Dissecting BFT Consensus:
In Trusted Components we Trust!

Suyash Gupta[†], Sajjad Rahnama, Shubham Pandey, Natacha Crooks[†], Mohammad Sadoghi
Exploratory Systems Lab, University of California, Davis
[†]SkyLab, University of California, Berkeley

Abstract

The growing interest in reliable multi-party applications has fostered widespread adoption of Byzantine Fault-Tolerant (BFT) consensus protocols. Existing BFT protocols need $\mathbf{f}$ more replicas than Paxos-style protocols to prevent equivocation attacks. TRUST-BFT protocols seek to minimize this cost by making use of trusted components at replicas.

This paper makes two contributions. First, we analyze the design of existing TRUST-BFT protocols and uncover three fundamental limitations that preclude most practical deployments. Some of these limitations are fundamental, while others are linked to the state of trusted components today. Second, we introduce a novel suite of consensus protocols, FLEXITRUST, that attempts to sidestep these issues. We show that our FLEXITRUST protocols achieve up to 185% more throughput than their TRUST-BFT counterparts.

CCS Concepts: • **Security and privacy → Systems security**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**.

Keywords: Byzantine fault-tolerance, consensus, SGX, responsiveness, parallelism, permissioned blockchain

1 Introduction

Byzantine Fault-Tolerant (BFT) protocols allow multiple parties to perform shared computations and reliably store data without fully trusting each other; the system will remain correct even if a subset of participants behave maliciously [15, 36, 38, 41, 48, 87]. These protocols aim to ensure that all the replicas reach *consensus* on the order of incoming client

requests. These protocols are increasingly popular today, with applications ranging from safeguarding replicated and sharded databases [40, 72, 81], edge applications [39, 62], and blockchain applications [7, 60, 85].

BFT protocols tolerate a subset of participants behaving arbitrarily: a malicious actor can delay, reorder or drop messages (*omission faults*); it can also send conflicting information to participants (*equivocation*) [36]. As a result, BFT consensus protocols are costly: maintaining correctness requires a minimum of $3\mathbf{f}+1$ participants, where $\mathbf{f}$ participants can be malicious. This is in contrast to crash-fault tolerant protocols (CFT), where participants can fail only by crashing, therefore requiring only $2\mathbf{f}+1$ participants for correctness [49, 66].

To minimise this additional cost, some existing BFT protocols leverage trusted components to curb the degree to which participants can behave arbitrarily [26, 27, 50, 79]. A trusted component provably performs a specific computation: incrementing a counter [51, 83], appending to a log [21], or more advanced options like executing a complex algorithm [45, 52, 75]. While there exists a large number of BFT protocols that leverage these trusted components [21, 51, 86] (we refer to these protocol as TRUST-BFT protocols for simplicity), they all proceed in a similar fashion: they *force* each replica to commit to a single order for each request by having the trusted component sign each message it sends. In turn, each trusted component either: (1) records the chosen order for a client request in an append-only log, or (2) binds this order for the request with the value of a monotonically increasing counter. Committing to an order in this way allows these protocols to remain safe with $2\mathbf{f}+1$ replicas only, bringing them in line with their CFT counterparts.

While reducing replication cost is a significant benefit, this paper argues that current TRUST-BFT protocols place *too much trust in trusted components*. Our analysis uncovers three fundamental issues with existing TRUST-BFT implementations: (i) limited responsiveness for clients, (ii) safety concerns associated with trusted components, and (iii) inability to perform multiple consensuses in parallel.

Responsiveness We observe that malicious replicas can successfully prevent a client from receiving a response for its transactions. While the transaction will still commit (consensus liveness), the system will appear to clients as stalled and thus appear non-responsive to clients. TRUST-BFT protocols allow a reduced quorum size of $\mathbf{f}+1$ to commit a request. As $\mathbf{f}$ of those may be malicious, only one honest replica is

guaranteed to execute the operation. This is insufficient to guarantee that a client will receive the necessary $f + 1$ matching responses post operation execution to validate that the response is indeed valid.

Loss of Safety under Rollback. Existing TRUST-BFT protocols consider an idealised model of trusted computation. They assume that the trusted components cannot be compromised and that their data remains persistent in the presence of a malicious host. This assumption does not yet align with current hardware functionality. A large number of these protocols employ Intel SGX enclaves for trusted computing [13, 28, 78]. Unfortunately, SGX-based designs have been shown to suffer from rollback attacks [43, 58, 84], and the solutions to mitigate these attacks lack practical deployments [30]. Hardware enclaves that do provably defend against rollback attacks, such as persistent counters [26] and TPMs [33], have prohibitively high latencies (tens of milliseconds) [51, 56, 68].

Sequential Consensus. Existing TRUST-BFT protocols are inherently sequential as they require each outgoing message to be ordered and attested by trusted components. While recent work mitigates this issue by pipelining consensus phases [28] or running multiple independent consensus invocations [13], their performance remains fundamentally limited by the RTT of each protocol phase. In fact, despite their lower replication factor, we observe that TRUST-BFT protocols achieve lower throughput than traditional parallel BFT protocols [15, 38, 65, 81].

This paper argues that TRUST-BFT protocols have targeted the wrong metric: while reducing the replication factor to $2f + 1$ may seem appealing from a resource efficiency or management overhead standpoint, it, paradoxically, comes at a significant performance cost as the lack of parallelism and heavy reliance on trusted hardware hurts throughput significantly. We propose a novel suite of consensus algorithms (**Flexi**ble **Trust**ed BFT (FLEXITRUST)), which address the aforementioned limitations. These protocols are always responsive and achieve high throughput as they (1) make minimal use of trusted components (once per client operation and at the primary replica only), and (2) support parallel consensus invocations. Both these properties are made possible by the ability to use large quorums (of size $2f + 1$) when using $3f + 1$ replicas. Our techniques can be used to convert any TRUST-BFT protocol into a FLEXITRUST protocol. We provide as examples two such conversions: FLEXI-BFT and FLEXI-ZZ, two protocols based on PBFT [15]/ MINBFT [83] and ZYZZYVA [48]/ MINZZ [83], respectively. FLEXI-BFT follows a similar structure to PBFT, but requires one less communication phase. FLEXI-ZZ is, we believe, of independent interest: the protocol achieves consensus in a single linear phase without using expensive cryptographic constructs such as threshold signatures. Crucially, unlike the ZYZZYVA and MINZZ protocols, FLEXI-ZZ continues to commit in a single-round even when a single participant misbehaves, thus maintaining high-throughput [23, 24, 38]. Further, FLEXI-ZZ presents

a simplified view-change protocol than other single phase consensus protocols, which prior works have illustrated is notoriously complex and error-prone [2, 31].

We evaluate our FLEXITRUST variants against *five* protocols: *three* TRUST-BFT and *two* BFT systems. For fair evaluation, we implement these protocols on the open-source RESILIENTDB fabric [37, 38, 65, 71]. Our evaluation on a real-world setup of 97 replicas and up to 80 k clients shows that our FLEXITRUST protocols achieve up to 185% and 100% more throughput than their TRUST-BFT and BFT counterparts, respectively. Further, we show that our FLEXITRUST protocols continue outperforming these protocols even when replicas are distributed across *six* locations in four continents. The aforementioned results use SGX and are thus subject to rollback attacks. When simulated with actual persistent counters, we find that FLEXITRUST's minimal use of hardware enclaves mitigates the cost of the required mechanisms to prevent rollbacks. In summary, we make the following *four* contributions:

- We identify a responsiveness issue in existing TRUST-BFT protocols when an honest replica faces temporary message delays.
- We highlight that rollback attacks on trusted components limit practical deployments.
- We observe that existing TRUST-BFT protocols are inherently sequential. The lack of support for parallel consensus invocations artificially limits throughput compared to traditional BFT prtoocols.
- We present FLEXITRUST protocols: a novel suite of protocols that support concurrent consensus invocations and require minimal access to the trusted components.

2 System model and notations

We adopt the standard communication and failure model adopted by most BFT protocols [15, 21, 35, 38, 65, 81], including all existing TRUST-BFT protocols [13, 21, 28, 83, 86].

We consider a replicated service S consisting of set of n replicas R of which at most f can behave arbitrarily. The remaining $n - f$ are honest: they will follow the protocol and remain live. We additionally assume the existence of a finite set of clients C of which arbitrarily many can be malicious.

Similarly, we inherit standard communication assumptions: we assume the existence of *authenticated channels* (malicious replicas can impersonate each other but no replica can impersonate an honest replica) and standard cryptographic primitives such as MACs and digital signatures (DS). We denote a message m signed by a replica R using DS as $\langle m \rangle_R$. We employ a *collision-resistant* hash function $\mathsf{Hash}(\cdot)$ to map an arbitrary value v to a constant-sized digest $\mathsf{Hash}(v)$. We adopt the same partial synchrony model adopted in most BFT systems: safety is guaranteed in an asynchronous environment where messages can get lost, delayed, or duplicated. Liveness, however, is only guaranteed during periods

of synchrony [15, 21, 38, 81, 87]. Each replica only accepts a message if it is **well-formed**. A well-formed message has a valid signature and passes all the protocol checks.

We follow Schneider's seminal work [77] to distinguish between consensus and Replicated State Machine (Rsm). A consensus protocol aims to order the operations among the participants, whereas the correctness of an Rsm is not just defined by the agreement but also that the response to a client's transaction T is the result of applying each transaction that precedes T in order. This description allows us to define the following guarantees:

Consensus Safety. If two honest replicas r1 and r2 order a transaction T at sequence numbers k and k', then $k = k'$.

Consensus Liveness. If a honest replica commits T, then all honest replicas eventually commit T.

Rsm Safety. Given a consensus order O of transactions and a transacton T, the output of T is consistent with applying T after applying all the transactions in O according to the semantic characterisation of the state machine.

Rsm Liveness. If a client sends a transaction T, then it will eventually receive a response for T.

Most works assume that RSM safety and liveness implicitly follow from consensus safety/liveness. We separate these properties to emphasise the need for $f + 1$ matching reads; traditional consensus definitions often ignore the response part of the request. We additionally assume that our replicated service $\mathcal{S}$ includes a set of trusted components. These trusted components offer the following abstraction:

Definition 1. *A trusted component τ is a cryptographically secure entity, which has a negligible probability of being compromised by malicious adversaries. τ provides access to a function* foo() *and, when called, always computes* foo().

Existing TRUST-BFT protocols assume that each replica $r \in \mathcal{R}$ has access to a *co-located* trusted component. We use the notation τ_r to denote the trusted component at the "host" replica r. As τ_r computes foo() and τ_r cannot be compromised by the host r, existing TRUST-BFT protocols claim integrity in the computation of foo().

3 Primer on BFT Consensus

To highlight the limitations of TRUST-BFT protocols, we first explain BFT consensus. To this effect, we summarize the structure of PBFT-like systems, which represent most modern BFT protocols today [31, 35, 37, 38, 48, 87]. For simplicity of exposition, we focus specifically on PBFT [15].

PBFT adopts the system and communication model previously described and guarantees safety and liveness for $n = 3f + 1$ replicas where at most f can be malicious. The protocol follows a primary-backup model: one replica is designated as the primary (or leader), while the remaining replicas act as backups. At a high-level, all BFT protocols consist of two logical phases: an agreement phase where replicas agree to commit a specific operation, and a second durability phase where this decision will persist even when the leader is malicious and replaced as part of a *view-change* (view-changes are the process through which leaders are replaced). For ease of understanding, we focus on explaining the simplest failure-free run of PBFT.

Client Library. The input to any Rsm is a client request. A client c invokes the *client side library* when it wants the Rsm to process its transaction T. To do so, c issues a signed message $\langle T \rangle_c$ to the library, which forwards the message to the Rsm's primary replica p. The client library at c waits for identical responses from $f + 1$ replicas before returning the result $\mathbf{r}$ to c. Waiting for $f + 1$ matching responses ensures execution correctness as at least one honest replica vouches to that result.

1. **Pre-prepare.** When the primary replica p receives a well-formed client request m, it assigns the corresponding transaction a new sequence number k and sends a PrePrepare message to all backups.

2. **Prepare.** When a replica $r \in \mathcal{R}$ receives, for the first time, a well-formed PrePrepare message from p for a given sequence number k, r broadcasts to all other replicas a Prepare message agreeing to support this (sequence number, transaction) pairing.

3. **Commit.** When a replica r receives identical Prepare messages from $2f + 1$ replicas, it marks the request m as *prepared* and broadcasts a Commit message to all replicas. When a request is prepared, a replica has the guarantee that no conflicting request m' for the same sequence number will ever be prepared by this leader.

4. **Execute.** Upon receiving $2f + 1$ matching Commit messages, r marks the request as *committed* and executable. Each replica r executes every request in sequence number order: the node waits until request for slot $k - 1$ has successfully executed before executing the transaction at slot k. Finally, r returns the result $\mathbf{r}$ of the operation to the client c.

4 Trusted BFT Consensus

BFT protocols successfully implement consensus, but at a cost: they require a higher replication factor ($3f + 1$) than their CFT counterparts and, in turn, must process significantly more messages (all of which must be authenticated and whose signatures/MACs must be verified). Equivocation is the primary culprit: a byzantine replica can tell one group of honest replicas that it plans to order a transaction T before transaction T', and tells another group of replicas that it will order T' before T.

In a CFT system, it is sufficient for quorums to intersect in a single replica to achieve *agreement*. In BFT systems, quorums must instead intersect in one *honest* replica (or, in other words, must intersect in at least $f + 1$ replicas).

To mitigate this increased cost, TRUST-BFT protocols make use of trusted components (such as Intel SGX, AWS Nitro) at each node [21, 22, 59]. Trusted components *cannot*, by assumption, be compromised by malicious actors. They can thus be used to prevent replicas from equivocating. *In theory*, this should be sufficient to safely convert any BFT protocol into a CFT system [22]. In practice, however, as we highlight in this work, this process is less than straightforward.

4.1 Trusted Component Implementations

The easiest option is to run the full BFT consensus inside the trusted component [9, 32, 67, 69, 76, 82]. This approach violates the principle of least privilege [54, 74].

Instead, existing protocols choose to put the smallest amount of computation inside the trusted component, which benefits by allowing custom hardware implementations that are more secure [21, 51]. There are two primary approaches: append-only logs, and monotonically increasing counters. Careful use of these primitives allows prior work to reduce the replication factor from $3f + 1$ to $2f + 1$.

Trusted Logs. TRUST-BFT protocols like PBFT-EA [21] and HOTSTUFF-M [86] maintain, in each trusted component T_R, a set of append-only logs. Each log has an identifier (say q) and a set of slots. We refer to each slot using an identifier k. Each trusted component T_R offers the following API.

1. **APPEND**(q, k_{new}, x) – Assume the last append to q-th log of trusted component T_R was at slot k, then
    *   If $k_{new} = \perp$, no slot location is specified, T_R sets k to $k + 1$ and appends the value x to slot k.
    *   If $k_{new} > k$, T_R appends x to slot k_{new} and updates k to k_{new}. The slots in between can no longer be used.
2. **LOOKUP**(q, k) – If there exists a value x at slot k in log q of T_R, it returns an attestation $\langle \textbf{ATTEST}(q, k, x) \rangle_{T_R}$.

The APPEND function ensures that no two requests are ever logged at the same slot; the LOOKUP function confirms this fact with a digitally-signed assertion $\langle \text{ATTEST}(q, k, x) \rangle_{T_R}$ where k is the log position and x is the stored message.

Trusted Counters. A separate line of work further restricts the scope of the trusted components. Rather than storing an attested log of messages, it simply stores a set of monotonically increasing counters [46, 83]. These counters do not provide a LOOKUP function as they do not store messages. Instead, the APPEND(q, k_{new}, x) function, at the time of invocation, returns the attestation proof $\langle \text{ATTEST}(q, k, x) \rangle_{T_R}$, which states that the q-th counter updates its current value k to k_{new} and bounds the updated value k to message x.

These two designs are not mutually exclusive; protocols like TRINC [51], HYBSTER [13] and DAMYSUS [28] require their trusted components to support both logs and counters, where logs record the last few client requests.

4.2 TRUST-BFT protocol

Existing TRUST-BFT protocols expect that in a system of $n = 2f + 1$ replicas at most f replicas are byzantine. Most

TRUST-BFT [13, 21, 46, 51, 83, 86] follow a similar design, based on the PBFT-EA [21] protocol. PBFT-EA is derived from PBFT and makes use of a set of trusted logs.

PBFT-EA protocol steps. In the PBFT-EA protocol, T_R stores *five* distinct sets of logs (note that only two logs are used in the failure-free case). When the primary P receives a client request m, it calls the APPEND(q, k, m) function to assign m a sequence number k. T_P logs m in the q-th *prepare log* and returns an attestation $\langle \text{ATTEST}(q, k, m) \rangle_{T_P}$, which P forwards to all replicas along with the PREPREPARE message. The existence of a trusted log precludes a primary from equivocating and sending two conflicting messages m and m' for the same sequence number k. When a replica R receives a well-formed PREPREPARE message for slot k, it creates a PREPARE message m', and calls the APPEND function on its T_R to log m'. As a result, T_R logs m' in its q-th *prepare log* and returns an attestation $\langle \text{ATTEST}(q, k, m') \rangle_{T_R}$, which R forwards along with the PREPARE message to all replicas.

Once R receives $f + 1$ identical PREPARE messages from distinct replicas (including itself), it declares the transaction prepared. Following this, R repeats the process by creating a COMMIT message (say m'') and asking its T_R to log this message at slot k in its q-th *commit log*. R broadcasts the signed attestation $\langle \text{ATTEST}(q, k, m'') \rangle_{T_R}$ along with the COMMIT message. Once R receives $f + 1$ matching COMMIT messages for m at slot k, it marks the operation committed. R executes m once it has executed the request at slot $k - 1$ and sends the result of execution (**r**). The client library returns **r** to the client when it receives $f + 1$ identical responses.

MINBFT and MINZZ protocols. MINBFT [83] improves over PBFT-EA by observing that the use of trusted components makes the COMMIT phase redundant. MINBFT allows a replica to mark transactions as committed once it receives $f+1$ identical PREPARE messages. MINZZ [83] makes a similar observation to improve the ZYZZYVA [48] protocol: it allows replicas to speculatively execute an operation once they receive a PREPREPARE message from the primary. However, unlike ZYZZYVA where the client needs identical responses from $n = 3f + 1$ replicas to mark its transaction as complete, in MINZZ the client requires $n = 2f + 1$ responses. Both MINBFT and MINZZ make use of trusted counters.

Checkpoints. Like BFT protocols, TRUST-BFT protocols also periodically share checkpoints. These checkpoints reflect the state of a replica R's trusted component T_R and enable log truncation. During the checkpoint phase, R sends CHECKPOINT messages that include all the requests committed since the last checkpoint. If T_R employs trusted logs, then it also provides attestations for each logged request. If instead T_R employs only trusted counters, T_R provides an attestation on the current value of its counters. Each replica R marks its checkpoint as stable if it receives CHECKPOINT messages from $f + 1$ other replicas possibly including itself.

View Change. The existence of trusted logs/counters precludes the primary from equivocating, but it can still deny

service and maliciously delay messages. The *view-change* mechanism enables replicas to replace a faulty primary; a view refers to the duration for which a specific replica was leader. All TRUST-BFT protocols provide such a mechanism [21, 46]: a view change is triggered when $f + 1$ replicas send a VIEWCHANGE message. Requiring at least $f + 1$ messages ensures that malicious replicas cannot alone attempt to replace leaders. Similar to the common-case protocol, $f + 1$ replicas must participate in every view-change quorum.

Unfortunately, we identify several limitations and performance bottlenecks in all of the aforementioned protocols (Figure 1), which we describe next.

5 Restricted Responsiveness

We first observe that current TRUST-BFT protocols lack guaranteed client responsiveness: temporary delaying the messages of a single honest replica is sufficient to prevent a client from receiving a result for its transaction until the next checkpoint. For any production-scale system, RSM liveness (or client responsiveness, the point at which the client receives a response to its operation) is as important a metric as consensus liveness (the operation commitment at the replica). Specifically, TRUST-BFT protocols *as currently implemented* guarantee that a quorum of $f + 1$ replicas will commit a request (consensus liveness), but do not guarantee that sufficiently many honest replicas will actually *execute the request*. These protocols cannot guarantee that the clients will receive $f + 1$ identical responses from distinct replicas (RSM liveness), which is the necessary threshold to validate execution results. The ease with which such a RSM liveness (or client responsiveness) issue can be triggered highlights the brittleness of current TRUST-BFT approaches. We highlight this issue using MINBFT as an example, but, to the best of our knowledge, the same problem arises with all existing TRUST-BFT protocols.

Claim 1. *In a replicated system $\mathcal{S}$ of replicas $\mathcal{R}$ and clients C, where $|\mathcal{R}| = n = 2f + 1$, there exist an execution in which RSM liveness is not guaranteed*

Proof. Assume a run of the MINBFT protocol (Figure 2). We know that f of the replicas in $\mathcal{R}$ are malicious. We represent them with set $\mathcal{F}$. The remaining $f + 1$ replicas are honest. Let us distribute them into two groups: D and R, such that $|D| = f$ replicas and R be the remaining replica.

Assume that the primary P (view v) is malicious (P $\in \mathcal{F}$) and all replicas in $\mathcal{F}$ intentionally fail to send replicas in D any messages. As is possible in a partially synchronous system, we further assume that the PREPARE messages from the replica R to those in D are temporarily delayed by an amount greater than the view change timeout.

P sends a PREPREPARE message for a client c's transaction T to all replicas in $\mathcal{F}$ and to R. These $f + 1$ replicas are able to prepare T. All messages from replica R to those in D are temporarily delayed by an amount greater than the client

and view change timeouts. During this time, R is the only honest replica to receive the transaction and reply to the client c as all the replicas in $\mathcal{F}$ fail to respond. Unfortunately, the client needs $f + 1$ responses to validate the correctness of the executed operation, and thus cannot make progress.

After its timeout expires, the client will inform to all the replicas that it has not received sufficient responses for its transaction T. As replica R has successfully committed and executed T, it will simply reply back to c. The f replicas in D will wait for the leader to initiate consensus on T. Having not heard from the leader about T, the f replicas in D will vote to trigger a view-change and will switch to view $v + 1$. Replica R will not agree to switch to view $v + 1$ and cannot trust the f replicas in D nor the (potentially malicious) client. Replicas in $\mathcal{F}$ will also not trigger a view-change.

Unfortunately, a view-change requires at least $f + 1$ votes to proceed; otherwise, malicious replicas could stall system progress by constantly triggering spurious view-changes. The f replicas in D will thus not successfully trigger a view-change. To make matters worse, in all BFT protocols, once a replica votes to enter a new view ($v + 1$), it must, for safety, discard any message it receives for view v. As such, even when the replicas in D do in fact receive the delayed messages for T, they can no longer process them! In summary, in this example, the system can no longer successfully execute operations: the client will never receive enough matching responses and no view-change can be triggered to address this issue. □

This attack is not specific to MINBFT and applies to other protocols like PBFT-EA, TRINC, CHEAPBFT, and HYBSTER as they have similar consensus phases and commit rules. It also applies to streamlined protocols HOTSTUFF-M and DAMYSUS which frequently rotate primaries.

Weak Quorums. The smaller set of replicas in TRUST-BFT protocols triggers this responsiveness issue. In TRUST-BFT protocols, a quorum of $f + 1$ matching votes suffices to enforce consensus safety as trusted components certify the position of the transaction in the log and preclude equivocation. Unfortunately, these smaller quorums of $f + 1$ replicas are insufficient to enforce RSM liveness in all current implementations of TRUST-BFT protocols. A quorum of $f + 1$ replicas only guarantees that one honest replica will commit, execute T, and reply to the client with the transaction result. It does not guarantee that $f + 1$ replicas will reply to the client with the transaction result, which is necessary for the client library to validate the execution result and thus return the correct answer to the client.

Existing TRUST-BFT protocols can be modified to support RSM liveness, but at additional cost. This cost is often higher than the $3f + 1$ setup that they sought to improve on. There are three ways to address this issue: checkpointing, added latency, broadcasting.

Replicas	Trusted	Liveness as BFT	Out-of-Order	Memory	Only Primary needs active TC	Protocols
2f + 1	Log	✗	✗	High	✗	PBFT-EA, HotStuff-M
	Counter + Log	✗	✗	Order of Log-size	✗	Trinc, Hybster, Damysus
	Counter	✗	✗	Low	✗	MinBFT, MinZZ, CheapBFT
3f + 1	Counter	✓	✓	Low	✓	FlexiTrust (Flexi-BFT, Flexi-ZZ)

Figure 1. Comparing TRUST-BFT protocols. From left to right: Col1: type of trusted abstraction; Col2: identical liveness guarantees as BFT protocols; Col3: support for out-of-order consensuses; Col4: amount of memory needed; and Col5: only primary replica requires active trusted component. FlexiTrust are our proposals, which we present in this paper.
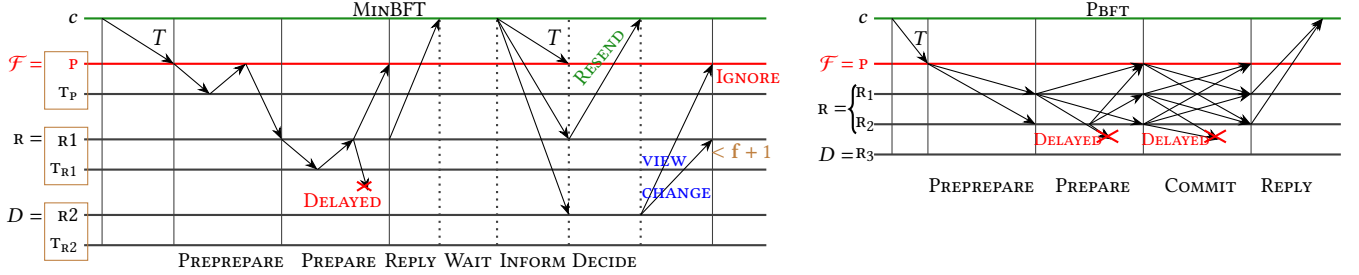

Figure 2. Disruption in service in MinBFT protocol due to weak quorums in comparison to the PBFT protocol (Section 5).

1. *Checkpointing.* Periodic checkpoints will eventually bring honest replicas up-to-date and disseminate the necessary commit certificates. Unfortunately, this implies that clients will incur latency that is directly dependent on the checkpoint frequency (which tend to be relatively infrequent). Moreover, checkpoints require only $f + 1$ replicas to participate, and thus may not immediately include the necessary honest replica.

2. *Added Latency.* A replica could, upon executing the transaction, include both the output and the commit certificate when replying to the client. Such a commit certificate informs the client that its request is successfully committed, but the client still needs matching responses from at least $f + 1$ replicas. Client cannot make progress with just one response because a malicious replica can always forward a commit certificate with incorrect result of executing the operation. The client could then, after a timeout, broadcast the commit certificates to all other replicas, thus informing them that the transaction is committed and can safely be executed. The remaining honest replicas could then execute the relevant operation and reply to the client library. Once the client library receives $f + 1$ responses from replicas, it could then finally return the result to the client. Clearly, this approach introduces an additional round-trip (from 3 to 4 for PBFT-EA) at a time when consensus protocols are concerned about latency [81, 83]. As clients may not be located near the RSM replicas, the added latency may be significant. Notice that a malicious client may fail to forward the commit certificates, further delaying the processing of subsequent (honest) clients.

3. *Broadcast.* Alternatively, upon committing a transaction T, replicas could systematically and preemptively broadcast the commit-certificate for T to other replicas in the system. This additional all-to-all communication phase may cause a significant throughput drop [87], especially when f is large.

What about $3f+1$? Moving to $3f+1$ and quorums of $2f+1$ ensures that all committed operations will be committed at $f + 1$ honest replicas, thus guaranteeing that the client will receive $f + 1$ responses for all operations.

6 Lack of Safety under Rollbacks

Existing TRUST-BFT protocols require some state to be *persisted* on the trusted hardware, corresponding to the logged requests or the counter values. These systems rely on this property to *guarantee* safety. Despite any failures or attacks, these protocols expect this state to remain uncorrupted and available. Unfortunately, realizing this assumption in available implementations of trusted hardware is challenging. Intel SGX enclaves, for instance, are the most popular platform for trusted computing [9, 17, 70, 73], but can suffer from power-failures and rollback attacks. While recent works try to mitigate these attacks, they remain limited in scope and have high costs [30]. Unfortunately, hardware that has been shown to resist these attacks, such as SGX persistent counter [26] or TPM [33], is prohibitively slow: they have upwards of tens of milliseconds access latency and support only a limited number of writes [26, 56, 68].

Persistent state, is, as we show below, necessary for correctness of TRUST-BFT protocols; a rollback attack can cause a node to equivocate. Once equivocation is again possible

in a TRUST-BFT protocol, a single malicious node can cause a *safety violation*. To illustrate, we consider the following run of the MINBFT protocol. Let $\mathcal{F}$ be the set of $\mathbf{f}$ Byzantine replicas, D and G a set of respectively $\mathbf{f}$ and 1 honest replicas.

Assume that the primary P is byzantine and all replicas in $\mathcal{F}$ intentionally fail to send replicas in D any messages. As is possible in a partially synchronous system, we further assume that the PREPARE messages from the replica in G to those in D are temporarily delayed by an amount greater than the client and view change timeouts. P asks its trusted component to generate an attestation for a transaction T to be ordered at sequence number 1. Following this, P sends a PREPREPARE message for T to all replicas in $\mathcal{F}$ and G. These $\mathbf{f} + 1$ replicas are able to prepare T, and they execute T and reply to the client. As a result, the client receives $\mathbf{f} + 1$ identical responses and marks T complete. Now, assume that the byzantine primary P rollbacks the state of its trusted component $\mathbf{T_P}$. Following this, P ask its $\mathbf{T_P}$ to generate an attestation for a transaction T' to be ordered at sequence number 1. Next, P sends a PREPREPARE message for T' to all replicas in D. Similarly, these replicas will be able to prepare and execute T' and the client will receive $\mathbf{f} + 1$ responses. There is a safety violation as replicas in D and G have executed two different transactions at the same sequence number.

How can we solve this? The straightforward approach is to replace all vulnerable enclave accesses with TPMs or persistent counters. While this solution may become viable in the future, we highlight in Figure 8 that it is still impractical.

What about 3f+1? We propose a set of protocols that, by increasing the replication factor to $3\mathbf{f} + 1$, reduce this overhead significantly: they limit the use of TPMs to once per transaction ($O(\mathbf{n})$ times for current TRUST-BFT protocols).

7   Lack of Parallelism

TRUST-BFT protocols are inherently sequential: they order client requests one at a time and cannot support parallel consensus invocations. Pipelining consensus phases can mitigate the performance impact of this approach: it allows for the *Preprepare* phase of transaction $i+1$ to begin directly after the *Preprepare* phase of i (similarly for *Prepare* and *Commit*). Pipelining does not address the root cause of the problem: the sequentiality of TRUST-BFT consensus protocols creates an artificial throughput bound on the throughput they can achieve (batch size / (number phases $\times$ RTT)). This is in direct contrast to traditional BFT protocols which are parallel in nature: replicas can attempt to commit transaction $i + 1$ concurrently with transaction i. As such, their throughput is bound by the available resources in the system. Sequential consensus protocols also perform poorly in the WAN-area as their throughput is directly proportional to phase latency. HYBSTER [13] attempts to mitigate this issue by allowing each of the $\mathbf{n}$ replicas to act as a primary in parallel, but each associated consensus invocation remains sequential.

To illustrate why TRUST-BFT protocols cannot run two instances of consensus in parallel, we assume the following run of MINBFT. Assume that the primary P allows consensus invocations of transactions T_i and T_j ($i < j$) to proceed in parallel. This implies that a replica R may receive PREPREPARE message for T_j before T_i. On receiving a message, R calls the APPEND function to access its $\mathbf{T_R}$. In this example, R would call APPEND on T_j (before T_i) and will receive an attestation $\langle \text{ATTEST}(q, j, T_j) \rangle_{\mathbf{T_R}}$, confirming that the counter value was updated to j and T_j was assigned the value j, which it will forward to all the replicas. When R then receives PREPREPARE message for T_i, its attempt to call the APPEND function would fail. Its $\mathbf{T_R}$ ignores this message as $i < j$ and $\mathbf{T_R}$ cannot process a lower sequence number request. The consensus for T_i will not complete, stalling progress. Similar issues arise for other TRUST-BFT protocols when attempts are made to parallelize their consensus.

What about $3\mathbf{f} + 1$? By increasing the replication factor back to $3\mathbf{f} + 1$, we can design a protocol that has higher throughput *per replica* than the $2\mathbf{f} + 1$ approach (Figure 6(ii)). In this setup, only the leader needs to invoke trusted hardware to order transactions, replicas no longer do. This permits replicas to process transactions in parallel.

8   FLEXITRUST Protocols

The previous sections highlighted several significant limitations with existing TRUST-BFT approaches, all inherited from their lower replication factor. In this section, we make two claims: (i) $2\mathbf{f} + 1$ is simply *not enough*: it either impacts responsiveness or requires an extra phase of all-to-all communication (§5); it requires the use of slow persistent trusted counters for every message (§6); and it sequentializes consensus decisions(§7). (ii) Trusted components *are still beneficial* to BFT consensus if used with $3\mathbf{f} + 1$ replicas as they can be used to reduce either the number of phases or the communication complexity. To this effect, we present FLEXITRUST, a new set of BFT consensus protocols that make use of trusted components. These protocols satisfy both the RSM and consensus safety/liveness conditions described in Section 4, and, through the use of $3\mathbf{f} + 1$ replicas, achieve the following appealing performance properties:

(G1) **Parallel Consensus.** FLEXITRUST protocols allow replicas to process consensus invocations concurrently.

(G2) **Minimal Trusted Component Use.** FLEXITRUST protocols require accessing a single trusted component per transaction rather than one per message. This is especially important when using TPMs or persistent counters as the counter/logging service to preclude rollback attacks.

(G3) **No Trusted Logging.** Moreover, FLEXITRUST protocols maintain low memory utilization at trusted components as they do not require trusted logging.

8.1 Designing a FlexiTrust protocol

We make three modifications to TRUST-BFT protocols. Together, these steps are sufficient to achieve significantly greater performance, and better reliability.

First, we modify the APPEND functionality. Recall that the participants use this function to bind a specific message with a counter value. The value of this counter can be supplied by the replica but must increase monotonically; no two messages can be bound to the same value. We restrict this function to preclude replicas from supplying their own sequence number, and instead have the trusted component increment counters internally, thus ensuring that counter values will remain contiguous.

APPENDF(q, x) – Assume the q-th counter of T_R has value k. This function increments k to $k+1$, associates k with message x, and returns an attestation $\langle \text{ATTEST}(q, k, x) \rangle_{T_R}$ as a proof of this binding.

This change is necessary to support parallel consensus instances efficiently: while multiple transactions can be ordered in parallel, the execution of these transactions must still take place in sequence number order. Existing APPEND functionality allows a Byzantine replica to either stall the system's progress, or exhaust the defined range of sequence numbers by issuing a sequence number that is far in the future. As a consequence, honest replicas are forced to frequently trigger view changes to "fill" the gap with no-ops [15] and update high and low watermark range.

CREATE(k) – Creates a new counter with identifier q and initial counter value k, such that no previous counter has an identifier q. This function also returns an attestation $\langle \text{ATTEST}(q, k) \rangle_{T_R}$.

We additionally make use of the standard functionality of creating new counters [26, 51]. This function helps the new primary (post view change) to re-start consensus on previously proposed requests. For each new counter that a replica creates, it has to share a certificate (attestation) that proves the newness of this counter.

Second, we ensure that only the leader (not the replicas) need to invoke a trusted counter. All other participants simply validate the trusted counter's signature when receiving a message from the primary. Specifically, upon receiving a client request $m := T$, the primary invokes APPENDF(q, m) to bind a unique counter value k to m and returns an attestation $\langle \text{ATTEST}(q, k, m) \rangle_{T_P}$. The primary then forwards this attestation as part of its first consensus phase. This allows replicas to process transactions in parallel; once the transactions have been assigned a sequence number at the leader (using APPENDF), replicas can process them out-of-order and thus in parallel.

Finally, we increase the quorum size necessary to proceed to the next phase of consensus to $2f + 1$. This higher quorum size guarantees that any two quorums will intersect in at least $f + 1$ distinct replicas (and thus in one honest replica).

Initialization:
// q be the latest counter with value k.
// v be the view-number, determined by the identifier of primary.

Client-role (used by client c to process transaction T):
1: Sends $\langle T \rangle_c$ to the primary P.
2: Awaits receipt of messages RESPONSE($\langle T \rangle_c, k, v, r$) from $f + 1$ replicas.
3: Considers T executed, with result r, as the k-th transaction.

Primary-role (running at the primary P):
4: **event** P receives $\langle T \rangle_c$ **do**
5:    Calculate digest $\Delta := \text{Hash}(\langle T \rangle_c)$.
6:    $\{k, \sigma\} := \text{APPENDF}(q, \Delta)$
7:    Broadcast PREPREPARE($\langle T \rangle_c, \Delta, k, v, \sigma$).

Non-Primary Replica-role (running at the replica R):
8: **event** R receives PREPREPARE($\langle T \rangle_c, \Delta, k, v, \sigma$) from P such that:
   - Message is well-formed and attestation σ is valid.
   - R did not accept a k-th proposal from P.
do
9:    Broadcast PREPARE(Δ, k, v, σ).

Replica-role (running at **any** replica R):
10: **event** R receives PREPARE(Δ, k, v, σ) messages from $2f + 1$ replicas such that:
    all the messages are well-formed and identical. **do**
11:    **if** R has executed transaction with sequence number $k - 1 \wedge k > 0$ **then**
12:       Execute T as the k-th transaction.
13:       Let r be the result of execution of T (if there is any result).
14:       Send RESPONSE($\langle T \rangle_c, k, v, r$) to c.
15:    **else**
16:       Place T in queue for execution.

Trusted-role (running at the trusted component T_P):
17: **event** T_P is accessed through APPENDF(q, Δ) **do**
18:    Increment q-th counter; $k := k + 1$
19:    $\sigma :=$ Generate attestation $\langle \text{ATTEST}(q, k, \Delta) \rangle_{T_P}$.
20:    Return $\{k, \sigma\}$.

Figure 3. FLEXI-BFT protocol (failure-free path).

Forcing an honest replica to be part of every quorum makes the need for accessing a trusted counter redundant as this replica will, by definition, never equivocate.

8.2 Case Study: FLEXI-BFT

We apply our transformations to MINBFT [83], a two-phase TRUST-BFT protocol that makes use of trusted counters. MINBFT requires one less phase than PBFT and PBFT-EA (two-phases total): as the primary cannot equivocate, it is safe to commit a transaction in MINBFT after receiving $f + 1$ PREPARE messages. Note that, in its current form, MINBFT does not guarantee consensus liveness (§ 5) and would need an extra phase to do so. FLEXI-BFT, the *new protocol* that we develop, preserves this property, but remains responsive and makes minimal use of trusted components (once per consensus). The view-change and checkpointing protocols remain identical to the PBFT view-change, we do not discuss them in detail here. We include pseudocode in Figure 3.

As stated, FLEXI-BFT consists of two phases. Upon receiving a transaction T, the primary P of view v requests its trusted component to generate an attestation for T. This attestation $\langle \text{ATTEST}(q, k, m) \rangle_{T_P}$ states that T will be ordered at position k (Line 6, Figure 3). The primary broadcasts a PREPREPARE with this proof to all replicas. When a replica R receives a valid PREPREPARE message in view v, it marks

the transaction T as *prepared*. Prepared transactions have the property that no conflicting transaction has been prepared for the same counter value in the same view. In the PBFT protocol, an additional round is necessary to mark messages as prepared, as replicas can equivocate. The replica R then broadcasts a PREPARE message in support of T and includes the attestation. When R receives PREPARE messages from $2f + 1$ distinct replicas in the same view, it marks T as *committed*. R will execute T once all transactions with sequence numbers smaller than k have been executed. The client marks T as complete when it receives matching responses from $f + 1$ replicas.

Replicas initiate the view-change protocol when they suspect that the primary has failed. As stated previously, the view-change logic is identical to the PBFT view-change; we only describe it here briefly. A replica in view v enters the view change by broadcasting a VIEWCHANGE message to all replicas. Each VIEWCHANGE message sent by a replica R includes all the valid *prepared* and *committed* messages received by R with relevant proof (the trusted component attestation for the PREPREPARE message and the $2f+1$ PREPARE messages for committed messages). The new primary starts the new view if it receives VIEWCHANGE message from $2f+1$ replicas for view $v + 1$.

8.3 Case Study: FLEXI-ZZ

We now transform MINZZ [83] into our novel FLEXI-ZZ protocol. MINZZ follows the design proposed by ZYZZYVA [48]. ZYZZYVA introduces a BFT consensus with a single-phase fast-path (when all replicas are honest and respond) and a two-phase slow-path. MINZZ uses trusted counters to reduce the replication factor from $3f + 1$ to $2f + 1$. The cost of transforming MINZZ to FLEXI-ZZ is that, once again, we use $3f + 1$ replicas. However, there are several benefits (1) FLEXI-ZZ can always go fast-path as it only requires $n - f$ matching responses (compared to the n for both MINZZ and ZYZZYVA). This helps improve performance under byzantine attacks, which past work has demonstrated is an issue for ZYZZYVA [23, 24, 38]. (2) FLEXI-ZZ minimizes the use of trusted components: a single access to a trusted counter is required at the primary per consensus invocation. (3) FLEXI-ZZ's view-change is significantly simpler than ZYZZYVA's view-change. View-change protocols are notorious complex and error-prone [2, 31] to design and implement; a simple view-change protocol thus increases confidence in future correct deployments and implementations. We present pseudocode in Figure 4.

Common Case. A client c submits a new transaction T by sending a signed message $\langle T \rangle_c$ to the primary P. The primary P invokes the APPENDF(q, m) function, binding the transaction to a specific counter value k and returning an attestation $\langle \text{ATTEST}(q, k, m) \rangle_{T_P}$ as proof. This step prevents P from assigning the same sequence number k to two conflicting messages m and m'. The primary then forwards this

---

Initialization: // q be the latest counter with value k.

Client-role (used by client c to process transaction T):
1: Sends $\langle T \rangle_c$ to the primary P.
2: Awaits receipt of messages RESPONSE$(\langle T \rangle_c, k, v, r)$ from $2f + 1$ replicas.
3: Considers T executed, with result r, as the k-th transaction.

Primary-role (running at the primary P of view v):
4: **event** P receives $\langle T \rangle_c$ **do**
5:      Calculate digest $\Delta := \text{Hash}(\langle T \rangle_c)$.
6:      $\{k, \sigma\} := \text{APPENDF}(q, \Delta)$.
7:      Broadcast $m := \text{PREPREPARE}(\langle T \rangle_c, \Delta, k, v, \sigma)$.
8:      Execute(m).

Non-Primary Replica-role (running at the replica R):
9: **event** R receives $m := \text{PREPREPARE}(\langle T \rangle_c, \Delta, k, v, \sigma)$ from P such that:
   - Message is well-formed and attestation σ is valid.
   - R did not accept a k-th proposal from P.
   do
10:      Execute(m).

Trusted-role (running at the trusted component T_P):
11: **event** T_P is accessed through APPENDF(q, Δ) **do**
12:      Increment q-th counter; $k := k + 1$
13:      $\sigma := \text{Generate attestation} \langle \text{ATTEST}(q, k, \Delta) \rangle_{T_P}$.
14:      Return $\{k, \sigma\}$.

15: **function Execute** (message: m)
16:      **if** R has executed transaction with sequence number $k - 1 \wedge k > 0$ **then**
17:          Execute T as the k-th transaction.
18:          Let r be the result of execution of T (if there is any result).
19:          Send RESPONSE$(\langle T \rangle_c, k, v, r)$ to c.
20:      **else**
21:          Place T in queue for execution.

Figure 4. FLEXI-ZZ protocol (common-case).

attestation along with the transaction to all replicas. Replicas, upon receiving this message, execute the transaction in sequence order, and reply directly to the client with the response. The client marks the transaction T complete when it receives $2f + 1$ identical responses in matching views.

View Change. If the client does not receive $2f + 1$ matching responses, it re-broadcasts its transaction to all replicas; the primary may have been malicious and failed to forward its request. Upon receiving this broadcast request, a replica either (1) directly sends a response (if it has already executed the transaction T) or, (2) forwards the request to the primary and starts a timer. If the timer expires before the replica receives a PREPREPARE message for T, it initiates a view-change. Specifically, the replica enters view $v + 1$, stops accepting any messages from view v, and broadcasts a VIEWCHANGE message to all replicas. VIEWCHANGE messages include all requests for which R has received a PREPREPARE message.

Upon receiving VIEWCHANGE messages from $2f + 1$ replicas in view $v + 1$, the replica designated as the primary for view $v + 1$ (say P') creates a NEWVIEW message and broadcasts it to all replicas. This message includes: (1) the set of VIEWCHANGE messages received by the primary as evidence, and (2) the (sorted-by-sequence-number) list of transactions that may have committed. The primary P' creates a new trusted counter and sets it to the transaction with the lowest sequence number. P' then re-proposes all transactions in this list, proposing specific no-op operations when there is a gap in the log between two re-proposed transactions.

To re-propose these transactions, the primary proceeds in the standard fashion: it accesses its trusted counter, obtains a unique counter value (setting the counter to the transaction with the lowest sequence number ensures that sequence numbers remain the same across views), and broadcasts the transaction and its attestation as part of a new PREPREPARE message. This mechanism guarantees that all transactions that *could* have been perceived by the client as committed (the client receiving $2f + 1$ matching replies) will be re-proposed in the same order: for the client to commit an operation, it must receive $2f + 1$ matching votes; one of those those votes is thus guaranteed to appear the NEWVIEW message. Transactions that were executed by fewer than $2f + 1$ replicas, on the other hand, may *not* be included in the new view, which may force some replicas to rollback.

9 Proofs

We first prove that in FLEXITRUST protocols, no two honest replicas will execute two different requests at the same sequence number.

Theorem 2. *Let R_i, $i \in \{1, 2\}$, be two honest replicas that executed $\langle T_i \rangle_{c_i}$ as the k-th transaction of a given view v. If $\mathbf{n} = 3f + 1$, then $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$.*

Proof. Replica R_i only executed $\langle T_i \rangle_{c_i}$ after R_i received a well-formed PREPREPARE message (PREPREPARE($\langle T \rangle_c, \Delta, k, v, \sigma$)) from the primary P. This message includes an attestation $\sigma = \langle \text{ATTEST}(q, k, \Delta) \rangle_{T_P}$ from T_P, which we assume cannot be compromised. Let S_i be the replicas that received PREPREPARE message for $\langle T_i \rangle_{c_i}$. Let $X_i = S_i \setminus \mathcal{F}$ be the honest replicas in S_i. As $|S_i| = 2f + 1$ and $|\mathcal{F}| = f$, we have $|X_i| = 2f + 1 - f$. An honest replica in T_i will only execute the k-th transaction in view v if it has an attestation from the trusted component at the primary. If $\langle T_1 \rangle_{c_1} \neq \langle T_2 \rangle_{c_2}$, then X_1 and X_2 must not overlap as the trusted component will never assign them the same sequence number. Hence, $|X_1 \cup X_2| \geq 2(2f + 1 - f)$. This simplifies to $|X_1 \cup X_2| \geq 2f + 2$, which contradicts $\mathbf{n} = 3f + 1$. Thus, we conclude $\langle T_1 \rangle_{c_1} = \langle T_2 \rangle_{c_2}$. □

Next, we show that both FLEXI-BFT and FLEXI-ZZ guarantee a safe consensus.

Theorem 3. *In a system $\mathcal{S} = \{\mathcal{R}, C\}$ where $|\mathcal{R}| = \mathbf{n} = 3f + 1$, FLEXI-BFT protocol guarantees a safe consensus.*

Proof. If the primary P is honest, then from Theorem 2, we can conclude that no two replicas will execute different transactions for the same sequence number. This implies that all the honest replicas will execute the same transaction per sequence number. If a transaction is executed by at least $f + 1$ honest replicas, then it will persist across views as in any view-change quorum of $2f + 1$ replicas, there will be one honest replica that has executed this request and has a valid PREPREPARE message and $2f + 1$ PREPARE messages corresponding to this request.

If the primary P is byzantine, it can only prevent broadcasting the PREPREPARE messages to a subset of replicas. P cannot equivocate as it does not assign sequence numbers. For each transaction T, P needs to access its T_P, which returns a sequence number k and an attestation that binds k to T. Further, Theorem 2 proves that for a given view, no two honest replicas will execute different transactions for the same sequence number. So, a byzantine P can send the PREPREPARE for T: (i) to at least $2f + 1$ replicas, or (ii) to less than $2f + 1$ replicas. In either cases, any replica R that receives T will send the PREPARE message. If R receives $2f + 1$ PREPARE messages, it will execute T and reply to the client. Any remaining replica that did not receive T will eventually timeout waiting for a request and trigger a VIEWCHANGE. If at least $f + 1$ replicas timeout, then a VIEWCHANGE will take place. If T was prepared by at least $f + 1$ honest replicas, then this request will be part of the subsequent view. Otherwise, the subsequent view may or may not include T. But this should not be an issue because such a transaction was not executed by any honest replica; no replica would have received $2f + 1$ PREPARE messages. Hence, system is safe even if this transaction is forgotten.

Each new view $v + 1$ is led by a replica with identifier i, where $i = (v + 1) \mod \mathbf{n}$. The new primary waits for VIEWCHANGE messages from $2f + 1$ replicas, uses these messages to create a NEWVIEW message, and forwards these messages to all the replicas. This NEWVIEW message includes a list of requests for each sequence number present in the VIEWCHANGE message. The new primary needs to set its counter to the lowest sequence number of this list (may need to create a new counter). Post sending the NEWVIEW message, the new primary re-proposes the PREPREPARE message for each request in the NEWVIEW. Each replica on receiving the NEWVIEW message can verify its contents. □

Theorem 4. *In a system $\mathcal{S} = \{\mathcal{R}, C\}$ where $|\mathcal{R}| = \mathbf{n} = 3f + 1$, FLEXI-ZZ protocol guarantees a safe consensus.*

Proof. If the primary P is honest, then from Theorem 2, we can conclude that no two replicas will execute different transactions for the same sequence number. This implies that all the honest replicas will execute the same transaction per sequence number. If a transaction is executed by at least $f + 1$ honest replicas, then it will persist across views as in any view-change quorum of $2f + 1$ replicas, there will be one honest replica that has executed this request and has a valid PREPREPARE message.

If the primary P is byzantine, it can only prevent broadcasting the PREPREPARE messages to a subset of replicas. P cannot equivocate as it does not assign sequence numbers. For each transaction T, P needs to access its T_P, which returns a sequence number k and an attestation that binds k to T. Further, Theorem 2 proves that for a given view, no two honest replicas will execute different transactions for the same sequence number. So, a byzantine P can send the

PREPREPARE for T: (i) to at least $2\mathbf{f} + 1$ replicas, or (ii) to less than $2\mathbf{f} + 1$ replicas. In either cases, any replica that receives T will execute it and reply to the client, while the remaining replicas will eventually timeout waiting for a request and trigger a VIEWCHANGE. If at least $\mathbf{f} + 1$ replicas timeout, then a VIEWCHANGE will take place. If T was executed by at least $\mathbf{f} + 1$ honest replicas, then this request will be part of the subsequent view. Otherwise, the subsequent view may or may not include T. In such a case, any replica that executed T would be required to rollback its state. However, for any request if the client receives $2\mathbf{f} + 1$ responses, it will persist across views because at least $\mathbf{f} + 1$ honest replicas have executed that request.

Each new view $v + 1$ is led by a replica with identifier i, where $i = (v + 1) \bmod \mathbf{n}$. The new primary waits for VIEWCHANGE messages from $2\mathbf{f} + 1$ replicas, uses these messages to create a NEWVIEW message, and forwards these messages to all the replicas. This NEWVIEW message includes a list of requests for each sequence number present in the VIEWCHANGE message. The new primary needs to set its counter to the lowest sequence number of this list (may need to create a new counter). Post sending the NEWVIEW message, the new primary re-proposes the PREPREPARE message for each request in the NEWVIEW. Each replica on receiving the NEWVIEW message can verify its contents.   □

10   Evaluation

The goal of our evaluation is to gauge how our FLEXITRUST protocols fare against their TRUST-BFT and BFT counterparts. To do so, we ask three *core* questions. (1) How do our FLEXITRUST protocol perform and scale? (§10.4 to §10.7) (2) What is the impact of failures? (§10.8) (3) How will these protocols behave as hardware technology evolves? (§10.9 and §10.10)

10.1   Implementation

We use the open-source RESILIENTDB fabric to implement all the consensus protocols [35, 37, 38, 65]. RESILIENTDB supports all standard BFT optimizations, including multi-threading at each replica and both client and server batching. The system relies on CMAC for MAC, ED25519 for DS-based signatures and SHA-256 for hashing.

SGX Enclaves. We use *Intel SGX for Linux* [26] to implement the abstraction of a trusted component at each replica. Specifically, we implement multiple monotonically increasing counters inside each enclave, which can be concurrently accessed by multiple threads through the function GetSequenceNo(<counter-id>). This API call returns an attestation that includes the latest value of the specific counter and a DS that proves that this counter value was generated by the relevant trusted component. To highlight the potential of trusted components under the $3\mathbf{f} + 1$ regime, we implement counters inside of the SGX enclave instead of leveraging Intel SGX Platform Services for trusted counters
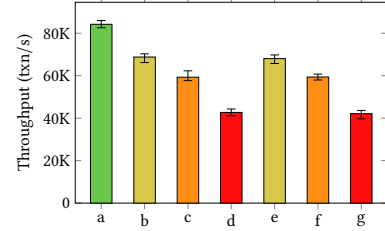

Figure 5. Impact of trusted counter (TC) and signature attestations (SA) on PBFT. [a] Standard PBFT protocol. [b] Primary (P) requires TC in Preprepare phase (Prep). [c] P requires TC and SA in Prep. [d] P requires TC and SA in all three phases. [e] All replicas require TC in Prep. [f] All replicas require TC and SA in Prep. [g] All replicas require TC and SA in all three phases.

as they have prohibitively high latency and are not available on most cloud providers. All protocols (including baselines and FLEXITRUST protocols) are thus subject to rollback attacks in the current experimental setup. We highlight the trade-offs associated with the choice of trusted hardware in Section 10.9.

10.2   Evaluation Setup

We compare our FLEXITRUST protocols against *eight* baselines: (i) PBFT [15], available with RESILIENTDB, as it outperforms the BFTSmart's [14] implementation, which is single-threaded and sequential [35, 38, 65, 81]; (ii) ZYZZYVA [48], a linear single phase BFT protocol where client expects responses from all the $3\mathbf{f} + 1$ replicas; (iii) PBFT-EA [21], a three phase TRUST-BFT protocol; (iv) MINBFT [83], a two phase TRUST-BFT protocol; (v) MINZZ [83], a linear single phase TRUST-BFT protocol where client expects responses from all the $2\mathbf{f} + 1$ replicas; and (vi) OPBFT-EA, a variation of PBFT-EA we develop that supports parallel consensus invocations. (vii) oFLEXI-BFT and (viii) oFLEXI-ZZ , variations of FLEXI-BFT and FLEXI-ZZ with no parallel consensus invocations. In all these protocols, we enable pipelining. We parallelize cryptographic computations in all the protocols. We do not compare against streamlined BFT protocols such as Hotstuff [87] or DAMYSUS [28], as their chained nature precludes concurrent consensus invocations.

We use the Oracle Cloud Infrastructure (OCI) and deploy up to 97 replicas on VM.Standard.E3.Flex machines (16 cores and 16 GiB RAM) with 10 GiB NICs. Unless explicitly stated, we use the following setup: each system supports up to $\mathbf{f} = 8$ Byzantine replicas. We intentionally choose a higher $\mathbf{f}$ to maximize the potential cost of increasing the replication factor to $3\mathbf{f} + 1$. Clients run in a closed-loop; each experiment runs for 180 seconds (60 seconds warmup/cooldown) and we report average throughput/latency over three runs. We adopt the popular *Yahoo Cloud Serving Benchmark* (YCSB), [25, 29, 38, 42]. YCSB generates key-value store operations that access a database of 600 k records.
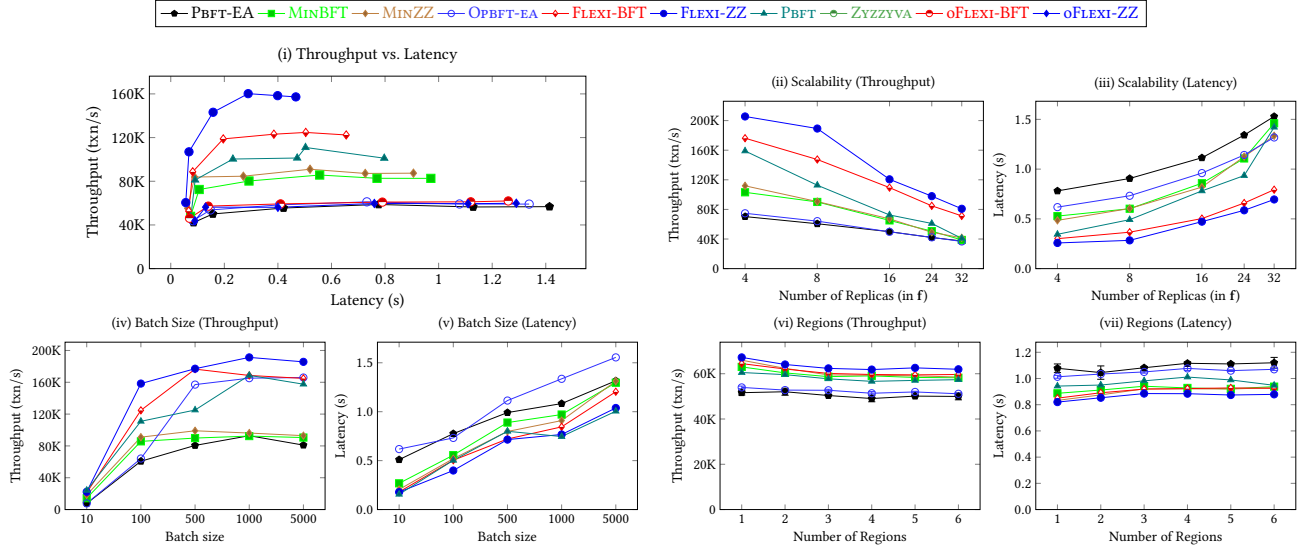
Suyash Gupta[†], Sajjad Rahnama, Shubham Pandey, Natacha Crooks[†], Mohammad Sadoghi

Figure 6. Throughput results as a function of number of clients, number of replicas, batch size and cross-site latency. We set the number of replicas in f as for BFT and FLEXITRUST protocols $n = 3f + 1$ and for TRUST-BFT and OPBFT-EA $n = 2f + 1$.

10.3 Trusted Counter Costs. In Figure 5, we quantify the costs of accessing trusted counters. To do so, we run a PBFT implementation with a single worker-thread. Bar [a] represent our baseline implementation; we report peak throughput numbers for each setup. Throughput degradation occurs when the primary replica needs to access the trusted component (Bar [b]). This degradation accelerates when the primary replica requires trusted component to perform signature attestations (Bar [c]), and needs to perform these operations during each phase of consensus (Bar [d]). The drop in throughput from [a] to [d] is nearly 2×. Bars [e] to [g] extend the use of trusted components to non-primary replicas. The system is already bottlenecked at the primary replica (it must process more messages than replicas); this change thus has no impact on performance

10.4 Throughput Results. In Figure 6(i), we increase the number of clients from 4 k to 80 k and report on latency and throughput. PBFT-EA achieves the lowest throughput as it requires three phases for consensus and disallows parallel consensus invocations. The reduced replication factor of $2f + 1$ does not help performance as threads are already under-saturated: the system is latency-bound rather than compute bound due to the protocol's sequential processing of consensus invocations. OPBFT-EA protocol attains up to 6% higher throughput (and lower latency) than PBFT-EA as it supports parallel consensus invocations but bottlenecks on trusted counter accesses at replicas. Specifically, the replica's worker thread has to sign the outgoing message, and perform two verifications on each received message: (i) MAC of the received message, and (ii) DS of the attestation from the trusted component. MINBFT and MINZZ achieve up to 47% and 68% higher throughputs than PBFT-EA respectively, as

they reduce the number of phases necessary to commit an operation (from three to two for MINBFT, and from three to one in the failure-free case for MINZZ). Interestingly, PBFT yields better throughput than all TRUST-BFT protocols. The combination of parallel consensus invocations and lack of overhead stemming from the use of trusted counters drives this surprising result. Our FLEXI-BFT and FLEXI-ZZ protocols instead achieve up to 22% and 58% higher throughput than PBFT, (and up to 87% and 77% higher throughput over MINBFT and MINZZ). This performance improvement stems from reducing the number of phases, accessing a trusted counter once per transaction, and permitting parallel consensus invocations. Note that supporting parallel consensus is key to these performance gains. Without this parallelism, our FLEXITRUST protocols perform worse than their TRUST-BFT counterparts (OFLEXI-BFT yields 33% less throughput than MINZZ) as the primary needs to sequentially attest an additional f messages.

10.5 Scalability. Figures 6(ii) and 6(iii) summarize the protocols' scaling behavior as the number of replicas increases from $f = 4$ to $f = 32$. As expected, an increased replication factor leads to a proportional increase in the number of messages that are propagated and verified. This increased cost leads to a significant drop in the latency and throughput of all protocols: going from $f = 4$ to $f = 8$ causes PBFT, FLEXI-BFT, and FLEXI-ZZ's performance to drop 3.89×, 2.48×, and 2.54× respectively. MINBFT, and MINZZ's throughput also drops by a factor of 2.66× and 2.67×. This performance drop is larger for BFT and FLEXITRUST protocols than for TRUST-BFT protocols as replicas are never fully saturated due to sequential consensus invocations.
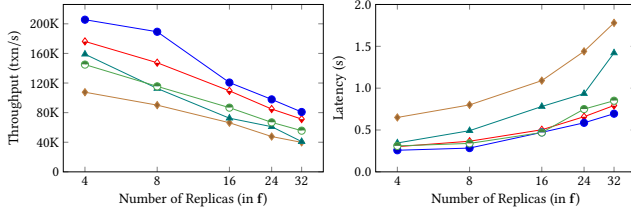
Figure 7. Impact of failure of one non-primary replica.

Access cost (in ms)	Flexi-ZZ	MinZZ	MinBFT
1.0	87 k	49 k	39 k
1.5	67 k	48 k	37 k
2.0	50 k	47 k	35 k
2.5	40 k	40 k	34 k
3.0	34 k	33 k	32 k
10	10 k	10 k	10 k
30	3 k	3 k	3 k
100	993	959	994
200	494	479	496

Figure 8. Peak throughput (in transactions per second) on varying the time taken to access a trusted counter while running consensus among 97 replicas.

10.6 Batching. Figures 6(iv) and 6(v) quantify the impact of batching client requests as we increase the batch size from 10 to 5 k. As expected, the throughput of all protocols increases as batch sizes increase until communication becomes a bottleneck.

10.7 Wide-area replication. For this experiment (Figures 6(vi) and 6(vii)), we distribute the replicas across five countries in six regions: San Jose, Ashburn, Sydney, Sao Paulo, Montreal, and Marseille, and use the regions in this order. We set $f = 20$; $n = 41$ and $n = 61$ replicas for $2f + 1$ and $3f + 1$ protocols, respectively.[1] To observe the gradual change in performance, on increasing the number of regions, for all protocols, we place the leader in San Jose.

Latency and throughput remain mostly constant as the number of regions increases. We attribute this phenomena to the following reason. Each replica needs to wait for only a quorum of messages, $f + 1$ (21) or $2f + 1$ (41), before it can transit to the next phase. These systems thus need to wait only for responses of North American replicas (San Jose, Ashburn, Montreal), which are connected by links that provide high bandwidth and low round-trip costs. The increase in latency or decrease in throughput is thus comparatively small. Importantly, the visible changes are within the margin of error bounds as they are averaged over multiple runs.

10.8 Single Replica Failure. Next, we consider the impact of failures on our protocols (Figure 7). Unlike MinZZ and Zyzzyva, our Flexi-ZZ protocol's performance does not degrade as it can handle up to f non-primary replica failures

---

[1]As the LCM of numbers in the range [1, 6] is 60, so we set $3f + 1 = 61$, which yields $f = 20$.

Replicas (in f)	Flexi-ZZ	MinZZ
4	15813	12431
8	7570	5329
16	2462	2038
24	1341	1002
32	834	640

Figure 9. Throughput-per-machine: total system throughput/(number of replicas)

on the fast path. In contrast, both MinZZ and Zyzzyva require their clients to receive responses from all replicas; in order to commit in a single round-trip.

10.9 Real-World Adoption

This paper's objective is to highlight current limitations of existing trust-bft approaches, be it hardware-related or algorithmic. Trusted hardware, however, is changing rapidly. Current SGX enclaves are subject to rollback attacks, but newer enclaves (Keystone, Nitro) may not be. Similarly, accessing current SGX persistent counters or TPMs currently takes between 80 ms to 200 ms for TPMs and between 30 ms to 187 ms for SGX [51, 58, 68]. New technology is rapidly bringing this cost down; counters like ADAM-CS [57] requires less than 10 ms.

Our final experiment aims to investigate the current performance of trust-bft protocols on both present and future trusted hardware. Our previous results were obtained using counters inside of SGX enclave as hardware providing access to SGX persistent counters and TMPs are not readily available on cloud providers. In our experiments so far, *all* trust-bft and FlexiTrust protocols were subject to rollback attacks. In this experiment, we gage the impact of throughput and latency as we increase the time to access the trusted counter (Figure 8) on Flexi-ZZ, MinBFT and MinZZ protocols. We run this experiment on 97 replicas, and highlight that, for this setup Pbft yields 40 ktxn/s. We find that Flexi-ZZ outperforms all protocols as long as the latency is less than 2.5 ms. Beyond this value, a single access to trusted hardware becomes the bottleneck; causing all protocols' performance to degrade to similar values (eg. at 10 ms, 10 k can be directly obtained by batch size × 1 s / 10 ms). This result highlights a path whereby, as trusted hardware matures, trust-bft protocols will become an appealing alternative to standard bft approaches.

10.10 Throughput-Per-Machine. trust-bft protocols seek to reduce the hardware necessary to deploy bft consensus; additional replicas increase operational complexity and resource costs. In fact, the high costs of accessing trusted hardware for every message combined with the lack of parallelism that results from a $2f + 1$ replication factor decreases overall system throughput. We find that reverting to $3f + 1$ actually *increases* the *throughput-per-machine* performance of the system (for the reasons outlined above). Per machine, $3f + 1$ FlexiTrust protocol achieve higher throughput than a $2f + 1$ trust-bft protocol (up to 30%, as shown in Figure 9).

11 Related Work

There is long line of research on designing efficient BFT protocols [1, 8, 10, 11, 15, 31, 35, 37, 38, 47, 48, 64, 72, 81, 87, 90] and BFT-based blockchain applications [5, 6, 16, 18, 19, 34, 44, 55, 61, 63, 88, 89]. As stated, TRUST-BFT protocols prevent replicas from equivocating, reducing the replication factor or the number of phases necessary to achieve safety. We summarized PBFT-EA in Section 4 and now describe other TRUST-BFT protocols.

CHEAPBFT [46] uses trusted counters and optimizes for the *failure-free* case by reducing the amount of *active* replication to $f + 1$. When a failure occurs, however, CHEAPBFT requires that all $2f + 1$ replicas participate in consensus. The protocol has the same number of phases as MINBFT; as higlighted in Sections 5 and 7, the protocol is inherently sequential and may not be responsive to clients.

HYBSTER [13] is a meta-protocol that takes as an input an existing TRUST-BFT protocol (such as PBFT-EA, MINBFT, etc.) and requires each of the n replicas to act as parallel primaries; a common deterministic execution framework will consume these local consensus logs to execute transactions in order. While multiple primaries improve concurrency, each primary locally invokes consensus in-sequence; each sub-log locally inherits the limitations of existing TRUST-BFT protocols. There continues to be an artificial upper-bound on the amount of parallelism supported in the system. Moreover, recent work shows that designing multiple primary protocols is hard as f of these primaries can be byzantine and can collude to prevent liveness [37, 80].

Streamlined protocols like HOTSTUFF-M [86] and DAMYSUS [28] follow the design of HOTSTUFF [87]. They linearize communication by splitting the all-to-all communication phases (PREPARE and COMMIT) into two linear phases. These systems additionally rotate the primary after each transaction, requiring $f + 1$ replicas to send their last committed message to the next primary. The next primary then selects the committed message with the highest view number as the baseline for proposing the next transaction for consensus. HOTSTUFF-M makes use of trusted logs; DAMYSUS requires its replicas to have trusted components that provide support for both logs and counters. Specifically, DAMYSUS requires two types of trusted components at each replica, an accumulator and a checker. The primary leverages the accumulator to process incoming messages and create a certificate summarizing the last round of consensus. Each replica instead accesses the checker to generate sequence numbers using a monotonic counter and logs information about previously agreed transactions. These protocols once again suffer from a potential lack of responsiveness; their streamlined nature precludes opportunities to support any concurrency [38, 65].

Microsoft's CCF framework uses Intel SGX to support building confidential and verifiable services [73, 78]. CCF provides a framework that helps to generate an audit trail for each request. To do so, they log each request and have it attested by the trusted components. CCF provides flexibility of deploying any consensus protocol.

In the specific case of blockchain systems, Teechain [53] designs a two-layer payment network with the help of SGX. Teechain designates trusted components as treasuries and only allows them to manage user funds. Teechain permits a subset of treasuries to be compromised, and it handles such attacks by requiring each fund to be managed by a group of treasuries. Ekiden [20] executes smart contracts directly in the trusted component for better privacy and performance. Avoine et al. [12] provide a good theoretical treatment of fair-exchange problem using trusted hardware. Aguilera et al. [3, 4] reduce the replication factor from $3f + 1$ to $2f + 1$, without relying on trusted hardware, with the help of disaggregated memory. They assume that each memory block permits only one writer and multiple readers.

12 Conclusion

In this paper, we identified three challenges with the design of existing TRUST-BFT protocols: (i) they have limited responsiveness, (ii) they suffer from safety violations in the presence of rollback attacks, and (iii) their sequential nature artificially bounds throughput. We argue that returning to $3f + 1$ is the key to fulfilling the potential of trusted components in BFT. Our suite of protocols, FLEXITRUST, supports parallel consensus instances, makes minimal use of trusted components, and reduces the number of phases necessary to safely commit operations while also simplifying notoriously complex mechanisms like view-changes. In our experiments, FLEXITRUST protocols outperform their BFT and TRUST-BFT counterparts by 100% and 185% respectively.

Acknowledgments

References

[1] Ittai Abraham, Natacha Crooks, Neil Giridharan, Heidi Howard, and Florian Suri-Payer. 2022. Brief Announcement: It's not easy to relax: liveness in chained BFT protocols. In *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA (LIPIcs, Vol. 246)*, Christian Scheideler (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:3. https://doi.org/10.4230/LIPIcs.DISC.2022.39

[2] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. 2017. Revisiting Fast Practical Byzantine Fault Tolerance. https://arxiv.org/abs/1712.01367

[3] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. 2023. UBFT: Microsecond-Scale BFT Using Disaggregated Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 862–877. https://doi.org/10.1145/3575693.3575732

[4] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Dalia Papuc, Athanasios Xygkis, and Igor Zablotchi. 2021. Frugal Byzantine Computing. In *35th International Symposium on Distributed Computing (DISC 2021)*, Vol. 209. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:19. https://doi.org/10.4230/LIPIcs.DISC.2021.3

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. CAPER: A Cross-application Permissioned Blockchain. *Proc. VLDB Endow.* 12, 11 (2019), 1385–1398. https://doi.org/10.14778/3342263.3342275

[6] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. *SharPer: Sharding Permissioned Blockchains Over Network Clusters*. Association for Computing Machinery, New York, NY, USA, 76–88.

[7] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 30:1–30:15. https://doi.org/10.1145/3190508.3190538

[8] Karolos Antoniadis, Antoine Desjardins, Vincent Gramoli, Rachid Guerraoui, and Igor Zablotchi. 2021. Leaderless Consensus. In *41st IEEE International Conference on Distributed Computing Systems*. IEEE, 392–402. https://doi.org/10.1109/ICDCS51616.2021.00045

[9] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. USENIX Association, 689–703.

[10] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. 2015. The Next 700 BFT Protocols. *ACM Trans. Comput. Syst.* 32, 4 (2015), 12:1–12:45. https://doi.org/10.1145/2658994

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

[12] Gildas Avoine, Felix Gärtner, Rachid Guerraoui, and Marko Vukolić. 2005. Gracefully Degrading Fair Exchange with Security Modules. In *Dependable Computing - EDCC 5*, Mario Dal Cin, Mohamed Kaâniche, and András Pataricza (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 55–71.

[13] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on Steroids: SGX-Based High Performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys '17)*. Association for Computing Machinery, 222–237. https://doi.org/10.1145/3064176.3064213

[14] Alysson Bessani, João Sousa, and Eduardo E.P. Alchieri. 2014. State Machine Replication for the Masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–362. https://doi.org/10.1109/DSN.2014.43

[15] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (2002), 398–461. https://doi.org/10.1145/571637.571640

[16] 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 (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 221–234. https://doi.org/10.1145/3448016.3452823

[17] Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 645–658.

[18] Junchao Chen, Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. 2022. Power-of-Collaboration: A Sustainable Resilient Ledger Built Democratically. *IEEE Data Eng. Bull.* 45, 2 (2022), 25–36.

[19] Junchao Chen, Suyash Gupta, Alberto Sonnino, Lefteris Kokoris-Kogias, and Mohammad Sadoghi. 2023. Resilient Consensus Sustained Collaboratively. *CoRR* abs/2302.02325 (2023). https://doi.org/10.48550/arXiv.2302.02325 arXiv:2302.02325

[20] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 185–200. https://doi.org/10.1109/EuroSP.2019.00023

[21] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. 2007. Attested Append-Only Memory: Making Adversaries Stick to Their Word. *SIGOPS Oper. Syst. Rev.* 41, 6 (2007), 189–204. https://doi.org/10.1145/1323293.1294280

[22] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. 2012. On the (Limited) Power of Non-Equivocation. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*. Association for Computing Machinery, 301–308. https://doi.org/10.1145/2332432.2332490

[23] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright Cluster Services. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, 277–290. https://doi.org/10.1145/1629575.1629602

[24] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 153–168.

[25] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, 143–154. https://doi.org/10.1145/1807128.1807152

[26] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. https://ia.cr/2016/086.

[27] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 857–874.

[28] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. 2022. DAMYSUS: Streamlined BFT Consensus Leveraging Trusted Components. In *Proceedings of the Seventeenth European Conference on Computer Systems*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3492321.3519568

[29] 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, 1085–1100. https://doi.org/10.1145/3035918.3064033

[30] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* 54, 6, Article 126 (jul 2021). https://doi.org/10.1145/3456631

[31] 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 *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 568–580. https://doi.org/10.1109/DSN.2019.00063

[32] Franz Gregor, Wojciech Ozga, Sébastien Vaucher, Rafael Pires, Do Le Quoc, Sergei Arnautov, André Martin, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. 2020. Trust Management as a Service: Enabling Trusted Execution in the Face of Byzantine Stakeholders. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 502–514. https://doi.org/10.1109/DSN48063.2020.00063

[33] Trusted Computing Group. 2019. Trusted Platform Module Library. https://trustedcomputinggroup.org/resource/tpm-library-specification/

[34] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, Peter Robinson and Faith Ellen (Eds.). ACM, 307–316. https://doi.org/10.1145/3293611.3331589

[35] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2021. Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT*. OpenProceedings.org, 301–312. https://doi.org/10.5441/002/edbt.2021.27

[36] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. *Fault-Tolerant Distributed Transactions on Blockchain.* Morgan & Claypool Publishers. https://doi.org/10.2200/S01068ED1V01Y202012DTM065

[37] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. In *37th IEEE International Conference on Data Engineering, ICDE 2021*. IEEE, 1392–1403. https://doi.org/10.1109/ICDE51399.2021.00124

[38] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proc. VLDB Endow.* 13, 6 (2020), 868–883. https://doi.org/10.14778/3380750.3380757

[39] Suyash Gupta, Sajjad Rahnama, Erik Linsenmayer, Faisal Nawab, and Mohammad Sadoghi. 2023. Reliable Transactions in Serverless-Edge Architecture. In *39th IEEE International Conference on Data Engineering, ICDE 2023*. IEEE.

[40] Suyash Gupta, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Permissioned Blockchain Through the Looking Glass: Architectural and Implementation Lessons Learned. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020*. IEEE, 754–764. https://doi.org/10.1109/ICDCS47774.2020.00012

[41] Suyash Gupta and Mohammad Sadoghi. 2019. Blockchain Transaction Processing. In *Encyclopedia of Big Data Technologies*. Springer, 1–11. https://doi.org/10.1007/978-3-319-63962-8_333-1

[42] R. Harding, D. Van Aken, A. Pavlo, and M. Stonebraker. 2017. An Evaluation of Distributed Concurrency Control. *Proc. VLDB Endow.* 10, 5 (2017), 553–564. https://doi.org/10.14778/3055540.3055548

[43] Manuel Huber, Julian Horsch, and Sascha Wessel. 2017. Protecting Suspended Devices from Memory Attacks. In *Proceedings of the 10th European Workshop on Systems Security (EuroSec'17)*. Association for Computing Machinery, New York, NY, USA, Article 10, 6 pages. https://doi.org/10.1145/3065913.3065914

[44] Dakai Kang, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2023. Practical View-Change-Less Protocol through Rapid View Synchronization. *CoRR* abs/2302.02118 (2023). https://doi.org/10.48550/arXiv.2302.02118 arXiv:2302.02118

[45] Luyi Kang, Yuqi Xue, Weiwei Jia, Xiaohao Wang, Jongryool Kim, Changhwan Youn, Myeong Joon Kang, Hyung Jin Lim, Bruce L.

Jacob, and Jian Huang. 2021. IceClave: A Trusted Execution Environment for In-Storage Computing. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 199–211. https://doi.org/10.1145/3466752.3480109

[46] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: Resource-Efficient Byzantine Fault Tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems*. Association for Computing Machinery, 295–308. https://doi.org/10.1145/2168836.2168866

[47] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. 2012. All about Eve: Execute-Verify Replication for Multi-Core Servers. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. USENIX, 237–250.

[48] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2009. Zyzzyva: Speculative Byzantine Fault Tolerance. *ACM Trans. Comput. Syst.* 27, 4 (2009), 7:1–7:39. https://doi.org/10.1145/1658357.1658358

[49] Leslie Lamport. 2001. Paxos Made Simple. *ACM SIGACT News* 32, 4 (2001), 51–58. https://doi.org/10.1145/568425.568433 Distributed Computing Column 5.

[50] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 38. https://doi.org/10.1145/3342195.3387532

[51] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems. In *6th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association.

[52] Wenhao Li, Yubin Xia, and Haibo Chen. 2018. Research on ARM TrustZone. *GetMobile Mob. Comput. Commun.* 22, 3 (2018), 17–22. https://doi.org/10.1145/3308755.3308761

[53] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter R. Pietzuch. 2019. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*. ACM, 63–79. https://doi.org/10.1145/3341301.3359627

[54] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Christof Fetzer, and Peter R. Pietzuch. 2017. Glamdring: Automatic Application Partitioning for Intel SGX. In *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017*. USENIX Association, 285–298.

[55] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. 2016. XFT: Practical Fault Tolerance beyond Crashes. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, USA, 485–500.

[56] André Martin, Cong Lian, Franz Gregor, Robert Krahn, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. 2021. ADAM-CS: Advanced Asynchronous Monotonic Counter Service. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 426–437. https://doi.org/10.1109/DSN48987.2021.00053

[57] André Martin, Cong Lian, Franz Gregor, Robert Krahn, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. 2021. ADAM-CS: Advanced Asynchronous Monotonic Counter Service. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 426–437. https://doi.org/10.1109/DSN48987.2021.00053

[58] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *Proceedings of the 26th*

USENIX Conference on Security Symposium (SEC'17). USENIX Association, USA, 1289–1306.

[59] Jämes Ménétrey, Christian Göttel, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. 2022. An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments. *CoRR* abs/2204.06790 (2022). https://doi.org/10.48550/arXiv.2204.06790

[60] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[61] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. 2019. Erlay: Efficient Transaction Relay for Bitcoin. Association for Computing Machinery, New York, NY, USA, 817–831. https://doi.org/10.1145/3319535.3354237

[62] Faisal Nawab. 2021. WedgeChain: A Trusted Edge-Cloud Store With Asynchronous (Lazy) Trust. In *37th IEEE International Conference on Data Engineering, ICDE*. IEEE, 408–419. https://doi.org/10.1109/ICDE51399.2021.00042

[63] Faisal Nawab and Mohammad Sadoghi. 2019. Blockplane: A Global-Scale Byzantizing Middleware. In *35th International Conference on Data Engineering (ICDE)*. IEEE, 124–135. https://doi.org/10.1109/ICDE.2019.00020

[64] Faisal Nawab and Mohammad Sadoghi. 2023. Consensus in Data Management: From Distributed Commit to Blockchain. *Foundations and Trends® in Databases* 12, 4 (2023), 221–364. https://doi.org/10.1561/1900000075

[65] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. 2021. Kauri: Scalable BFT Consensus with Pipelined Tree-Based Dissemination and Aggregation. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 35–48. https://doi.org/10.1145/3477132.3483584

[66] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX, 305–320.

[67] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. 2017. Eleos: ExitLess OS Services for SGX Enclaves. In *Proceedings of the Twelfth European Conference on Computer Systems*. ACM, 238–253. https://doi.org/10.1145/3064176.3064219

[68] Bryan Parno, Jacob R. Lorch, John R. Douceur, James W. Mickens, and Jonathan M. McCune. 2011. Memoir: Practical State Continuity for Protected Modules. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 379–394. https://doi.org/10.1109/SP.2011.38

[69] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A. Sartakov, and Peter R. Pietzuch. 2019. SGX-LKL: Securing the Host OS Interface for Trusted Execution. *CoRR* abs/1908.11143 (2019). arXiv:1908.11143

[70] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. 264–278. https://doi.org/10.1109/SP.2018.00025

[71] Sajjad Rahnama, Suyash Gupta, Thamir Qadah, Jelle Hellings, and Mohammad Sadoghi. 2020. Scalable, Resilient and Configurable Permissioned Blockchain Fabric. *Proc. VLDB Endow.* 13, 12 (2020), 2893–2896. https://doi.org/10.14778/3415478.3415502

[72] Sajjad Rahnama, Suyash Gupta, Rohan Sogani, Dhruv Krishnan, and Mohammad Sadoghi. 2022. RingBFT: Resilient Consensus over Sharded Ring Topology. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022*. OpenProceedings.org, 2:298–2:311. https://doi.org/10.48786/edbt.2022.17

[73] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olya Ohrimenko, Felix Schuster, Roy Schwartz, Alex Shamis, Olga Vrousgou, and Christoph M. Wintersteiger. 2019. *CCF: A Framework for Building Confidential Verifiable Replicated Services*. Technical Report MSR-TR-2019-16. Microsoft. https://www.microsoft.com/en-us/research/publication/ccf-a-framework-for-building-confidential-verifiable-replicated-services/

[74] J.H. Saltzer and M.D. Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. https://doi.org/10.1109/PROC.1975.9939

[75] Vasily A. Sartakov, Stefan Brenner, Sonia Ben Mokhtar, Sara Bouchenak, Gaël Thomas, and Rüdiger Kapitza. 2018. EActors: Fast and flexible trusted computing using SGX. In *Proceedings of the 19th International Middleware Conference*, Paulo Ferreira and Liuba Shrira (Eds.). ACM, 187–200. https://doi.org/10.1145/3274808.3274823

[76] Vasily A. Sartakov, Daniel O'Keeffe, David M. Eyers, Lluís Vilanova, and Peter R. Pietzuch. 2021. Spons & Shields: practical isolation for trusted execution. In *VEE '21: 17th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Virtual USA, April 16, 2021*. ACM, 186–200. https://doi.org/10.1145/3453933.3454024

[77] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (dec 1990), 299–319. https://doi.org/10.1145/98163.98167

[78] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cedric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, Julien Maffre, Olga Vrousgou, Christoph M. Wintersteiger, Manuel Costa, and Mark Russinovich. 2022. IA-CCF: Individual Accountability for Permissioned Ledgers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 467–491.

[79] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. *Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX*. Association for Computing Machinery, New York, NY, USA, 955–970.

[80] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. 2022. State Machine Replication Scalability Made Simple. In *Proceedings of the Seventeenth European Conference on Computer Systems*. Association for Computing Machinery, New York, NY, USA, 17–33. https://doi.org/10.1145/3492321.3519579

[81] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. 2021. Basil: Breaking up BFT with ACID (Transactions). In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3477132.3483552

[82] Jörg Thalheim, Harshavardhan Unnibhavi, Christian Priebe, Pramod Bhatotia, and Peter R. Pietzuch. 2021. rkt-io: a direct I/O stack for shielded execution. In *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*. ACM, 490–506. https://doi.org/10.1145/3447786.3456255

[83] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. 2013. Efficient Byzantine Fault-Tolerance. *IEEE Trans. Comput.* 62, 1 (2013), 16–30. https://doi.org/10.1109/TC.2011.221

[84] Wenbin Wang, Chaoshu Yang, Runyu Zhang, Shun Nie, Xianzhang Chen, and Duo Liu. 2020. Themis: Malicious Wear Detection and Defense for Persistent Memory File Systems. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. 140–147. https://doi.org/10.1109/ICPADS51040.2020.00028

[85] Gavin Wood. 2015. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf

[86] Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael Reiter. 2021. Brief Announcement: Communication-Efficient BFT Using Small Trusted Hardware to Tolerate Minority Corruption. In *35th International Symposium on Distributed Computing (DISC 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 209)*. Schloss

Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 62:1–62:4. https://doi.org/10.4230/LIPIcs.DISC.2021.62

[87] 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 ACM Symposium on Principles of Distributed Computing*. ACM, 347–356. https://doi.org/10.1145/3293611.3331591

[88] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. 2020. OHIE: Blockchain Scaling Made Simple. In *2020 IEEE Symposium on Security and Privacy*. IEEE, 90–105. https://doi.org/10.1109/SP40000.2020.00008

[89] Rui Yuan, Yubin Xia, Haibo Chen, Binyu Zang, and Jan Xie. 2018. ShadowEth: Private Smart Contract on Public Blockchain. *J. Comput. Sci. Technol.* 33, 3 (2018), 542–556. https://doi.org/10.1007/s11390-018-1839-y

[90] Gengrui Zhang, Fei Pan, Michael Dang'ana, Yunhao Mao, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. 2022. Reaching Consensus in the Byzantine Empire: A Comprehensive Review of BFT Consensus Algorithms. *CoRR* abs/2204.03181 (2022). https://doi.org/10.48550/arXiv.2204.03181 arXiv:2204.03181

A Artifact Appendix

This paper argues that existing TRUST-BFT protocols have targeted the wrong metric: while reducing the replication factor to $2f + 1$ may seem appealing from a resource efficiency or management overhead standpoint, it, paradoxically, comes at a significant performance cost. Nonetheless, trusted components can still bring huge benefits to BFT consensus *when they use* $3f + 1$ *replicas*. We propose a novel suite of consensus algorithms (**Flexi**ble **Trust**ed BFT (FlexiTrust)), which address the aforementioned limitations. These protocols are always responsive and achieve high throughput as they (1) make minimal use of trusted components (once per client operation and at the primary replica only), and (2) support parallel consensus invocations. Both these properties are made possible by the ability to use large quorums (of size $2f + 1$) when using $3f + 1$ replicas.

Our techniques can be used to convert any TRUST-BFT protocol into a FlexiTrust protocol. We provide as examples two such conversions: Flexi-BFT and Flexi-ZZ, two protocols based on MinBFT [83] and Zyzzyva [48], respectively.

We evaluate our FlexiTrust variants against *five* protocols: *three* TRUST-BFT and *two* BFT systems. For fair evaluation, we implement these protocols on the open-source ResilientDB fabric [38]. Our evaluation on a real-world setup of 97 replicas and up to 80 k clients shows that our FlexiTrust protocols achieve up to 185% and 100% more throughput than their TRUST-BFT and BFT counterparts, respectively. Further, we show that our FlexiTrust protocols continue outperforming these protocols even when replicas are distributed across *six* locations in four continents.

A.1 Abstract

The aim of this artifact is to demonstrate the performance of our FlexiTrust protocols and evaluate them against their BFT and TRUST-BFT counterparts. This artifact will help to prove two of our key claims: (1) Existing TRUST-BFT protocols, despite needing only $2f + 1$ replicas, yield lower throughputs and higher latencies than their BFT counterparts. (2) The use of trusted components, such as Intel SGX, can help to design efficient versions of existing BFT protocols.

As described in the paper, we provide access to two protocols from our suite of FlexiTrust protocols: Flexi-BFT and Flexi-ZZ. We evaluate these protocols against *six* other protocols: Pbft [15], Zyzzyva [48], Pbft-EA [21], MinBFT [83], MinZZ [83], and Opbft-ea, a variation of Pbft-EA we develop that supports parallel consensus invocations.

A.2 Description & Requirements

To recreate the same experimental setup as used in the paper, we provide access to necessary binaries, scripts, and the complete codebase. This has been open-sourced at https://doi.org/10.5281/zenodo.7734495, and is freely available for anyone to access or download. In the git repository, we provide a **README** file that includes step-by-step instructions to install and run the experiments.

Note. As we ran all our experiments on Oracle Cloud Infrastructure (OCI), to facilitate artifact evaluation, we also provide two *images*, which reviewers can use if they have access to OCI. These images reduce the installation and setup time to zero as we have pre-built all the dependencies and all the scripts are pre-loaded. The steps to use these images are provided in the README.

A.2.1 How to access. To access to artifact, please follow the following open-sourced link: https://doi.org/10.5281/zenodo.7734495.

A.2.2 Hardware dependencies. None.

A.2.3 Software dependencies. Following software dependencies need to be installed prior to running any experiment: (1) Intel SGX, (2) Boost, (3) Crypto++, (4) Jemalloc, (5) NNG, and (6) SQLite. Steps to download and install, Intel SGX dependencies are provided in the README. For other dependencies, we have packaged them into a folder **deps** and they need to be simply untarred.

A.2.4 Benchmarks. None.

A.3 Set-up

Please follow the steps stated in the README file.

A.4 Evaluation workflow

In our paper, we run a variety of experiments that help to gauge the performance of our FlexiTrust protocols against existing TRUST-BFT and BFT protocols.

A.4.1 Major Claims. Following are the major claims of our paper:

- *(C1): How do our FLEXITRUST protocol perform and scale?*
  In our paper, we run a series of experiments to illustrate that our FLEXITRUST protocols outperform existing TRUST-BFT and BFT protocols. We do this in Sections 9.4 to 9.5. For example, Figures 6(i) shows that our FLEXI-BFT and FLEXI-ZZ protocols instead achieve up to 22% and 58% higher throughput than PBFT, (and up to 87% and 77% higher throughput over MINBFT and MINZZ).
- *(C2): What is the impact of failures?* The aim of this experiment is to illustrate that despite failures, our TRUST-BFT and BFT protocols continue yielding high throughput and low latencies under a single failure, while protocols like MINZZ and ZYZZYVA observe a drop in their throughputs. We illustrate this in Section 9.8 of the paper.

A.4.2 Experiments. To certify our claims, the reviewers can run experiments that help to re-plot graphs in Figures 6(i) to (iii) and Figure 7. Each figure would require around 3.5–4 hours for setup, compile, and deployment, while it would require 2.5-3 hours to execute. We provide in the repo explicit configuration parameters for each protocol. Further, we have provided a directory that lists down configuration parameters to run each experiment.

A.5 General Notes

While running experiments for the paper, we averaged runs over at least *three* runs. We have often observed that in large-scale deployments, like ours where there can be up to 60 replicas, the results vary over runs. Hence, we had to tune up different parameters to find the peak throughput in each case. We would be happy to answer any queries regarding this artifact evaluation phase.