 GB of memory in the Byzantine fault tolerant experiment of each blockchain (Section 7). All the VMs are run on a Proxmox cluster of physical servers, each equipped with 4x AMD Opteron 6378 16-core CPUs running at 2.40 GHz, 256 GB of RAM, and 10 GbE NICs. We used the following versions of the blockchains: Algorand v3.22.0, Aptos v1.9.3, Avalanche C-Chain v1.10.18-rc.2, Redbelly v0.36.2 and Solana v1.18.1.

In the following sections, we use the sensitivity to different types of failures to evaluate the Resilience, Recoverability, Partition Tolerance and Byzantine Node Tolerance of blockchain whose results are summarized in Fig. 3.

4 Resilience

In this section we evaluate the resilience of the five tested blockchains. Our conclusion from the sensitivity score to permanent failures is that all blockchains but Redbelly lack resilience. This is due to these blockchains relying on a set of specific servers to make progress at each decision. In particular, Avalanche and Solana are the least resilient with Solana experiencing higher sensitivity due to better performance in the baseline condition.

Assessing resilience. The test consists of comparing transaction latencies with constant workload in two experiments. The first experiment captures transaction latencies in a fault-free case. In the second experiment, we crash simultaneously at time 133 seconds f blockchain nodes.

Fig. 3a compares the sensitivity of blockchains when $f = t$ nodes experience permanent crash. Fig. 4 compares the throughput over time in the baseline and altered conditions and offers complementary data to explain the cause of the sensitivity differences between blockchains.

Solana leader impacts performance. The throughput instability in Solana can be explained by the design decision of not having a mempool [9]. Instead of every node maintaining a temporary storage for transactions, nodes send them directly to scheduled leaders. While such design decision may improve the performance in the best case scenario, when the scheduled leader processes the transactions, it leads to a snowball effect when a scheduled leader is non-responsive. With the constant workload, the new leader has to process a higher volume of transactions since one or more scheduled leaders are down. Hence, with crashed nodes being present in the leader schedule, we observe periods of low throughout when the scheduled leader is down, and throughput peaks when the transactions are processed by a responsive node, resulting in higher latencies, and therefore higher score.

Avalanche throttling leads to instability. In Fig. 4, we observe that Avalanche throughput is unstable. This is explained by its throttling mechanism. Several voting rounds should successfully pass in succession to commit a block. Since nodes are sampled for every voting round from all the nodes in the network, in the presence of crashes, faulty nodes may be included in the samples as well.

Intuitively, even with node crashes, the repeated sampling should allow the network to come to an agreement on a

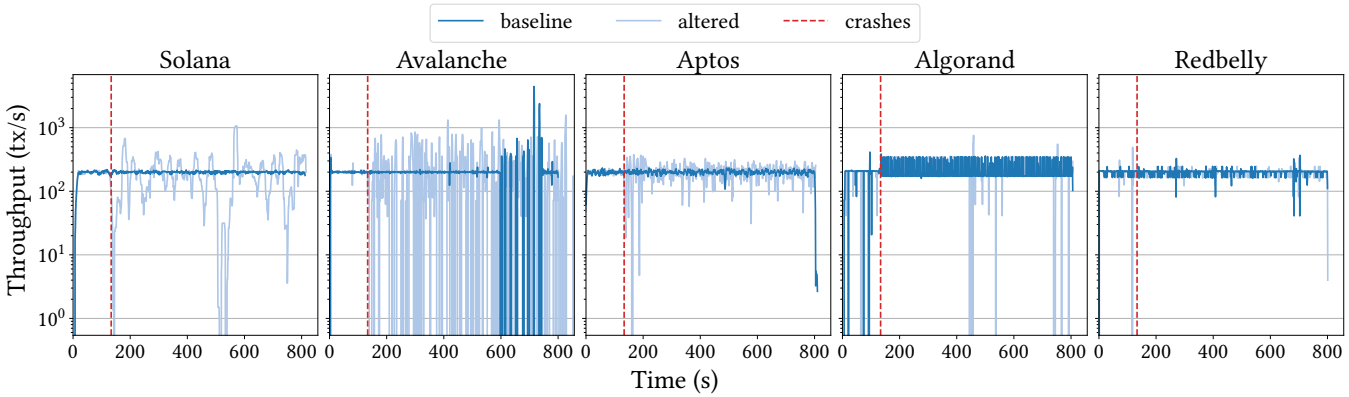

Figure 4. Throughput of the 5 blockchains over time as we crash simultaneously $f = t$ nodes at time 133 as indicated by the red dashed line.

block. However, as mentioned in Section 2 the current implementation includes multiple layers of message throttling based on CPU usage, bandwidth, and number of messages. The nodes exchange the messages, including transactions and consensus data, and the messages are first stored in queues before being processed. With the 200 TPS constant workload and default throttling settings, the nodes do not process the messages, even though the messages are sent and received over the network. The nodes consuming their respective CPU quotas cause the messages not to be processed, leading to messages not reaching the consensus module and increasing the throughput instability.

Additionally, we discovered a previously reported bug [8] with the help of Stabl. However, after running the experiments with a fixed version, we did not observe a measurable performance improvement because throttling has greater impact on performance than the bug.

Aptos mitigates the leader impact. Aptos displays significant oscillations immediately after the crashes, however unlike Solana and Avalanche, the throughput instability reduces in about 82 seconds at the 215 second timepoint. This behavior matches the description of the DiemBFT protocol [65]. While we tested a network of 10 nodes and observed noticeable performance drop, we can expect the performance to get increasingly worse with the growth of the network size.

Algorand adapts slowly to sudden failures. Algorand throughput depends on the timing parameters, which are calculated dynamically based on the observed round finalization times. Since the servers are selected using the VRF of $BA^\star$, samples may include crashed nodes, which increase the round finalization time. Initially, default timing parameters are used, which are then reduced, explaining the throughput increase after approximately 133 seconds have passed since the start of the experiment. In the presence of crashes, there are periods when the decreased timing parameters are reset to their default values, which reduces the average throughput and increases transaction latency. Such periodic increases in latency are reflected in the score.

Redbelly eradicates the leader impact. Redbelly is not affected by the presence of $f = t$ crashes. The reason is that Redbelly uses the leaderless consensus algorithm, called DBFT. In particular, Redbelly features a Byzantine fault tolerant binary consensus algorithm and a classic reduction from the multi-value consensus problem to the binary consensus problem.

First, Redbelly is not affected by the slow reponsive node that affects Solana because no individual slow node can significantly slow down the DBFT consensus protocol. More specifically, even if its binary consensus algorithm uses a weak coordinator to break symmetry, a faulty weak coordination does not prevent the DBFT consensus algorithm from converging towards a decision [31].

Second, Redbelly does not show the sign of oscillation of Aptos. As confirmed by previous results [68] this is due to DBFT being less impacted than HotStuff-like protocols (including DiemBFT) when their leader crashes. As a result, the leaderless consensus protocol reduces the effect that of not only a single slow node but also a single crashed node have on the overall blockchain execution.

5 Recoverability

In this section, we evaluate the recoverability to transient failures of the 5 tested blockchains. We can conclude from our results that two blockchains, Avalanche and Solana, cannot recover. The other blockchains recover at varying speeds.

Assessing recoverability. To test recovery we run an experiment that starts with no failures. We then halt $f = t + 1$ nodes by killing their process simultaneously at 133 seconds and recover them simultaneously at 266 seconds by restarting their process.

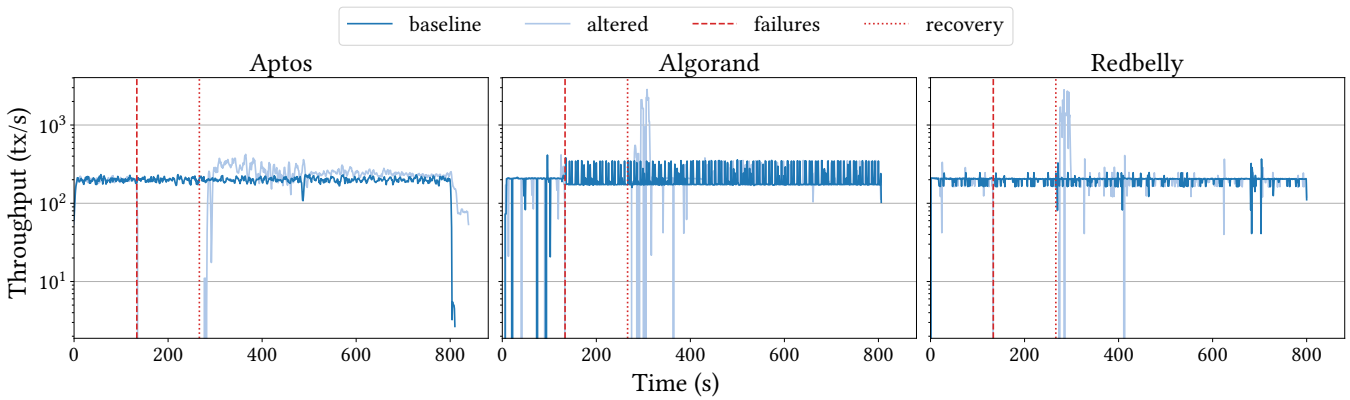

Figure 5. Throughput of the 5 blockchains over time as we transiently stop $f > t$ nodes at time 133 as indicated by the dashed red line and as we recover them at time 233 as indicated by the dotted red line.

Fig. 3b depicts the score for each blockchain with the presence of a transient failures of $f = t + 1$ nodes, and Fig. 5 depicts the throughput over time in the baseline and altered conditions. When the blockchain is unable to recover from the crashes and restore liveness, ∞ symbol is displayed in Fig. 3b instead of the corresponding bar. We also used bars with stripes to indicate that the blockchain showed better response time in the altered environment.

Solana generalized failure. We observed, surprisingly, that the transient failures of some nodes crash all the nodes of Solana. Our in-depth investigations led to conclude that this is related to a bug that prevents a node from synchronizing its state with another.

As explained in Section 2, Solana requires an Epoch Accounts Hash (EAH) to be calculated for the consensus. The panic observed in the validator node stems from an unmet precondition during the EAH calculation process [12], specifically within the `wait_get_epoch_accounts_hash` function. This function is responsible for ensuring that the EAH calculation is either completed or correctly initiated at the expected point in the epoch. The panic occurs because of the ordering of two parallel events: the EAH calculation and the EAH integration. In particular, no EAH calculation was started or in-flight when the bank (described in Section 2) attempted to integrate the EAH into the bank hash at three-quarters (3/4) of the epoch duration, which is a critical part of the consensus process.

By investigating the stack trace, no bank was rooted at the beginning of the epoch, preventing the EAH calculation from starting. As a result, when the bank reached 3/4 of the epoch duration, it was unable to complete the EAH process that had not started. This led to a critical failure because the bank cannot retroactively initiate the EAH calculation, making it impossible to fulfill the required consensus step.

After identifying the cause of the panic, we went to Solana discord channel and found that Solana needed at least 360 slots per epoch [13] while being configured with a smaller amount of slots. The reason is that Solana needs enough time to compute the EAH and to root the relevant bank before the 3/4 mark. Given that rooting can sometimes take up to 150 slots and the freeze-to-rooting process requires at least 32 slots, a buffer is necessary to ensure everything completes correctly. Therefore, the minimum epoch length must be around 360 slots to allow this process to happen without causing a panic. This ensures the EAH computation and rooting are completed in time, preventing errors that could occur if an epoch were too short.

The default epoch duration for the development cluster is 8192 slots. However, with the deployment scripts provided in the Solana repository, genesis block is generated with `enable-warmup-epochs` flag, which shortens the first 8 epochs to progressively smaller slot counts, beginning with 32 slots in epoch 0. These warm-up epochs follow a geometric progression, where the number of slots doubles with each epoch. The epoch size returns to normal (8192 slots) after the warm-up period. The first full-length epoch occurs after 54m24s, and prior to that, each epoch's duration is much shorter. We introduce transient faults to the system at 133 seconds during one of the warm-up epochs, specifically when the number of slots per epoch is still under 360, leading to the described issue.

Avalanche lack of liveness. In Avalanche, we did not experience a generalised outage like in Solana, however, Avalanche's throttling implementation described in Section 4 also prevents the network from reaching consensus.

More specifically, the `InboundMsgThrottler` of Avalanche contains multiple throttler structures, among which the CPU quota-based throttler, `cpuThrottler`, and the message buffer-based throttler, `bufferThrottler`, are of particular interest.

First, the `cpuThrottler` leverages functions to block the CPU consumption of incoming message processing.

When a message arrives, the `systemThrottler.Acquire` function checks if there is enough CPU quota available based on the current CPU usage tracked by `cpuResourceTracker.Usage`. The decision is also influenced by the `targeter.TargetUsage`, which sets a CPU usage threshold. When the threshold is almost reached, `cpuThrottler` blocks further processing of messages, effectively throttling them until CPU resources are freed, preventing the system from exceeding the allocated CPU quota.

Second, the `bufferThrottler` rejects messages depending on the buffer availability with `inboundMsgBufferThrottler.Acquire`. When the system buffers are saturated—typically because the CPU throttling has prevented messages from being processed—the buffer throttler restricts further intake of messages. This backpressure mechanism ensures that incoming messages do not overflow the system when the processing pipeline is already overwhelmed and cannot clear the buffers.

We observed from the logs that the messages were successfully sent and received by the nodes during the experiments, but the throttling prevented them from being processed in a timely manner, resulting in no new blocks being agreed upon. Note that Avalanche is known to require a variant of asynchrony with some form of synchrony for liveness [61]. What this experiment seems to demonstrate is that Avalanche stops working when some messages arrive 2 minutes late.

Algorand and Redbelly recovery. From Fig. 3c and Fig. 3b Algorand and Redbelly display the best behaviors among the studied blockchains when the number of failed nodes exceeds the fault tolerance threshold for a short period. After the crashed nodes are restarted (at 266 seconds), we quickly observe a sharp peak in throughput. This peak corresponds to processing the accumulated backlog of transactions during the downtime. For the rest of the experiment, throughput and latencies match the measurements acquired during the first 133 seconds.

Aptos unrecoverable performance drop. Among three remaining blockchains, Aptos is most significantly impacted by the loss of liveness in the presence of $f = t + 1$ transient failures. While the network starts to create and commit new blocks shortly after restarting the crashed nodes, the transaction throughput does not return to the values observed in the first 133 seconds. Compared to Algorand and Redbelly, the throughput amplitude is significantly lower for Aptos, meaning that it cannot process the pending transactions as fast as Algorand or Redbelly. Furthermore, since we record committed blocks after the end of the experiment, we can observe that the blocks are still being created for a certain period of time. Therefore, we can conclude that Aptos fails to clear the backlog of transactions accumulated during the downtime, and the performance remains degraded for the rest of the experiment, displaying increased transaction latencies.

6 Partition Tolerance

In this section we evaluate the partition tolerance of the 5 blockchains. We conclude that proactive monitoring of link failures can improve the blockchain performance. We also confirm that the blockchains that could not tolerate transient node failures cannot tolerate partition either.

Measuring partition tolerance. To study the blockchain tolerance to network partitions, we run an experiment in three phases similar to the recoverability one of Section 5. For the first 133 seconds, no messages are lost. Between 133 and 266 seconds, a partition is created between $f = t+1$ nodes and the rest of the network so that the messages between the two partitions are dropped. Finally at 266 seconds, the partition stops and from then on all new messages go through.

We used Linux traffic control subsystem facilities to create a transient link failure, or a network partition. First, we used `tc qdisc add dev eth2 root handle 1: prio` to establish a priority queuing discipline at the root of the eth2 interface. Next, we used `tc filter add` to define a set of filters that match IP packets with a destination of IP addresses of the nodes we wanted to disconnect and direct them to the third priority band (`flowid 1:3`). Then, we applied `tc qdisc add` to introduce a `netem qdisc` on `flowid 1:3`, simulating 100% packet loss for the matched traffic. Finally, we used `tc qdisc del dev eth2 root` to remove all traffic control configurations on eth2.

We report the sensitivity scores in Fig. 3c, and show the corresponding throughput over time in Fig. 6.

Solana and Avalanche lack of recovery. Solana and Avalanche fail to recover and restore liveness after the partition, hence we show the infinity symbol, ∞, in addition to the corresponding bar. The issues that make Solana and Avalanche fail to recover after a partition are the same as with the transient node faults previously described in Section 5. In Solana, the EAH calculation, which occurs after the network connectivity is restored, causes all the nodes to crash. In Avalanche, the throttling mechanism prevents the nodes from exchanging transactions and reaching consensus.

As we explained below (Section 6), the other three blockchains, Algorand, Aptos and Redbelly, that recover from transient node failures, show different scores under network partition.

Algorand and Redbelly timeouts. The score of Algorand and Redbelly observed under network partition is higher than their respective score obtained under transient node failures in Section 5. In particular, if we compare Fig. 5 to Fig. 6, we observe that the recovery time of Redbelly increased from 7 to 81 seconds while the recovery time for Algorand increased from 9 to 99 seconds.

After investigating the code of Algorand and Redbelly, we concluded that the recovery time was a function of specific

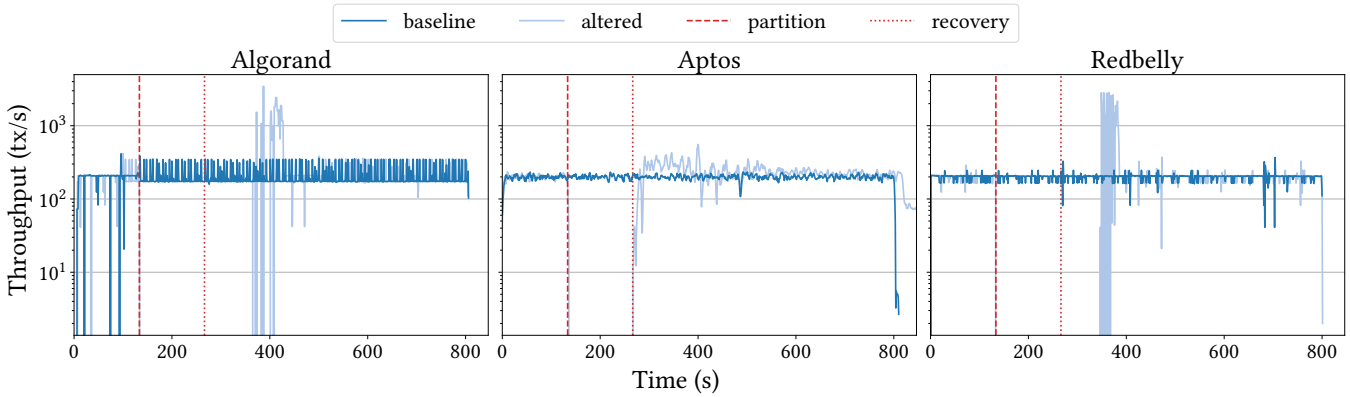

Figure 6. Throughput of the 5 blockchains over time as we transiently partition $f > t$ nodes at time 133 as indicated by the dashed red line and as we stop the partition at time 233 as indicated by the dotted red line.

timeouts. After these timeouts expire, the nodes attempt to reconnect with each other. By looking closer at the code of Redbelly and discussing with its developers, we noticed that an existing MaxIdleTime timeout variable of 30 seconds, could help speedup the recovery further.

Aptos backoff time for quick recovery. By contrast with Algorand and Redbelly, Aptos displays the same sensitivity to nodes being under a transient failure (Fig. 3b) and to a network partition (Fig. 3c). Such a contrast can be explained by different implementation strategies for detecting the network connectivity being restored. Aptos checks peer connectivity every 5 seconds by default. Validator connections are maintained with exponential backoff waiting time with the base value of 2 seconds. A timeout for the connection to open and complete all of the upgrade steps is 30 seconds. Such parameters allow quick connection recovery after the network partition is restored compared to Algorand and Redbelly.

7 Byzantine Node Tolerance

In this section we measure the sensitivity of blockchain systems to a mechanism that copes with some Byzantine behaviors. This behavior consists of having clients send their request to $t_B + 1$ blockchain nodes instead of just 1. We conclude that duplicating the client requests to cope with Byzantine responses can benefit the transaction latency in mempool-based blockchains and that this duplication should be tested in the development process.

Assessing Byzantine node tolerance. The problem of assessing whether a blockchain system is tolerant to Byzantine failures is practically impossible. In fact, by definition the number of possible executions involving Byzantine failures is infinite. In order to assess Byzantine fault tolerance, we thus implement a *secure client* that compares the responses from t_B+1 blockchain nodes before returning the aggregated answer to the application.

To test whether a blockchain performance is impacted by the secure client implementation, we sent the same transaction to 4 different nodes instead of a single node, and reported the transaction as being committed only after all 4 nodes have responded. We used 4 nodes since it is the maximum value for t_B+1 with $n = 10$ across the blockchains under test. The experiment has a single phase during which we use the secure client. As discussed in Section 3, we deployed VMs with 8 vCPUs and 16 GB RAM in order to prevent dropped transactions in Aptos, since a secure client causes extra CPU load on the nodes, as explained later in Section 7.

We depict in Fig. 3d, the sensitivity score for each blockchain with a secure client connected to 4 nodes.

Algorand and Solana remain unchanged. The low sensitivities of Algorand and Solana in Fig. 3d indicate that neither Algorand nor Solana are significantly affected by the redundant requests from the secure client.

In Algorand experiments, since we used a fully-connected network, where each node acts both as relay and participant, we do not observe the expected reduction in transaction latency and throughput improvements with a secure client. Each node maintains a transaction pool, holding transactions in memory before proposing them in a block. Additionally, push and pull gossip methods propagate these transactions across all connected nodes. However, since every node is directly connected and plays dual roles, the network lacks the hierarchical or segmented structure that typically benefits from such optimizations. Consequently, the benefits of reduced latency and enhanced performance are mitigated by the inherent redundancy and uniform connectivity, leading to minimal impact on overall network efficiency.

In Solana, sending a transaction to multiple nodes neither reduce latency nor increase throughput because of its lack of mempool. As discussed in Section 2, Solana uses an approach where transactions are directly forwarded to the

expected leaders based on a pre-determined leader schedule. This process eliminates the need for a mempool, where transactions typically wait to be processed by validators. As a result, broadcasting a transaction to multiple nodes can be seen as redundant since all correct nodes will ultimately route the transaction to the same set of leaders, who will anyway handle it according to the network deterministic leader schedule.

Aptos speculative execution drawback. The root cause of the performance degradation in Aptos on Fig. 3d seems to come from the speculative execution of Block-STM transactions [38]. We observe the following differences in blockchain node logs between the baseline experiment with a single client, and the altered case with the redundant client connected to 4 nodes. In both executions, first, a transaction is added to the mempool, reported by a log message from `Mempool::add_txn` function. Then, a transaction is committed and removed from the mempool, reported by a log message from `Mempool::log_commit_transaction` function. However, in the altered execution, we additionally observe a log message from `SpeculativeEvent::dispatch` 10 milliseconds later with a SEQUENCE_NUMBER_TOO_OLD error, since the transaction is already committed. This means that some transactions are processed at least twice with the redundant client, adding load to the nodes.

Redbelly speedup. As discussed in Section 3, the sensitivity score is always positive as it represents the difference expressed as the absolute value between the area of the baseline environment $\hat{S}_1(b_1)$ and the area of the altered environment $\hat{S}_2(b_2)$. Without this absolute value, the sensitivity could be negative, if the altered environment was offering lower latencies than the baseline environment. This interesting scenario is observed here, because the altered environment benefits Redbelly more than the baseline environment, i.e., $\hat{S}_1(b_1) < \hat{S}_2(b_2)$, as depicted by the crossed bar in Fig. 3d.

The slight latency drop that Redbelly experiences in the altered environment is probably due to the superblock optimization it uses to solve the Set Byzantine Consensus [64]. In particular, as opposed to classic blockchains that decide one of the proposed blocks, Redbelly decides a superblock that combines the valid transactions from all the proposed blocks. As the altered environment sends the same transactions to multiple nodes, it can increase the chances of a transaction being included in the superblock slightly earlier.

Finally, it is important to note that Redbelly already offers its specific recommended library to ensure Byzantine node tolerance [59]. This library called `credence.js` guarantees to a client that the responses it obtains had identical hashes across $t + 1$ replicas. We decided not to use this library and to use our modified client described in Section 7 to obtain a fair comparison with other blockchains.

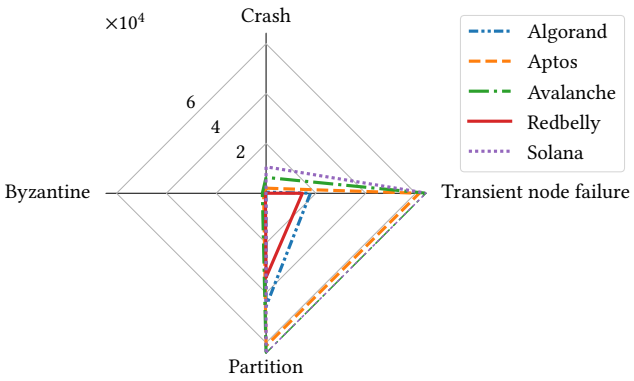

Figure 7. The sensitivity of the tested blockchains to partition, crash, transient failures and to the mechanism that copes with Byzantine nodes.

Avalanche slower sequential execution. Avalanche experiences the largest sensitivity among all blockchains (Fig. 3d). Interestingly, however, Avalanche, just like Redbelly in Section 7 benefits from the redundant requests sent by the client to cope with Byzantine node tolerance, which is indicated in Fig. 3d with $\hat{S}_1(b_1) < \hat{S}_2(b_2)$. This is because this redundancy compensates for transaction reordering and the throttling effects, as we explain below.

First, leader rotation combined with gossip implementation can increase latency. Transactions are executed in issuance order, enforced by a nonce counting previous transactions from the same account owner. For a transaction of an account owner to be executed, all its previous transactions (with lower nonces) must first reach the leader. This can take a long time depending on the leader rotation order and the gossip protocol. Consider a client submitting two transactions with nonces 1 and 2 to the network. The node receiving the transactions may not become a leader for a certain period of time given the leader rotation. Avalanche gossip-based protocol collects transactions from a HashMap in a loop [3], but since HashMap keys do not enforce order [6], lower-nonce transactions may be delayed. But sending a transaction to multiple nodes for Byzantine fault tolerance increases the chances of a transaction being immediately available for the current leader and being included in a block earlier.

Second, the throttling mentioned in Section 4 prevents gossip messages from being immediately processed by the nodes. The message queues being handled by throttling include different internal messages for consensus and transactions. Since clients are processed separately from the blockchain nodes, using the secure client allows to mitigate the negative performance impact caused by throttling and to improve transaction latency.

8 Discussion

We summarize our results and discuss possible limitations.

Synthesizing the results. To get a better understanding of the dependability of the 5 studied blockchains, we report all the sensitivity scores we measured in the previous sections on Figure 7. The scores are displayed in a radar chart with four dimensions representing their sensitivity to crash failures, transient node failures, partitions and to the secure client that copes with Byzantine nodes. For each type of failures, the higher the reported value, the higher the sensitivity to this type of failure. Note that Avalanche and Redbelly benefits from the secure client as explained in Section 7.

The first observation is that generally blockchains are more sensitive to transient failures than permanent failures. First, the blockchains are generally very sensitive to transient failures whether these are link failures, as illustrated by partitions, or node failures, as illustrated by a crash followed by a recovery of individual nodes. Second, they are not as sensitive to the permanent failures. In fact, one can barely see the sensitivity to Byzantine failures and the sensitivity to crash failures is significantly lower than the sensitivity to partitions and transient node failures. After a specific number of transient failures, some blockchains (Solana and Avalanche) could not even recover. Note that because of the fault tolerance threshold of these blockchains, we introduced less permanent failures ($f \leq t_B$) than we have introduced transient failures ($f > t_B$). We can conclude that Solana and Avalanche are likely tuned to only support as many slowly responsive nodes as they can afford permanent failures.

The second observation is that some blockchains (Algorand, Aptos and Redbelly) recover as one would expect but with varying speeds. In particular, Aptos recovers particularly slowly from transient failures while Algorand and Redbelly recover signficantly faster. Finally Redbelly is the fastest blockchain to recover. This can be due to two things. First, the extensive research done around its dependability under Byzantine attacks [32] and flooding attacks [64] required it to cope with adversarial network scenarios impacting the delays of messages. Second, it was shown to have better performance than most of the other blockchains [64] not only due to a reduction in the number of verifications that it needs but also due to its superblock optimization that commits a large batch of accumulated transactions faster.

Limitations of our approach. Our work is a first attempt towards evaluating blockchain dependability. We focused on only five blockchains that are claimed to be Byzantine fault tolerant and there are many more blockchain proposals with the same claim that we could evaluate as well, however, we were unsure of their level of maturity. It is relatively easy to add other blockchains to our framework and we encourage the research community to reuse our results and measure the sensitivity of other blockchains.

The settings of our experiments may seem far from being realistic because blockchain networks are generally of larger scales than our 15-node distributed system and the distance between nodes is generally larger than within a cluster. However, recent results showed that one can get a deep understanding of the performance (both in latency and throughput) of a blockchain at large scale even when deployed in a much smaller environment [50].

Finally, the workload that we use for our experiments only sends native transfer transactions at a constant rate of 200 TPS, which is not representative of realistic fluctuating workloads, request bursts or demanding workloads. The reason for using simple transactions is that some blockchains are unable to support complex smart contract invocations because of the amount of gas they would consume [43]. The reason for using a relatively low sending rate is that some blockchains would lose transactions if the sending rate is too high [43], which typically incurs congestion bottlenecks. Limiting these undesirable effects allowed us to better observe the impact of failures on latencies, which was crucial to measure sensitivity.

9 Conclusion

We presented the first fault tolerance comparison of blockchain systems. To this end, we introduced a new *sensitivity* metric, interesting in its own right, derived from the super-cumulative distribution functions of service response times. This sensitivity metric allowed us to measure *(i)* the resilience, *(ii)* the recoverability, *(iii)* the partition tolerance, and *(iv)* the Byzantine node tolerance of five modern blockchain systems: Algorand, Aptos, Avalanche, Redbelly and Solana. We hope that the developer community will use Stabl to improve the fault tolerance of their blockchain systems. Our future work includes evaluating Byzantine fault tolerance using recommended specialized client libraries, such as `credence.js` for Redbelly. It would also be interesting to measure the sensitivity of blockchains in larger networks, especially for probabilistic consensus protocols that rely on the law of large numbers.

Acknowledgments

We wish to thank the developers of the blockchains who helped us understand the cause of our results: The Avalanche developers confirmed the bug mentioned in Section 4 we also found. The Redbelly developers confirmed that the recovery time could be shortened with the 30-second timeout as discussed in Section 6. The head of the developer relations at Algorand helped us understand the throughput variations of Section 2 before and after the dynamic round time. We thank the anonymous Middleware reviewers and our shepherd, Alysson Bessani, for their insightful comments. This research is supported under Australian Research Future Fellowship and Discovery Project funding scheme (projects number 180100496 and 250101739) entitled "The Red Belly Blockchain" and "Fair Ordering of Decentralised Access to Resources".

References

[1] 2022. The Aptos Blockchain: Safe, Scalable, and Upgradeable Web3 Infrastructure. https://aptosfoundation.org/whitepaper/aptos-whitepaper_en.pdf. Accessed on 31 Aug. 2024.

[2] 2024. AvalancheGo Configs and Flags | Avalanche Docs. https://docs.avax.network/nodes/configure/configs-flags. Accessed on 31 Aug. 2024.

[3] 2024. coreth/core/txpool/legacypool/legacypool.go at master · ava-labs/coreth. https://github.com/ava-labs/coreth/blob/master/core/txpool/legacypool/legacypool.go#L611. Accessed on 31 Aug. 2024.

[4] 2024. go-algorand-sdk/client/v2/algod/rawTransaction.go at develop · algorand/go-algorand-sdk. https://github.com/algorand/go-algorand-sdk/blob/develop/client/v2/algod/rawTransaction.go. Accessed on 31 Aug. 2024.

[5] 2024. go-ethereum/ethclient/ethclient.go at master · ethereum/go-ethereum. https://github.com/ethereum/go-ethereum/blob/master/ethclient/ethclient.go#L624. Accessed on 31 Aug. 2024.

[6] 2024. The Go Programming Language Specification - The Go Programming Language. https://go.dev/ref/spec#For_range. Accessed on 31 Aug. 2024.

[7] 2024. Hyperledger Caliper. https://hyperledger.github.io/caliper/. Accessed on 31 Aug. 2024.

[8] 2024. Pull gossip not working on coreth? · Issue #515 · ava-labs/coreth. https://github.com/ava-labs/coreth/issues/515. Accessed on 31 Aug. 2024.

[9] 2024. Retrying Transactions | Solana. https://solana.com/docs/advanced/retry. Accessed on 31 Aug. 2024.

[10] 2024. RpcClient in solana_client::rpc_client - Rust. https://docs.rs/solana-client/latest/solana_client/rpc_client/struct.RpcClient.html#method.send_and_confirm_transaction. Accessed on 31 Aug. 2024.

[11] 2024. Solana Leader Rotation | Solana Validator. https://docs.solanalabs.com/consensus/leader-rotation. Accessed on 31 Aug. 2024.

[12] 2024. Solana Tech community on Discord. https://discord.com/channels/428295358100013066/838890116386521088/1250587271913013258. Accessed on 31 Aug. 2024.

[13] 2024. Validator fails to restart · Issue #1491 · anza-xyz/agave. https://github.com/anza-xyz/agave/issues/1491. Accessed on 31 Aug. 2024.

[14] [n. d.]. Client Diversity | Ethereum. https://clientdiversity.org Accessed on 27 Mar. 2025.

[15] [n. d.]. Considerations | Avalanche Builder Hub. https://builders-na4bsvdak-ava-labs.vercel.app/docs/avalanche-l1s/upgrade/considerations Accessed on 27 Mar. 2025.

[16] Mohammad Javad Amiri, Chenyuan Wu, Divyakant Agrawal, Amr El Abbadi, Boon Thau Loo, and Mohammad Sadoghi. 2024. The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocols Analysis, Implementation, and Experimentation. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 371–400. https://www.usenix.org/conference/nsdi24/presentation/amiri

[17] Ignacio Amores-Sesar, Christian Cachin, and Philipp Schneider. 2024. An Analysis of Avalanche Consensus. In *Proceedings of the Structural Information and Communication Complexity: 31st International Colloquium (SIROCCO)*. 27–44.

[18] Tom Anderson. 1981. *Fault Tolerance: Principles and Practice*. Prentice Hall, Englewood Cliffs.

[19] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quema. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, Philadelphia, PA, USA, 297–306. https://doi.org/10.1109/ICDCS.2013.53

[20] Avinash Dixit. 2007. Stochastic Dominance | Economics of Uncertainty. https://www.princeton.edu/~dixitak/Teaching/EconomicsOfUncertainty/Slides&Notes/Notes04.pdf. Accessed on 31 Aug. 2024.

[21] Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, Zekun Li, Avery Ching, and Dahlia Malkhi. 2022. Twins: BFT Systems Made Robust. (2022), 29 pages, 1042071 bytes. https://doi.org/10.4230/LIPICS.OPODIS.2021.7

[22] BBC. 2024. CrowdStrike IT outage affected 8.5 million Windows devices, Microsoft says. https://www.bbc.com/news/articles/cpe3zgznwjno. Accessed on 31 Aug. 2024.

[23] Christian Berger, Hans P. Reiser, João Sousa, and Alysson Bessani. 2022. AWARE: Adaptive Wide-Area Replication for Fast and Resilient Byzantine Consensus. *IEEE Transactions on Dependable and Secure Computing* 19, 3 (2022), 1605–1620. https://doi.org/10.1109/TDSC.2020.3030605

[24] N. Bertrand, V. Gramoli, M. Lazić, I. Konnov, P. Tholoniat, and J. Widder. 2022. Holistic Verification of Blockchain Consensus. In *36th International Symposium on Distributed Computing (DISC)*.

[25] Nathan Bronson, Abutalib Aghayev, Aleksey Charapko, and Timothy Zhu. 2021. Metastable failures in distributed systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. ACM, Ann Arbor Michigan, 221–227. https://doi.org/10.1145/3458336.3465286

[26] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild (Keynote Talk). In *31st International Symposium on Distributed Computing, DISC (LIPIcs, Vol. 91)*, Andréa W. Richa (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1:1–1:16.

[27] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX Association, USA, 173–186. event-place: New Orleans, Louisiana, USA.

[28] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail?: A Study of Hyperledger Fabric. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, Virtual Event China, 221–234. https://doi.org/10.1145/3448016.3452823

[29] Giorgio Ciotti. 2024. Algorand's Latest Upgrade: Dynamic Round Times & AVM v10 | Algorand Developer Portal. https://developer.algorand.org/articles/algorands-latest-upgrade-dynamic-round-times-avm-v10/. Accessed on 31 Aug. 2024.

[30] Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 09)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/nsdi-09/making-byzantine-fault-tolerant-systems-tolerate-byzantine-faults

[31] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. 2018. DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains. In *17th IEEE International Symposium on Network Computing and Applications NCA*. IEEE, 1–8.

[32] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red Belly: a Secure, Fair and Scalable Open Blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*.

[33] Aptos developers. 2025. Validator Nodes Overview. https://aptos.dev/en/network/blockchain/validator-nodes Accessed on 19 May 2025.

[34] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, Chicago Illinois USA, 1085–1100. https://doi.org/10.1145/3035918.3064033

[35] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (April 1988), 288–323.

[36] Eddie Mitchell. 2024. Solana Outage: Full List Of SOL Network Blockchain Mainnet Failures. https://cryptomaniaks.com/crypto-news/solana-outage-list-failures-sol-blockchain-mainnet. Accessed on 31 Aug. 2024.

[37] Hal Finney. 2011. Finney's attack. https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384. Accessed on 31 Aug. 2024.

[38] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. 2023. Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. ACM, Montreal QC Canada, 232–244. https://doi.org/10.1145/3572848.3577524

[39] Frank Christian Geyer, Hans-Arno Jacobsen, Ruben Mayer, and Peter Mandl. 2023. An End-to-End Performance Comparison of Seven Permissioned Blockchain Systems. In *Proceedings of the 24th International Middleware Conference on ZZZ*. ACM, Bologna Italy, 71–84. https://doi.org/10.1145/3590140.3629106

[40] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proc. 26th Symp. Operating Syst. Principles*. 51–68.

[41] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: A Scalable and Decentralized Trust Infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Portland, OR, USA, 568–580. https://doi.org/10.1109/DSN.2019.00063

[42] Vincent Gramoli. 2022. *Blockchain Scalability and its Foundations in Distributed Systems*. Springer. https://doi.org/10.1007/978-3-031-12578-2

[43] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. 2023. Diablo: A Benchmark Suite for Blockchains. In *Proceedings of the Eighteenth European Conference on Computer Systems*. ACM, Rome Italy, 540–556. https://doi.org/10.1145/3552326.3567482

[44] Divya Gupta, Lucas Perronne, and Sara Bouchenak. 2016. BFT-Bench: Towards a Practical Evaluation of Robustness and Effectiveness of BFT Protocols. In *Distributed Applications and Interoperable Systems*, Márk Jelasity and Evangelia Kalyvianaki (Eds.). Vol. 9687. Springer International Publishing, Cham, 115–128. https://doi.org/10.1007/978-3-319-39577-7_10 Series Title: Lecture Notes in Computer Science.

[45] Lexiang Huang, Matthew Magnusson, Abishek Bangalore Muralikrishna, Salman Estyak, Rebecca Isaacs, Abutalib Aghayev, Timothy Zhu, and Aleksey Charapko. 2022. Metastable Failures in the Wild. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 73–90. https://www.usenix.org/conference/osdi22/presentation/huang-lexiang

[46] David Hyland, João Sousa, Gauthier Voron, Alysson Bessani, and Vincent Gramoli. 2024. Ten Myths About Blockchain Consensus. In *Blockchains*, Sushmita Ruj, Salil S. Kanhere, and Mauro Conti (Eds.). Vol. 105. Springer International Publishing, Cham, 3–24. https://doi.org/10.1007/978-3-031-32146-7_1 Series Title: Advances in Information Security.

[47] Ghassan Karame, Androulaki Elli, and Capkun Srdjan. 2012. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. *IACR Cryptology ePrint Archive 2021* (2012).

[48] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401.

[49] Jean-Claude Laprie. 1984. Dependability Evaluation of Software Systems in Operation. *IEEE Trans. Software Eng.* 10, 6 (1984), 701–714.

[50] Andrei Lebedev and Vincent Gramoli. 2023. On the Relevance of Blockchain Evaluations on Bare Metal. In *7th Symposium on Distributed Ledger Technologies (SDLT)*.

[51] Hyojeong Lee, Jeff Seibert, Endadul Hoque, Charles Killian, and Cristina Nita-Rotaru. 2014. Turret: A Platform for Automated Attack Finding in Unmodified Distributed System Implementations. In *2014 IEEE 34th International Conference on Distributed Computing Systems*. IEEE, Madrid, Spain, 660–669. https://doi.org/10.1109/ICDCS.2014.73

[52] Fuchen Ma, Yuanliang Chen, Meng Ren, Yuanhang Zhou, Yu Jiang, Ting Chen, Huizhong Li, and Jiaguang Sun. 2023. LOKI: State-Aware Fuzzing Framework for the Implementation of Blockchain Consensus Protocols. In *Proceedings 2023 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2023.24078

[53] Fuchen Ma, Yuanliang Chen, Yuanhang Zhou, Jingxuan Sun, Zhuo Su, Yu Jiang, Jiaguang Sun, and Huizhong Li. 2023. Phoenix: Detect and Locate Resilience Issues in Blockchain via Context-Sensitive Chaos. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1182–1196. https://doi.org/10.1145/3576915.3623071

[54] Liyuan Ma, Xiulong Liu, Yuhan Li, Chenyu Zhang, Gaowei Shi, and Keqiu Li. 2024. GFBE: A Generalized and Fine-Grained Blockchain Evaluation Framework. *IEEE Trans. Comput.* 73, 3 (March 2024), 942–955. https://doi.org/10.1109/TC.2024.3349654 Conference Name: IEEE Transactions on Computers.

[55] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf. Accessed on 31 Aug. 2024.

[56] Bulat Nasrulin, Martijn De Vos, Georgy Ishmaev, and Johan Pouwelse. 2022. Gromit: Benchmarking the Performance and Scalability of Blockchain Systems. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, Newark, CA, USA, 56–63. https://doi.org/10.1109/DAPPS55202.2022.00015

[57] Christopher Natoli and Vincent Gramoli. 2017. The Balance Attack or Why Forkable Blockchains are Ill-Suited for Consortium. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN*. IEEE Computer Society, 579–590.

[58] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Jorge Esteves Veríssimo. 2019. *Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure*. Technical Report 1908.08316. arXiv. http://arxiv.org/abs/1908.08316.

[59] Redbelly Network. 2022. Redbelly Blockchain: a Combination of Recent Advances. *Bull. EATCS* 137 (2022). http://bulletin.eatcs.org/index.php/beatcs/article/view/703

[60] A. Ranchal-Pedrosa and V. Gramoli. 2024. ZLB: A Blockchain to Tolerate Colluding Majorities. In *54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[61] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2020. Scalable and Probabilistic Leaderless BFT Consensus through Metastability. http://arxiv.org/abs/1906.08936.

[62] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT protocols under fire. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, USA, 189–204. event-place: San Francisco, California.

[63] Jakub Sliwinski, Quentin Kniep, Roger Wattenhofer, and Fabian Schaich. 2024. Halting the Solana Blockchain with Epsilon Stake. In *Proceedings of the 25th International Conference on Distributed Computing and Networking (ICDCN)*. 45–54.

[64] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. 2023. Smart Redbelly Blockchain: Reducing Congestion for Web3. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, St. Petersburg, FL, USA, 940–950. https://doi.org/10.1109/IPDPS54959.2023.00098

[65] The Diem Team. 2021. DiemBFT v4: State Machine Replication in the Diem Blockchain. https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf. Accessed on 31 Aug. 2024.

[66] vector76. [n. d.]. The vector76 attack. https://bitcointalk.org/index.php?topic=36788.msg463391. Accessed on 31 Aug. 2024.

[67] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2009. Spin One's Wheels? Byzantine Fault Tolerance

with a Spinning Primary. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, Niagara Falls, New York, USA, 135–144. https://doi.org/10.1109/SRDS.2009.36

[68] Gauthier Voron and Vincent Gramoli. 2020. *Dispel: Byzantine SMR with Distributed Pipelining*. Technical Report 1912.10367v2. arXiv. https://arxiv.org/pdf/1912.10367.

[69] Levin N. Winter, Florena Buse, Daan De Graaf, Klaus Von Gleissenthall, and Burcu Kulahcioglu Ozkan. 2023. Randomized Testing of Byzantine Fault Tolerant Algorithms. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (April 2023), 757–788. https://doi.org/10.1145/3586053

[70] Anatoly Yakovenko. 2021. Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf. Accessed on 31 Aug. 2024.

[71] Youngseok Yang, Taesoo Kim, and Byung-Gon Chun. 2021. Finding Consensus Bugs in Ethereum via Multi-transaction Differential Fuzzing. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 349–365. https://www.usenix.org/conference/osdi21/presentation/yang

[72] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*.

[73] Long Zhang, Javier Ron, Benoit Baudry, and Martin Monperrus. 2023. Chaos Engineering of Ethereum Blockchain Clients. *Distrib. Ledger Technol.* 2, 3, Article 22 (Sept. 2023), 18 pages. https://doi.org/10.1145/3611649