A Local-First Approach for Green Smart Contracts

Stokkink, Q.A.; Pouwelse, J.A.

DOI
10.1145/3607196

Publication date
2024

Document Version
Final published version

Published in
Distributed Ledger Technologies: Research and Practice

A Local-First Approach for Green Smart Contracts

QUINTEN STOKKINK and JOHAN POUWELSE, Delft University of Technology, The Netherlands

Shared code in blockchains, known as *smart contracts*, stands to replace important parts of our digital governance and financial infrastructure. The permissionless execution of smart contracts is tightly coupled to cryptocurrencies and Proof-of-Work blockchains. As a result, smart contracts inherit the environmental impact of Proof-of-Work blockchains, such as its energy consumption, carbon footprint, and electronic waste. The four concepts of relaxed consistency, strong identities, probabilistic consensus, and the use of liabilities instead of assets may change the status quo. This work explores the integration of these concepts to decouple smart contracts from Proof-of-Work blockchains. By means of a local-first approach, which may expose users to inconsistent ephemeral contract states, the architecture of smart contracts can be transformed to become green. Because such contract states may be dropped, we base the interactions between users on liabilities. We propose a novel paradigm for smart contract architectures, named Green Smart Contracts, that is based on a local-first approach. Furthermore, we present and implement a prototype solution for this paradigm. We validate the need for a mechanism to resolve consistency violations by replaying the contract calls of a real smart contract. Our simulation shows that violations occur more often (13% of contract invocations) when using liabilities than when using a traditional blockchain (3% of contract invocations). However, we additionally validate that they can be avoided using a consensus mechanism, and our experiments show that a publish-subscribe messaging pattern uses the fewest messages to do so, though it may not be applicable for use cases that disallow the inherent imbalance in the messaging between peers. Our carbon emission estimation shows that a Green Smart Contract approach lowers carbon emissions by 52.31% when compared with the messaging behavior of a typical peer-to-peer blockchain with 1000 nodes.

CCS Concepts: • **Computer systems organization → Peer-to-peer architectures;**

Additional Key Words and Phrases: Smart contract, local-first, green, Web3, blockchain, execution

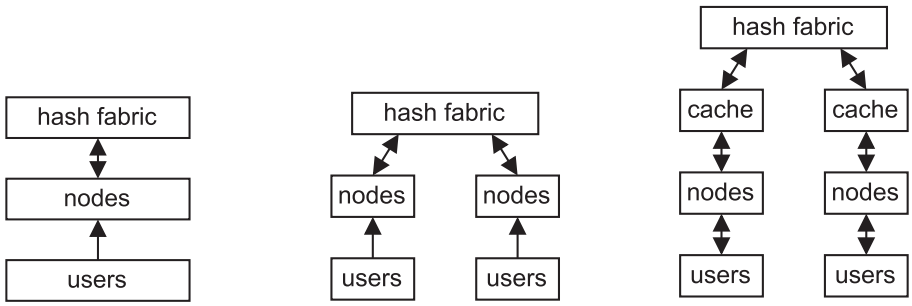
ACM Reference format:
Quinten Stokkink and Johan Pouwelse. 2024. A Local-First Approach for Green Smart Contracts. *Distrib. Ledger Technol.* 3, 2, Article 13 (June 2024), 21 pages.
https://doi.org/10.1145/3607196

1 INTRODUCTION

By 2030, it is expected that digitization and automation will make 80% of current financial firms "go out of business, become commoditized or exist only formally but not competing effectively" [23]. Executing the newly emerging digitized and automated processes will require multiple servers that are governed by different entities that have mutual distrust (e.g., due to geopolitics). Therefore, the financial sector is looking into Web3 developments such as blockchains and smart contracts to modernize its infrastructure [19]. Smart contracts, which consist of program code that is published in a blockchain and seek to provide permissionless execution, transparency, and availability of source code, can serve to meet the demands of new digitized governance and financial infrastructure [8, 11, 14, 18, 26, 50, 59, 65]. Some smart contracts are even already seeing up to 20 000 invocations

Authors' address: Q. Stokkink and J. Pouwelse, Delft University of Technology, Mekelweg 5, Delft, Zuid-Holland, The Netherlands, 2628 CD; emails: {q.a.stokkink, j.a.pouwelse}@tudelft.nl.

2769-6472/2024/06-ART13 $15.00
https://doi.org/10.1145/3607196



(a) Single user pool; single node pool; single shared hash fabric.

(b) Multiple user pools; multiple node pools; multiple hash fabric branches.

(c) Multiple user pools; multiple node pools; multiple hash fabric branches; multiple ephemeral hash fabric states (caches).

Fig. 1. System models for information flows between users' smart contract invocations and the underlying hash-based data structures: from traditional proposals (a) to recent proposals (b) and local-first (c).

per day [48]. Unfortunately, the blockchains that power smart contracts are not environmentally friendly [57]. The Chinese government has even banned cryptocurrencies due to their utilized resources, energy consumption, carbon footprint, and electronic waste [70]. This work seeks to integrate and leverage recent proposals to realize a paradigm shift in the architecture of smart contract systems and, thereby, provide a greener alternative for future digitization and automation.

Our work proposes a shift away from the traditional models used by, for example, Bitcoin [43] and Ethereum [71], in which all users are able to *observe and verify* that the program code of a smart contract is executed correctly while collaboratively appending to the same hash fabric (e.g., a blockchain). In the traditional system model, users attempt to invoke smart contract code and nodes attempt to add these invocations to a blockchain in return for payment. When added to a blockchain, the code can be observed and executed on all nodes. Thereby, every node in such a system can calculate the state of a smart contract based on the history of invocations. As all nodes are expected to execute all invocations of the smart contract code, the responsibility for its correct execution does not depend on a single party but rather on the entire network. Network-wide verification allows smart contracts to achieve fairness and cooperation among competitors in a trustless context [20]. In short, the system model of the traditional approach to smart contracts, shown in Figure 1(a), consists of a pool of users that invoke smart contracts and a pool of nodes that attempt to add these invocations to a shared hash fabric, consisting of a blockchain.

Several concepts have been proposed for the blockchain ecosystem that could be integrated to realize a paradigm shift. Generally, the metric of "throughput" (i.e., the number of contract invocations per time unit for smart contracts) is optimized and with it comes lesser communication requirements and lesser environmental impact [3]. A throughput increase is often achieved through the concepts of weaker consistency models (to capture how invocations are recorded) and weaker consensus models (the manner in which invocations are agreed upon), and typically these approaches no longer use a traditional single chain of blocks but rather a hash fabric of blocks. For example, "sharding" uses cliques of nodes, known as "shards", that communicate intensively internally but very little between each other [64], and Hashgraph opts for a Directed Acyclic Graph instead of a single chain [6]. As a result, the system models of recent proposals, shown in Figure 1(b), have to consider multiple pools of nodes that attempt to synchronize with a shared hash fabric. However, users do not actively take part in this synchronization.

Smart contracts are not necessarily compatible with weaker consistency and consensus models. Depending on how a consistency model is weakened, the safety and liveness guarantees of a system may radically change.

Users and nodes may not be able to observe (all) interactions with smart contracts, they may not be able to observe interactions in the right order, and perhaps they may only be able to observe interactions after a long delay. Furthermore, weaker consensus may lead to temporary decisions on allowable inputs and, by extension, a change in the values of the outputs of smart contracts [36], which must be retroactively corrected.

A next step, away from network-wide verification, is a local-first approach to the smart contract execution model. The approach proposes to make sharing interactions with other nodes a secondary concern [33]. The corresponding system model, shown in Figure 1(c), requires users and nodes to actively engage in change management by merging the changes in their hash-fabric caches to the global shared hash fabric. The local-first approach was originally proposed for human users that collaboratively edit a shared data structure but produce very little consistency conflicts [33]. Thereby, local-first is a green approach to smart contract execution, as it eliminates the need for Proof-of-Work and minimizes the required communication of data. However, when integrated into existing blockchain systems, a local-first approach would still need its invocations paid for using the cryptocurrencies of Proof-of-Work blockchains. Otherwise, users would not be incentivized to store invocations so they can be verified by other users. Furthermore, like a sharding approach, a local-first approach depends heavily on the identities of nodes to edit caches, whereas blockchains typically do not have identity management beyond public keys.

This work enables a local-first approach by challenging the need for cryptocurrencies and arguing that identities can be used in a permissionless smart contract system. Firstly, our insight is that cryptocurrencies represent assets and, therefore, require network-wide verification. By changing from assets (i.e., what is owned) to the concept of liabilities (i.e., what is owed), network-wide verification is no longer required before smart contract execution. Secondly, our insight is that modern identity management no longer requires central governance. Identities can be stronger than just public keys: strong identities do not necessarily violate smart contracts' promise of a trustless context that enables fairness and cooperation. However, due to the application of these concepts, green architectures are necessarily subject to new design constraints. In this work, we derive these constraints for systems and their architectures in order to make use of novel green concepts. Thereby, these resulting novel systems can be applied to meet all of the functional requirements of smart contracts to modernize governance and financial infrastructure without the operational risk of being banned by governments due to environmental concerns.

This work defines a novel paradigm for smart contract execution called Green Smart Contracts (GSCs), which is based on concepts observed in proposals for the Web3 ecosystem and executes smart contracts using a local-first approach. The overall contribution of this work is the definition of **a novel architectural paradigm for local-first smart contract execution**. We envision our paradigm challenging Proof-of-Work blockchains as the de facto standard for execution of smart contracts, which currently require network-wide verification for each contract invocation. Our aim is to allow both system and application designers to leverage a greener alternative for contract execution. Our work offers the following contributions:

- **Contribution 1:** We discuss how the concepts of relaxed consistency, strong identities, probabilistic consensus, and liabilities allow for greener smart contracts (Section 2).
- **Contribution 2:** We derive the constraints for system architectures to leverage the four concepts for greener smart contracts (Section 3).
- **Contribution 3:** We design a greener smart contract prototype solution (Section 4).
- **Contribution 4:** Our simulation shows the necessity of consensus for a widely used real smart contract and the amount of time that nodes work with a cache that is inconsistent with the global hash fabric (Section 5).
- **Contribution 5:** Through further experiments, we show that inconsistencies in the hash fabric can be effectively resolved for multiple use cases (Section 6).
- **Contribution 6:** Based on our results, we create a model for CO_2 emissions that shows when our approach becomes a greener alternative to existing solutions (Section 6).

The remainder of this work is structured as follows. In Section 2, we identify the main problem of smart contract execution and we discuss four concepts that are used in the design of our proposed architectural paradigm. In Section 3, we give the design constraints that define our model. In Section 4, we discuss the implementation of our paradigm (Green Smart Contracts). In Section 5, we validate the need for careful configuration of our paradigm. In Section 6, we experiment with different configurations to show that our model accurately executes smart contracts and what the consequences of the configurations are both in terms of messages and their resulting CO_2 emission estimates. In Section 7, we discuss related work that did not get discussed in the preliminaries (Section 2). Lastly, in Section 8, we give the conclusion of this work.

2 PROBLEM STATEMENT AND PRELIMINARIES

The problem that smart contracts face is that they inherit a large ecological footprint by being tied to Proof-of-Work blockchains. We look for an execution model that can make use of recent trends in the blockchain ecosystem to address this problem. We present four concepts that can be leveraged for greener smart contracts, and we explain how they can be leveraged.

Concept 1: Relaxation of the consistency model. Weakening the consistency model will improve throughput of a system. In practice, this is why systems that weaken their consistency model to allow for independent block updates (like Directed Acyclic Graphs) have shown higher transaction throughput [5, 35]. However, network-wide consistency cannot be weakened to the point of being eliminated. The order of executed operations matters for smart contracts [41, 66]: users' reads and writes to smart contracts are interdependent. Thereby, completely forfeiting consistency enables front-running attacks through information hiding [21]. Nevertheless, basing the consistency model on application-defined locality is a winning strategy [13, 17], especially if a small group of nodes has additional influence over a data structure's contents [40]. Exploring the application of a weaker form of consistency for smart contracts remains promising.

Examples of solutions that leverage weak consistency are Avalanche [53], Hashgraph [6], and the Tangle [49]. The commonality between these solutions is that they use Directed Acyclic Graphs that see their transactions interlinked based on some form of locality. Locality is based on "transactions" for Avalanche, "actors" for Hashgraph, and "sites" for the Tangle. We propose taking the concept of relaxing consistency based on locality to the extreme and investigate local-first consistency.

Concept 2: Using strong identities to detect forks. Individual countries and the European Union are creating passport-level identity solutions for use in the blockchain ecosystem, known as Self-Sovereign Identity solutions [4, 62, 63]. If strong identities were used for smart contracts, there would be no way for users to interfere with contract operations (i.e., "writes") of other users. However, despite a lack of writing contention, even when grounded in natural persons, identity does not guarantee validity. Secondly, a strong identity does not imply any special permissions (like in a permissioned blockchain). A strong identity is not necessarily a trusted identity. A well-identified user may still produce a conflict with itself (e.g., to fool other users and through bugs), which is the classic blockchain forking problem and still needs consensus. Nevertheless, strong identities and public key infrastructure have been shown to greatly improve the detection efficiency of information that users attempted to hide [34].

Examples of solutions that leverage strong identities are Corda [27], eBay, and Uber. Corda proposes ownership of contract applications based on public keys and the usage of identity to assign legal weight to documents on its ledger. Identity is leveraged both for privacy (not all nodes have to know of all transactions) and efficiency (not all nodes need to process all transactions). eBay and Uber have a centrally governed platform that provides strong user identities, which are a legal requirement, vital for employee management (Uber), and required for the selling of goods (eBay) and services (Uber) between their respective users. In short, for eBay and Uber strong identity is used for accountability. We investigate the application of strong identities to gain both efficiency and accountability without central governance.

Concept 3: Using probabilistic consensus for smart contract invocations. A probabilistic approach achieves high throughput and hardens its probabilistic guarantees over time [36]. Using a small-world assumption, detection of violations of agreements made through probabilistic consensus becomes more efficient with witness and audit protocols [10, 25]. Just like for the consistency model, the highest throughput is achieved when the *locality* of the data that consensus is formed over is close to the nodes that are selected to form consensus. Practically, the consensus mechanism should be tightly coupled to the consistency mechanism. For instance, there is a large increase in throughput for smart contracts if locality of Ethereum is optimized, known as "sharding" [64].

Examples of solutions that leverage probabilistic consensus are Avalanche [53], Bitcoin [43], and Ethereum [71]. One of the key innovations of Bitcoin is its probabilistic consensus, known as Proof-of-Work, requiring no communication between nodes in order to reach consensus (simply deciding on the longest known chain). Ethereum's "sharding" further leverages locality and adopts a model of communication between cliques of nodes. Avalanche even proposes "metastable consensus", relying on majority votes in overlapping localities. We note that probabilistic consensus, especially when exploiting locality, is mostly focused on making *any decision*, which is not necessarily the *best decision* for some use case or application. Therefore, we enable the use of these probabilistic consensus mechanisms but we do not pick a single mechanism in order to remain use case agnostic.

Concept 4: Postponing consensus using liabilities. Liabilities can materialize as tokens in contracts and can represent many different things, such as assets, trades, and loans, to support a token economy [69]. An example of a liability is payment through credit, where a user clears a transaction regardless of the user's balance, versus payment through debit, where the user must have the required sum beforehand. In other words, liabilities support systems that depend on authorization instead of ownership. However, even though liabilities can represent assets, to avoid double-spending of assets (like currency) a system requires some form of network consensus. Nevertheless, consensus is then only required after executing the smart contract and it can be postponed (or possibly even avoided) for use cases that depend on liabilities.

Liabilities stand to change the nature of a token economy when used in combination with the concepts of probabilistic consensus and weak consistency. Instead of serving one network-wide token (e.g., a token that serves a cryptocurrency), a unique token can be used for the locality of a contract that does not require the whole network to verify it. Thereby, every contract would have its own exchange rate against currency from other contracts (akin to exchanges between fiat currencies). Therefore, the vision of a general token economy, or digital currency, is transformed into one that exists only in the multiple localities of nodes: token microeconomies.

Examples of solutions to leverage tokens are Ethereum Request for Comment 20 (ERC-20) tokens [68], non-fungible tokens (NFTs) [39], and Basic Attention Tokens [38]. The ERC-20 standard exists to capture fungible tokens and the NFT standard was made for non-fungible tokens (i.e., assets). Both standards operate from smart contracts on the Ethereum blockchain and tokens that derive from them can represent the breadth of use cases for digital liability and asset management. For example, the Basic Attention Token is an ERC-20 token that uses a user's ad viewing time as the basis for its value. Instead of treating tokens as something to be implemented in a smart contract, we investigate the treatment of tokens as the primitive to power smart contracts.

Combining concepts. Any combination of the four concepts that we have previously presented can be leveraged to adapt a hash fabric to a given application domain. However, these concepts can also be viewed as the evolution toward individual accountability and decentralization of governance to replace network-wide verification. When viewed in this manner, shown in Figure 2, we see the communication costs of solutions lessen as individual accountability is increased. As communication costs are tied to the environmental impact of blockchains [3], it follows that the concept of liabilities is the next logical step to investigate, which we do in this work.

The four concepts we present are highly reminiscent of digitized governance and financial infrastructure in the physical world and, therefore, are applicable to digitizing these processes. For example, a person may use one's physical credit card (an identity) to withdraw physical money (a liability) at a physical bank (a node). Of course, to limit risk, the bank will disallow the person to withdraw more money than the individual's credit card limit allows. If the transaction succeeds, banks will engage in clearing and settlement with each other (logic

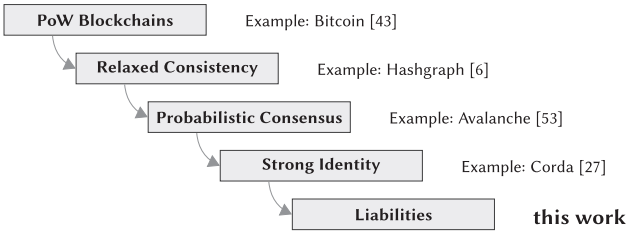


Fig. 2. The addition of concepts to Proof-of-Work blockchains and examples of solutions that implement all concepts up to that point.

Table 1. Differences Between GSCs and Smart Contracts that Use Traditional (PoW) Blockchains

Property	Smart Contracts	GSC
incentive	payment	reciprocity
fork detection	majority	probabilistic
impartiality	majority	randomness
use case	assets	liabilities
finality	probabilistic consensus	configurable

that needs consensus) and record the overall exchange of money between banks (consistency) in their ledgers. Another example is the casting of votes: a person may show one's passport (an identity) to cast a vote (a liability) at a government office (a node). The different offices then check whether duplicate votes have been cast (logic that needs consensus) and tally the votes (consistency) to record election results.

3 PROPOSED MODEL

We now determine to what extent the concepts presented in Section 2 can be applied and to what extent their application changes the smart contract execution model. In order for a system to integrate these concepts into a local-first system model (Figure 1), we derive the design constraints that we use to create a prototype solution. The findings are summarized in Table 1.

Any system that executes smart contracts requires incentive compatibility. In general, no rational user performs more work than necessary. Inherently, a decentralized system such as a smart contract system is prone to freeriding, which must be alleviated by implementing either a payment or a reciprocity scheme [22]. Traditional blockchains opt for the former, requiring payment for both proposing—and interacting with—a contract. However, the payment approach provides an unfortunate link between contract execution and payment through cryptocurrency, coupling the low throughput of cryptocurrencies to contracts. Therefore, the dependency on payments can only be broken by using reciprocity, leading to *Constraint 1: Green smart contract execution and dissemination should only depend on reciprocity between users, not on payment.*

A fork detection mechanism requires scalability with respect to the number of users. Smart contracts may have a large volume of interactions, e.g., up to 20 000 interactions per day have been observed [48]. Of course, multiple contract invocations fit inside a block but Proof-of-Work blockchains both have a limited number of blocks per day (e.g., Bitcoin roughly sees one block per 10 minutes) and require the majority of the network to observe and accept each newly proposed block. To this end, to overcome the limitations of Proof-of-Work, relaxations to the consistency and consensus models of blockchains have been proposed (Section 2). However, these relaxations change the nature of the smart contract invocations, which can no longer be assumed to be finalized through consensus but should be assumed to be tentative and part of an ephemeral state. Therefore, in

order to leverage these novel probabilistic approaches, our second constraint is formulated as *Constraint 2: Green smart contract execution should build on probabilistic fork detection, not consensus.*

A green smart contract system should ensure that the processing of invocations remains impartial to their contents. Traditionally, blockchains assume that the majority of nodes in a network is sufficient to quell any individual nodes that are partial to the contents of invocations (e.g., incentivizing the acceptance of valid invocations with a block-mining bounty in Bitcoin [43]). However, issues with impartiality may arise even when the majority of nodes in a network is used [59]. Furthermore, depending on the explicit involvement of a majority of nodes conflicts with Constraint 2. Instead, more recent proposals trust in the random selection of nodes to ensure impartiality. For example, Verifiable Random Functions have been proposed to elect verifiably random quorums [42] and randomly selected nodes (witnesses) can be used for fault detection [25]. For invocation processing to remain impartial, without violating Constraint 2, we impose *Constraint 3: Green smart contracts should use a random selection of nodes to ensure impartiality, not a majority.*

Greener smart contract systems can be decoupled from cryptocurrencies. Cryptocurrencies inherently require a system for asset management. To solve "double-spending" of the assets (i.e., transferring ownership of a single asset to more than one user), a form of consensus needs to be used. To avoid assets in their entirety, a liability-based interaction model (Concept 4) can be used to power smart contracts. By switching to liabilities as the underlying primitive, the system model of smart contracts (Figure 1(a)) necessarily changes into that of a local-first approach (Figure 1(c)). Asset management now becomes a part of the application layer instead of the underlying substrate for contracts. We explore the resulting solution space by imposing *Constraint 4: Green smart contract execution should be based on liabilities, not on assets.*

A greener smart contract system can exploit a relaxed consistency model. For example, even Bitcoin uses a gossip network to share contract invocations (and blockchain blocks in general) between nodes and later forms consensus on the finality of transactions based on the longest chain of blocks [43]. By explicitly separating the consistency mechanism from the consensus mechanism, recent works have shown the benefits of immediate availability of data in the system, fewer exchanged messages, and a higher throughput [1, 56]. However, the downside is that the invocations of users may not be deemed valid on the application layer at a later time and they may be rolled back. For Bitcoin, the probabilistic finality of transactions of six blocks (about one hour) [53] is sufficient for digital currency. However, the finality requirement may change depending on the application layer. For example, the Corda white paper [27] argues that a PDF document is binding, even if it is not even on a blockchain, as long as it was signed by an authority. Therefore, in order to remain application agnostic, we postulate the following *Constraint 5: Green smart contracts should allow for configuration of their consensus mechanism, not provide a single fixed mechanism.*

4 IMPLEMENTATION

A greener architecture design that makes use of the concepts that we have presented is different from traditional blockchain architectures. We present the architecture of our GSC prototype, which satisfies the design constraints presented in Section 3. Our prototype architecture has five components: (1) a *hash fabric* storing smart contracts and their operations as an immutable history, (2) a *consensus* component to detect and resolve forks, (3) a *runtime* for users to interact with contracts, (4) a *virtual machine* that runs the code and operations captured in the hash fabric on every node, and (5) a networking protocol for *contract discovery*. We now explain how these components interact and we discuss their necessity, with Figure 3 as a visual reference.

The hash fabric is the central component of GSC architectures, persisting (i.e., both storing and sharing between users) the content and operations of smart contracts (that run in the virtual machine). The hash fabric replaces what would traditionally necessarily be a single "main" (block)chain. For instance, traditionally, in Ethereum all executed user code ends up in one chain. In contrast, due to weaker consistency and consensus models, the hash fabric can represent a data structure that is no longer a single chain. For example, next to a

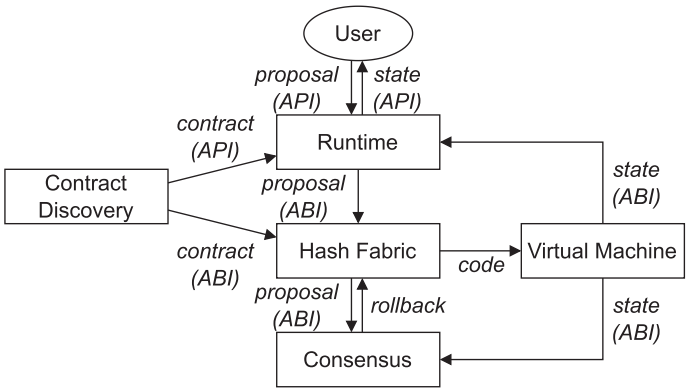


Fig. 3. Main components of GSC architectures and their interactions.

traditional single chain, the hash fabric may also use a mesh or a Distributed Acyclic Graph. Nevertheless, the data structure that is used by the hash fabric should form an immutable history.

As users interact with GSC architectures through an interface, they generate so-called *proposals* to modify the underlying data structure. These proposals capture and hide the semantics of the smart contract information in the hash fabric. Proposals are shared between users using a consistency mechanism of a hash fabric. The primary function of the consistency mechanism is to synchronize new proposals with the network and to apply received proposals. Secondarily, the consistency mechanism may also be forced to change the data structure itself, a rollback that invalidates proposals in the hash fabric in case of consistency violations. In short, proposals lead to the temporary—and possibly inconsistent—states that are typical for a local-first approach.

Our prototype defines simple push and pull gossip messages to share proposals in network overlays. Each proposal contains fields for its *code*, the *consensus round* it belongs to, its *proposal type* (i.e., contract creation, operation, or rollback), its *base address* in the virtual machine, and its *block number* in the hash fabric. Consensus rounds are necessary when the hash fabric needs to decide between inconsistent proposals using a consensus mechanism, which we describe shortly.

Figure 4 shows an example of how our prototype transforms proposals, received through a network overlay, into blocks in its hash fabric. The first block contains the contract creation proposal, followed by interactions with the contract. This first proposal (*Proposal 1*) places its contract code at address `0x00000000` in the virtual machine component, claiming block number 1 and participating in consensus round 0. Proposals are rejected if they are proposed for a consensus round that has already finished. An arbitrary consensus mechanism may decide to accept or reject blocks that are proposed in a certain consensus round. Of course, one of the blocks necessarily has to be rejected if the order of applying the proposals (*Proposal 2* and *Proposal 3*) leads to different states. However, a new ordering may be chosen in a consensus round, such as *Proposal 3* being placed in a new *Block 3* in round 1.

The consensus mechanism is tasked with evaluating the structure and semantics of the hash fabric to select the *dominant* history from the valid histories captured by the hash fabric. For example, one may select the "longest chain" as the dominant history (known as *Nakamoto consensus* [72]) from multiple forks of valid blocks in a blockchain. Though forming consensus on an entire history is certainly possible (e.g., in a relational database or a simple log of executions), it is more efficient to only form consensus on new entries in an append-only log as the data structure grows. Within the scope of this work, probabilistic consensus is considered (Constraint 2), which may lead to multiple conflicting histories. The consensus mechanism determines the currently valid proposals and their corresponding causal history (i.e., the "head of the chain" in traditional blockchains).

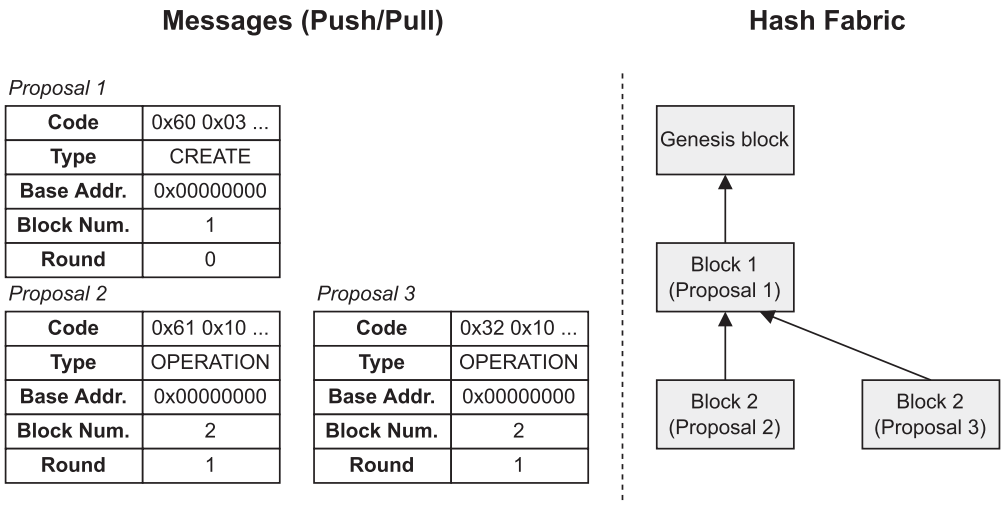


Fig. 4. How proposals (push/pull gossip messages) are captured in the hash fabric.

Not all contract interactions require consensus (e.g., inspecting the value of a variable in a smart contract's state). What interactions do need consensus can be derived through the hash fabric and the virtual machine. Firstly, in the case that the hash fabric fails to apply a proposal (e.g., when two proposals define the same block number when using a single chain) consensus is needed. Secondly, in the case that the virtual machine fails to apply a proposal (e.g., when two proposals write a different value to the same memory address) consensus is once again required. In both cases, consensus is used to select a single dominant history.

What proposals a user is required to store depends on the chosen consensus mechanism. However, what users end up storing also depends on the trust between users. For example, in theory, Bitcoin requires that all nodes wishing to add blocks to a blockchain store the entire longest chain. In practice, "light nodes" may store only a subset of all blocks and trust in other nodes that store the entire chain [24]. Within the scope of this work, we acknowledge that more efficient storage schemes exist that exploit reciprocity and randomness (Constraints 1 and 3) and trust, for example, Timely Sharing with Reputation Prototype [61], and we assume that they are leveraged by users to obtain the necessary data to form consensus.

A runtime is needed for users to create contract code. This is a common approach, for example, found in Bitcoin and Ethereum [73], to make contracts version independent and portable. The hash fabric only provides the history of operations on the contract. From its history, the runtime derives the current *human-readable state* of a contract to present to the user. The state is calculated by applying the application programming interface (API) abstractions to the result of executing the operations captured in the dominant history of the hash fabric.

The GSC architecture compiles contract code written in a Domain-Specific Language. In our prototype, contracts are written in the Solidity language. The creation of a new contract causes two proposals, one for the hash fabric's language API and one for its compiled equivalent, the application binary interface (ABI). We make the distinction between the source contract and the actual compiled contract, as the high-level language implementation may not produce the same compiled code for different virtual machines. However, the API is still practically necessary for human interaction, as the ABI exposes users to low-level details that are difficult to work with [60].

A virtual machine optimistically locally executes all compiled (ABI) code that is received through a user's network. Essentially, this is no different from how blockchain solutions normally offer their transactions to their respective virtual machines [44, 64]. However, in contrast to normal execution of virtual machine instructions, GSC systems explicitly maintain the state of all forks of the hash fabric (Constraint 2). In our prototype implementation, we use the Ethereum Virtual Machine (EVM), which maintains a "main chain" to execute contract

code on (normally stored in blockchain blocks) [28]. To execute code from any arbitrary preceding state, we select a previous state as the main chain and execute code from that point.

New proposals, regardless of their semantics (both contract creation and operations on contracts), consist of virtual machine instructions that have been compiled from an API call. The virtual machine is responsible for retrieving the state from a given header and applying given virtual machine instructions to potentially persist a new state and header and—in case of state changes—may return code to share with other users, just like in Ethereum [28].

The biggest difference between GSC's proposals and traditional smart contract execution is the lack of currency (e.g., Ethereum's "gas" [28]). In GSC systems, depending on the chosen hash fabric and consensus mechanism, code is not necessarily pushed to strangers that do not have an intrinsic benefit to run code (i.e., users that have no causal relationship). Therefore, there is no need for currency to power a contract. However, our prototype does still use the currency mechanism of the underlying EVM to protect against infinite loops [28]. To do so, every execution is supplied with an ample amount of artificial currency (equivalent to several millions of dollars).

Contract discovery, consistency, and consensus can be based on locality for GSC systems, and blockchains in general [67], to make a system scalable. Whether or not an application allows for this depends on the system configuration (Constraint 5). In other words, if desired, contracts and their interactions may only be discovered by third parties when they are interacted with. The ability to exploit locality depends on the second concept for GSCs: strong decentralized identities. These strong identities make it possible for users to determine that a particular contract belongs to the user presenting it beyond reasonable doubt (i.e., using cryptography [15]). The locality-based approach ensures that network-wide consensus is not strictly necessary (but can still be applied) to establish ownership of—and interactions with—a contract. The absence of network-wide consensus can make GSCs more energy efficient ("green") than traditional blockchains [3].

What constitutes locality may differ from application to application and governs the permissible underlying networking technology [51]. For example, when the GSC paradigm is used to manage contracts that govern physical systems, networking technology such as Bluetooth or Wi-Fi Direct may suffice. Over the Internet, this locality may be a common application-driven interest of multiple peers (e.g., a particular file in Bittorrent).

5 VALIDATION

One of the key concepts of local-first software is that humans do not produce many "conflicts" when interacting with each other [33]. In smart contracts, these conflicts would materialize as forks. We explore the need for fork resolution in GSCs by running a real-world trace of an Ethereum smart contract, focusing on the time to resolve inconsistencies between nodes. The smart contract used in these simulations is from the "CryptoKitties" game, which allows users to generate and trade cartoonish pictures of cats. Conflicts can arise on a high abstraction level when users attempt to "breed" with each other's cats or transfer their ownership, but also on a lower level when the contract writes to a shared memory address in the virtual machine.

5.1 Dataset

The CryptoKitties game consists of five smart contracts: the "Core" contract, "GeneScience" contract, "Offers" contract, "SalesAuction" contract, and "SiringAuction" contract [31]. The latter three contracts are used to support the exchange of cat pictures and the "GeneScience" contract is used to support generation of entirely new pictures. Apart from these supporting contracts, the core game logic is implemented in the "Core" contract, which we focus on. We query `bitquery.io`, a website for blockchain analytics, for the first 5 839 blocks of the CryptoKitties "Core" smart contract (defined at address `0x06012c8cf97bead5deae237070f9587f8e7a266d`). These blocks contain the first 9 769 transactions, not evenly spread over the blocks (shown in Figure 5(a)), between 543 unique addresses. The transactions contain 12 distinct contract calls defined by the CryptoKitties contract, for which we give the number of occurrences in the dataset in Figure 5(b).

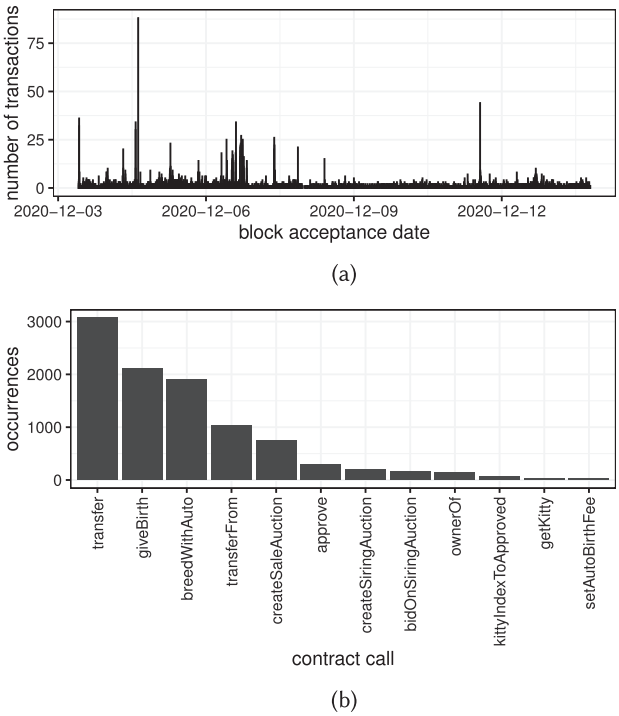


Fig. 5. The number of transactions per block (a) and the number of occurrences per contract call type (b) in the CryptoKitties dataset.

Due to privacy concerns, `bitquery.io` omits the actual argument values of contract calls. We do not attempt to circumvent this omission, but we replace the arguments with random values. Consequentially, the smart contract may execute different instructions in the EVM. Therefore, the state of the smart contract in our simulation is not expected to be equal to that of the real CryptoKitties contract on Ethereum. To mitigate this limitation, our simulation does not keep track of the instructions that were invoked but rather the number of times that the EVM was called to execute any set of instructions.

5.2 Setup and Methodology

Our dataset consists of addresses calling methods on the CryptoKitties contract code, mapped to a network of nodes that propose these method calls to a consensus mechanism. We create a node for each address in our dataset. We use a fully connected network of nodes and we fix the latency for all messages between nodes to $50ms$. This choice is rooted in the network protocol of Bitcoin, in which up to 1000 peers can be discovered in a single message without further communication, well more than the 543 users in the dataset [54]. Similar to a real blockchain, we use a list of blocks as the data structure for the hash fabric and we adopt the oldest-known value to reach consensus.

Our dataset is replayed using a *batches* strategy that introduces transactions using the dataset timestamps and a *spaced* strategy that introduces transactions with a spacing of $250ms$. For the former strategy, every node in our network proposes its transactions using the timestamps defined in the dataset. The batches strategy leads to nodes that have transactions in the same block attempting to claim the same sequence number in our list data structure. Therefore, the consensus mechanism must select one of the conflicting calls. For the spaced strategy, consecutive contract calls in the dataset are spaced by $250ms$, causing all preceding transactions to be finalized before new ones are initiated. Thereby, the second strategy maintains the order of the original transactions, but

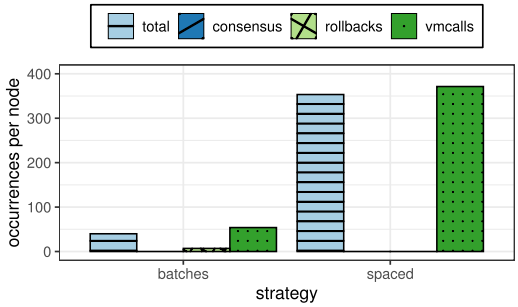


Fig. 6. Average number of occurrences of the four metrics per node for the CryptoKitties dataset replay.

not their timing. The intention of these strategies is to show the difference between real contract interactions and artificial workloads.

We capture a total of five metrics. The first four metrics are occurrences relative to the node count to expose any nonlinear message complexity. Firstly, we keep track of the total number of received messages. Included in the total number of messages are those that may be needed to form consensus, to push messages, and to pull specific messages (due to rollbacks, a single message may need to be pulled more than once). The messages needed for consensus are also counted as the second metric, which is always zero in this simulation, but they will play a role in Section 6. Thirdly, the number of rollbacks of (part of) the hash fabric is recorded. We also keep track of the required EVM calls, as opportunistic execution implies higher CPU loads, which may be restrictive to CPU-limited devices. The fifth metric is the time that nodes are in an inconsistent state, which we call the *convergence time*. The convergence time is the time between when the first consensus message is received for a particular consensus round (which corresponds to its block height; see Figure 4) and the time at which the last consensus message is received.

5.3 Results

We first discuss the metrics related to the number of messages, presented for both of our strategies in Figure 6. As mentioned before, no messages are needed to reach consensus due to the choice of consensus mechanism. Even so, the replay of a real dataset leads to relatively few rollbacks, with 7 rollbacks for 54 EVM calls for the batches strategy and no required rollbacks when the hash fabric is finalized between contract invocations. Therefore, we conclude that real smart contracts do not necessarily have a lot of conflicting interactions. Regarding the different strategies, we see much fewer messages and EVM calls for the batches strategy as opposed to the spaced strategy. This absence of messages occurs due to conflicting interactions not being forwarded and applied in the network, which could be resolved by a retry mechanism.

In Figure 7, we show the time it takes for contract calls to converge for both of our strategies. Our results show that the batches strategy causes the convergence time to reach into the order of several seconds, up to almost a minute. This is explained by nodes attempting to claim the same list index in the list of blocks, which require a consensus mechanism to select only one of the conflicting calls for the index. Our second observation is that the convergence time goes to 0 seconds if each call is proposed 250ms after the last call finished (the spaced strategy). The chosen spacing allows each node to receive the previous contract call and propose a new contract call with a list index that does not conflict with the preceding transaction. Clearly, when a smart contract has very little (or no) conflicting calls, no consensus mechanism is required for a consistent state for all nodes.

5.4 Modeling Conflicts

Our results support that little to no conflicts in the consistency layer enable a system in which network-wide consensus is unnecessary. However, it may be unrealistic to assume that no conflicts occur. For example, out

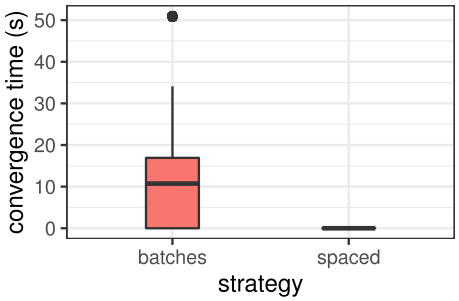


Fig. 7. Box plot of the time until contract interactions have converged, when nodes interact concurrently ("batches") and when interactions are spaced out ("spaced").

of the proposed transactions to Ethereum, an estimated 3% fails [46]. In contrast, in our CryptoKitties replay, we observed that 13% of the proposed transactions fails due to consistency violations when using the batches strategy. Furthermore, in our replay, the resolution of consistency violations is based on the time a call was made. Currently, we do not know of technology that allows timestamps to be (unconditionally) verified. Therefore, we do not have evidence for any system being able to exist to benefit from that finding.

As, to our knowledge, systems do not exist that forego consistency violations in the hash fabric, some form of consensus mechanism is required. Supported by the results of Section 6.3, we believe that using identities to provide conflict resolution is the next-best option for real smart contracts. However, as mentioned in Section 3, using the GSC paradigm and depending on identities is not compatible with all use cases that smart contracts are currently serving. For asset-based use cases, the consensus model of traditional smart contract execution is still the only viable option.

We use a simple model to validate the use of our local-first approach. Given the period of time an invocation is vulnerable to conflicts f, the number of new invocations per second r, and the probability of conflict between two transactions p, the expected number of conflicts experienced by a single transaction is $p \times r \times f$. Our results yield $p = 0.13$ in the worst case of the *batches* strategy and our dataset has $r = 0.011296$ invocations per second. Therefore, invocations succeed without conflict more often than not (when $p \times r \times f < 0.5$) if the time to reach finality for each invocation is $f < 340.49$ seconds. From our results from the *spaced* strategy, we observe that $f < 0.25$ seconds. Therefore, the inequality is satisfied and, at least in the case of CryptoKitties, invocations will succeed without conflict more often than not, validating the use of a local-first approach. In fact, in comparison with the 3% failures of Ethereum, that is, 0.03 expected conflicts, our approach leads to only a fraction of this, with 0.00036712 expected conflicts when $p = 0.13$, $r = 0.011296$, and $f = 0.25$. Furthermore, given $p = 0.13$ and $f = 0.25$, our approach is applicable up to $r = 15.38$ invocations per second per contract. As the most popular smart contracts receive 20 000 invocations per day [48], that is, $r = 0.23$, our approach offers 66.44 times the necessary invocations per second.

6 EXPERIMENTS

Inconsistencies do occur in real smart contracts (Section 5), which are exacerbated in a green local-first approach. GSC architectures must leverage a mechanism to decide a dominant history (Section 4) and, depending on the use case, different consensus mechanisms may be used. For example, if a single leader is permissible, a publish-subscribe communication pattern can be used. If a completely leaderless mechanism is required, a metastable consensus mechanism such as Snowflake [52] can be used. In this section, we conduct experiments in order to provide insight into the consequences of deploying a selection of different consistency and consensus mechanisms.

```
pragma solidity ^0.5.11;
contract SimpleContract {
     uint value = 0;
     function setValue(uint _value) external {
          value = _value;
     }
     function getValue() external view returns(uint) {
          return value;
     }
}
```

Fig. 8. A simple solidity smart contract.

6.1 Setup

Four different mechanisms are used as the consensus mechanism with our GSC prototype: Raft[1] [47], metastable consensus (similar to Snowflake [52]), a publish-subscribe mechanism, and adopting the oldest-known value. The chosen algorithms represent the breadth of approaches to peer-to-peer agreements; we discuss them in further detail later.

To determine a node count for our experiments, we note that Raft was tested with five servers [47], Snowflake was tested with up to 2000 nodes [52], and the remaining two mechanisms depend on the limits of the communication substrate. We pick a middle-ground of up to 1000 nodes for our experiments and we simulate networks of 10, 100, and 1000 nodes. We again use a fully connected network topology, rooted in the same rationale as in Section 5 (even 1000 nodes can be discovered in a single message). We measure the four metrics as in Section 5 that pertain to message handling ("total", "consensus", "rollbacks", "vmcalls"). The smart contract shown in Figure 8, created by S. Verma,[2] is used to run our experiments.

Two different mechanisms for consistency are used. Firstly, the simple list requires all received records to have a unique list index. When two records attempt to occupy the same list index, the consensus mechanism is invoked. Secondly, a conflict-free replicated data type (CRDT) [58] represents the other extreme for consistency mechanisms; it is the recommended data structure for a local-first approach [33]. The CRDT data structure adds incoming records to the current index (a "merge" in a *Sequence CRDT* [45]) until two records are found that are order dependent and, therefore, require a decision from the consensus mechanism (as discussed in Section 4). If the consensus mechanism requires the nodes to take an initial vote, every one of them votes for the oldest record it knows of.

6.2 Methodology

Our methodology consists of introducing a conflict in the simulated network and waiting until all nodes have accepted a new record, resolving the conflict. Our experiment follows three synchronized phases: sharing the initial contract, creating the conflict, and waiting for it to be resolved.

The first phase of our experiment consists of sharing the contract code. Records are created for two indices: the API code ("block 1") and the ABI code ("block 2"). We then wait for all nodes to receive both blocks, forming consensus according to the consensus mechanism (without conflicts, all nodes accept both blocks). We start counting toward our metrics after this first phase has completed.

In the second phase, we introduce a conflict using two nodes that invoke the API of the contract (Figure 8). One node invokes `setValue(11)` and one node invokes `setValue(13)`, which would leave the system in an inconsistent state if both transactions were applied without ordering them (some nodes would have 11 and some nodes

[1]Specifically, https://github.com/streed/simpleRaft.
[2]https://medium.com/better-programming/part-1-brownie-smart-contracts-framework-for-ethereum-basics-5efc80205413.

would have 13 as the return value of `getValue()`). After introducing this conflict, we resume communication between nodes and end the experiment (the final phase) by waiting for all nodes to accept either value.

6.3 Results

The consensus mechanism of adopting the oldest-known value is discussed first. Every node, except for the two originators, only receives a total number of two messages, the lowest number of messages out of all experiments. For the list consistency (Figure 9(a)) the second message conflicts with the first message, which requires a rollback, leading to two executed EVM calls. For the CRDT (Figure 9(b)), nodes attempt a merge of three possible records: `setValue(11)`, `setValue(13)`, and the set `{setValue(11), setValue(13)}`. The former two proposals are made by the originators, whereas all other nodes forward the latter form. The latter form is reevaluated by the originators, leading to the average of just over two EVM calls.

The results for a single publisher, given in Figures 9(c) and 9(d), are largely similar to the results of the oldest-known value. For the list consistency (Figure 9(c)), the required number of rollbacks in the system is now equal to one. Only one of the conflicting messages is published by the originating identity and the other one is rolled back on the node that produced it. For the CRDT consistency layer, the conflicting messages can still exist with the same identifier, which is only later corrected by the consensus mechanism.

Metastable consensus is the first nontrivial consensus protocol, shown in Figures 9(e) and 9(f) (note the change in the vertical axis range). The number of messages increases as the number of nodes increases, which is a direct consequence of the design choice to avoid a leadership election protocol. Due to state deduplication in the EVM calls and the consensus layer being able to freely add and remove messages for a given index, the EVM call count is inflated.

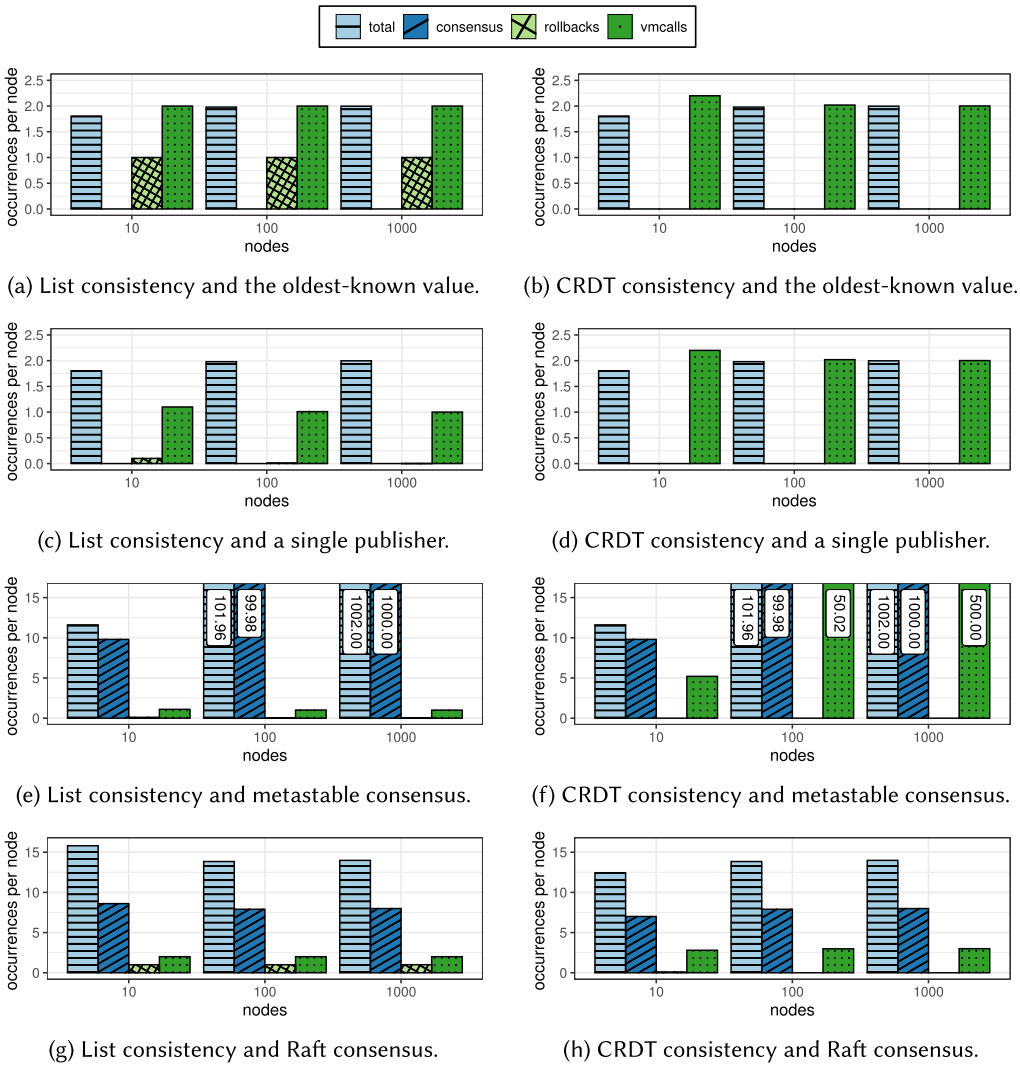
Raft is the final consensus mechanism that we evaluate. Its results, given in Figures 9(g) and 9(h), show a decrease in message count as opposed to metastable consensus, largely due to its leadership election protocol [47]. Electing a single identity to resolve a conflict after its detection requires fewer messages, as opposed to having all nodes converge to a single value over time, as with metastable consensus. However, when the number of messages is a concern, having a pre-established leader (i.e., a single publisher) is still superior.

All of the combinations of consensus mechanisms and consistency mechanisms successfully resolve conflicts, though their environmental impact may be different. The choice of CRDT consistency mainly causes a higher number of EVM calls and—from the perspective of environmental impact—is therefore less desirable due to the higher CPU utilization. More traditional consensus mechanisms are known to have a higher environmental impact [3]. Therefore, the choice of consensus mechanism is key to making GSCs environmentally friendly. Applications should strive to make use of the concepts of GSCs as much as possible if they wish to reduce their environmental impact.

6.4 Model for Environmental Impact

We now define a model for environmental impact by estimating CO_2 emissions based on our observed messaging behaviors and their associated expected conflicts. We derive the behaviors of our four metrics from our results, shown in Table 2. The behaviors we define are a simplification of our actual results as they change with the number of nodes. For example, though we model the total number of messages for CRDT and Raft as $14n$, our results are actually $12.4n$ for 10 nodes, $13.84n$ for 100 nodes, and $13.98n$ for 1000 nodes. Our defined behaviors become more accurate as the number of nodes grows. As related work suggests that the exchanged number of messages of a method is the primary driver for environmental impact [3], we focus on three functions to describe the distinct messaging behaviors (Table 2): $b_1(n) = 2n$, $b_2(n) = 14n$, and $b_3(n) = n^2$.

For each conflict that is introduced, its total number of messages $b_i(n)$ are added to the aggregate total number of messages for the entire network. In Section 5.4, we determined that the number of conflicts is given by $p \times r \times f$. Therefore, the aggregated total number of messages is given by $p \times r \times f \times b_i(n)$. For fully connected networks, f is simply the maximum latency l between nodes, while random graphs may have $f = O(log(n) \times l)$ and linear



(a) List consistency and the oldest-known value.

(b) CRDT consistency and the oldest-known value.

(c) List consistency and a single publisher.

(d) CRDT consistency and a single publisher.

(e) List consistency and metastable consensus.

(f) CRDT consistency and metastable consensus.

(g) List consistency and Raft consensus.

(h) CRDT consistency and Raft consensus.

Fig. 9. The number of occurrences of the "total", "consensus", "rollbacks", and "vmcalls" metrics versus the number of nodes for the eight combinations of consistency and consensus mechanisms. Bars that go beyond the vertical plotting range are labeled.

topologies have $f = O(n \times l)$. By assuming, without loss of generality, that r is measured per time unit l, we can eliminate l from our equations. Thereby, we obtain the following three functions for the aggregated total number of messages: $O(m_1) = p \times r \times O(1) \times b_i(n)$, $O(m_2) = p \times r \times O(log(n)) \times b_i(n)$, and $O(m_3) = p \times r \times O(n) \times b_i(n)$.

To tie the messaging behavior to CO_2 emissions, we require estimates for the energy expenditure of transmitted messages and the emissions of the expended energy. Firstly, a realistic upper bound for the energy expenditure per gigabyte is $0.2kWh/GB$ [12]. Secondly, though the CO_2 emissions vary per region, we use the United States average of $0.603kgCO_2/kWh$ to model emissions [2]. We use the Ethereum maximum smart contract invocation size of $24kB$ as the message size, assuming that the overhead data needed for consensus is negligible. Given these assumptions, we obtain a result of $0.0028944\ gCO_2/message$ to estimate our carbon emissions.

Table 2. Behavior of Our Four Metrics to Resolve a Conflict Between Two Contract Invocations Given the Number of Nodes (n) for the Measured Consistency and Consensus Mechanisms

Consistency	Consensus	Total Msgs.	Consensus Msgs.	Rollbacks	EVM Calls
List	Oldest-Known	$2n$	0	n	$2n$
CRDT	Oldest-Known	$2n$	0	0	$2n$
List	Publish-Subscribe	$2n$	0	1	n
CRDT	Publish-Subscribe	$2n$	0	0	$2n$
List	Metastable	n^2	n^2	0	n
CRDT	Metastable	n^2	n^2	0	$\frac{1}{2}n^2$
List	Raft	$14n$	$8n$	n	$2n$
CRDT	Raft	$14n$	$8n$	0	$3n$

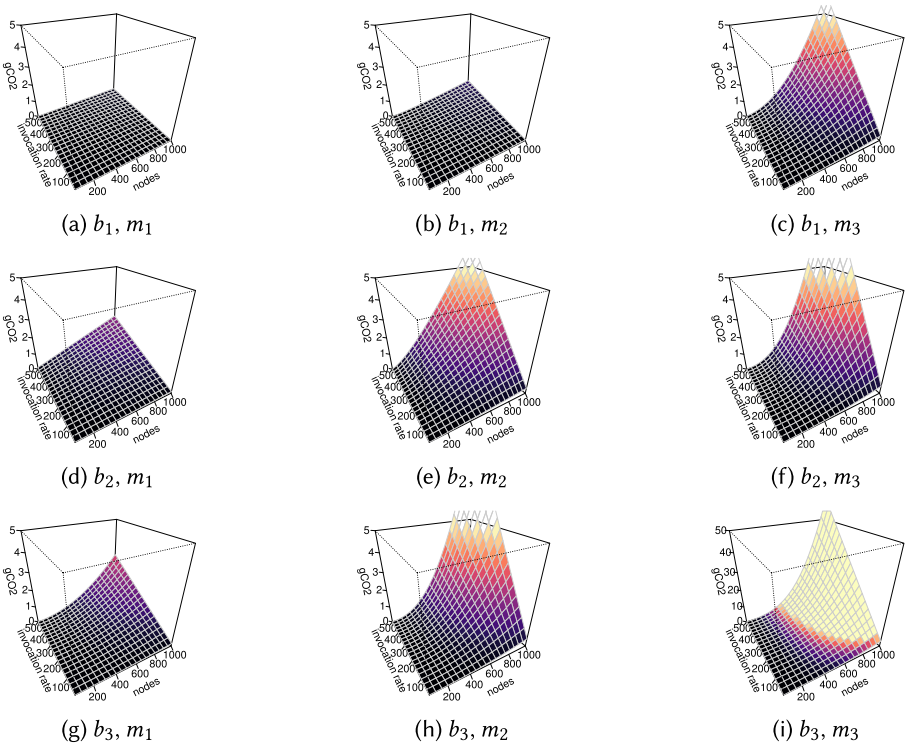
We evaluate our model to calculate CO_2 emissions for the nine different combinations of b_i (b_1, b_2, and b_3) and m_j (m_1, m_2, and m_3). To compare between the different behaviors, we fix the value $p = 0.13$ to match our CryptoKitties results. Our results are visualized in Figure 10. They highlight that a low node count or a low invocation rate trivially keeps carbon emissions down. However, when the node count and the invocation rate are both increased, the choice of consensus mechanism and messaging topology starts to matter. With a local-first approach of fully connected nodes (m_1), the choice of consensus mechanism is hardly an influence. For a more loosely connected messaging topology (m_2 and m_3), a fixed publisher or predetermined finalization strategy such as "oldest-known" (b_1) is preferential. Given a typical peer-to-peer network with logarithmic message propagation time and Raft consensus (i.e., b_2, m_2), even the worst-performing consensus mechanism with local-first (i.e., b_3, m_1) has only 47.69% of the carbon emissions at $n = 1000, r = 500$. In the majority of these cases, the Green Smart Contract approach is a greener approach.

7 RELATED WORK

Throughout this work, we have highlighted the individual works that are closely related to the topics we addressed. We now position our contributions on a coarser scale. On the coarsest scale, one can consider using solar panels to power the hardware of nodes [37], which necessitates additional physical hardware, and the trade of emission certificates [9, 29], which requires a secondary market. For a more focused discussion, we regard work that proposes changes to software architecture.

One of the key works that precedes our work and falls within our definition of a "recent proposal" (Figure 1(b)) is the Hedera Hashgraph [7]. The Hedera Hashgraph proposes both a weaker consistency model (in the form of a Directed Acyclic Graph data structure) and a weaker consensus model (which they call "asynchronous Byzantine Fault Tolerance"). Furthermore, the consensus layer is optionally permissioned (i.e., configurable). Whereas Hedera is similar to our Green Smart Contracts in its relaxations, it falls short of going to the extreme of a local-first model and it uses the traditional blockchain model for users to offer interactions to nodes. Our work goes one step further and actively embraces a new paradigm for user interactions with smart contracts.

Our work is closely related to the "local-first software" proposal by Kleppmann et al. [33], which mainly focuses on change-based updates to shared data structures. Our work builds on their findings and uses a local-first approach to the consistency layer in smart contracts. However, the work of Kleppmann et al. is mainly focused on how humans interact to form a shared data structure. One of their conclusions is that "conflicts are not as significant a problem" because "users have an intuitive sense of human collaboration and avoid creating conflicts". Whereas we have found that there are certainly less conflicts in a real-world smart contract than in a lab setting (see Section 5), with a 13% failure rate, we believe that conflicts are a significant problem for smart contract execution.



(a) b_1, m_1 (b) b_1, m_2 (c) b_1, m_3

(d) b_2, m_1 (e) b_2, m_2 (f) b_2, m_3

(g) b_3, m_1 (h) b_3, m_2 (i) b_3, m_3

Fig. 10. Estimated carbon emissions as a function of the invocation rate and the number of nodes, given functions b_i for messaging behavior and functions m_j for the aggregated total number of messages.

Smart contract execution is the next evolution after the state machine replication movement (which came after shared memory systems). Even though smart contract execution does away with any trust assumptions between nodes, Byzantine Fault Tolerance (BFT) in its execution model is still shared with the domain of state machine replication. Very few works explore execution without a critical dependency on BFT consensus. Of particular note is Eve [32], proposing state machine replication with tunable fault tolerance to speed up execution. The proposal of the authors of [32] is to first agree on an order of operations, then to execute those operations, and lastly to verify the resulting state. In contrast, our work proposes to execute operations opportunistically, verify the resulting state, and then to agree on an order only when a conflict is found. However, just like our work, Eve exposes a design space of "nondeterminism introduced by allowing parallel execution". Our work argues that this design space consists of liability-based applications.

Our proposal to make contracts the central point of interaction is closely related to the publish-subscribe communication pattern. However, regarding the execution of program code, most works only consider the publish-subscribe pattern to distribute executable tasks to nodes (e.g., Sadoghi et al. [55] and Dayal et al. [16]). Jehl and Meling [30] explore a publish-subsribe-based state machine model based on broadcasts in the presence of Byzantine failures. In contrast to the aforementioned work (and in agreement with local-first software), we believe that a single publisher (a contract) and its subscribers (the users that interact with the contract) naturally emerge from smart contract use and do not require additional reliable broadcast.

8 CONCLUSION

Green Smart Contracts are able to challenge Bitcoin and Ethereum as ubiquitous technology, to serve all applications. This work has identified the concept of liabilities, to replace the established model of cryptocurrencies

serving as the primitive to support smart contract execution. By depending on liabilities as a primitive, Green Smart Contracts are able to maximally leverage weak consistency and probabilistic consensus to form a novel green "local-first" architectural paradigm for smart contract execution. The requirement of stronger identity management for such architectures can be overcome without violating the context of permissionless and trustless smart contract execution. The benefit of our local-first approach is that the requirements of applications that use liabilities are severely lowered as opposed to applications that depend on assets. Going forward, smart contract applications should carefully examine their application domain to potentially make use of the communication improvements offered by the Green Smart Contract paradigm. Future digitization and automation can become greener and more performant.

ACKNOWLEDGMENTS

The authors thank Can Umut Ileri and the anonymous reviewers whose comments helped improve and clarify this work.

REFERENCES

[1] Alex Auvolat, Davide Frey, Michel Raynal, Francois Taiani, et al. 2020. Money transfer made simple: A specification, a generic algorithm, and its proof. *Bulletin of EATCS* 3, 132 (2020).

[2] Inês M. Lima Azevedo, M. Granger Morgan, and Lester Lave. 2011. Residential and regional electricity consumption in the US and EU: How much will higher prices reduce CO2 emissions? *The Electricity Journal* 24, 1 (2011), 21–29.

[3] Abigael Okikijesu Bada, Amalia Damianou, Constantinos Marios Angelopoulos, and Vasilios Katos. 2021. Towards a green blockchain: A review of consensus mechanisms and their energy consumption. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 503–511.

[4] Johannes Bahrke and Charles Manoury. 2021. Commission proposes a trusted and secure Digital Identity for all Europeans. European Commission. (June 2021). Retrieved May 16, 2022 from https://ec.europa.eu/commission/presscorner/detail/en/IP_21_2663.

[5] Chong Bai. 2018. State-of-the-art and future trends of blockchain based on DAG structure. In *International Workshop on Structured Object-Oriented Formal Language and Method*. Springer, 183–196.

[6] Leemon Baird. 2016. The Swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep* 34 (2016).

[7] Leemon Baird, Mance Harmon, and Paul Madsen. 2019. Hedera: A public hashgraph network & governing council. *White Paper* 1 (2019).

[8] Roman Beck, Christoph Müller-Bloch, and John Leslie King. 2018. Governance in the blockchain economy: A framework and research agenda. *Journal of the Association for Information Systems* 19, 10 (2018), 1.

[9] Umit Cali, Komal Khan, Shammya Shananda Saha, Tamara Hughes, Farrokh Rahimi, Leonard C. Tillman, Islam El-Sayed, Pablo Arboleya, and Sri Nikhil Gupta Gourisetti. 2022. Smart contract as an enabler for the digital green transition. In *2022 IEEE PES Transactive Energy Systems Conference (TESC)*. IEEE, 1–5.

[10] Ming Cao and Chai Wah Wu. 2007. Topology design for fast convergence of network consensus algorithms. In *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 1029–1032.

[11] Lin William Cong and Zhiguo He. 2019. Blockchain disruption and smart contracts. *The Review of Financial Studies* 32, 5 (2019), 1754–1797.

[12] Vlad C. Coroama and Lorenz M. Hilty. 2014. Assessing Internet energy intensity: A review of methods and results. *Environmental Impact Assessment Review* 45 (2014), 63–68.

[13] Arturo Crespo and Hector Garcia-Molina. 2004. Semantic overlay networks for P2P systems. In *International Workshop on Agents and P2P Computing*. Springer, 1–13.

[14] Pierluigi Cuccuru. 2017. Beyond bitcoin: An early overview on smart contracts. *International Journal of Law and Information Technology* 25, 3 (2017), 179–195.

[15] Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. 2003. Beyond "web of trust": Enabling P2P E-commerce. In *EEE International Conference on E-Commerce, 2003. CEC 2003.* IEEE, 303–312.

[16] Jai Dayal, Drew Bratcher, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. 2014. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 246–255.

[17] Rubén de Juan-Marín, Hendrik Decker, José Enrique Armendáriz-Íñigo, José M. Bernabéu-Aubán, and Francesc D. Muñoz-Escoí. 2016. Scalability approaches for causal multicast: A survey. *Computing* 98, 9 (2016), 923–947.

[18] Rodrigo Couto de Souza, Edimara Mezzomo Luciano, and Guilherme Costa Wiedenhöft. 2018. The uses of the blockchain Smart Contracts to reduce the levels of corruption: Some preliminary thoughts. In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*. 1–2.

[19] Randall E. Duran and Paul Griffin. 2019. Smart contracts: Will Fintech be the catalyst for the next global financial crisis? *Journal of Financial Regulation and Compliance* (2019).
[20] Helen Eenmaa-Dimitrieva and Maria José Schmidt-Kessen. 2019. Creating markets in no-trust environments: The law and economics of smart contracts. *Computer Law & Security Review* 35, 1 (2019), 69–88.
[21] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. SoK: Transparent dishonesty: Front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security*. Springer, 170–189.
[22] Michal Feldman and John Chuang. 2005. Overcoming free-riding behavior in peer-to-peer systems. *ACM SIGecom Exchanges* 5, 4 (2005), 41–50.
[23] Gartner. 2018. Digitalization Will Make Most Heritage Financial Firms Irrelevant. (10 2018). https://www.gartner.com/en/doc/338356-digitalization-will-make-most-heritage-financial-firms-irrelevant.
[24] Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. 2014. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 326–335.
[25] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. 2007. PeerReview: Practical accountability for distributed systems. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 175–188.
[26] Samer Hassan and Primavera De Filippi. 2017. The expansion of algorithmic governance: From code is law to law is code. *Field Actions Science Reports. The Journal of Field Actions* Special Issue 17 (2017), 88–90.
[27] Mike Hearn and Richard Gendal Brown. 2016. Corda: A distributed ledger. *Corda Technical White Paper* 2016 (2016).
[28] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. 2018. KEVM: A complete formal semantics of the Ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 204–217.
[29] Fabien Imbault, Marie Swiatek, Rodolphe de Beaufort, and Robert Plana. 2017. The green blockchain: Managing decentralized energy production and consumption. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*. IEEE, 1–5.
[30] Leander Jehl and Hein Meling. 2013. Towards byzantine fault tolerant publish/subscribe: A state machine approach. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems*. 1–5.
[31] Xin-Jian Jiang and Xiao Fan Liu. 2021. Cryptokitties transaction network analysis: The rise and fall of the first blockchain game mania. *Frontiers in Physics* (2021), 57.
[32] 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 *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 237–250.
[33] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. 2019. Local-first software: You own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 154–178.
[34] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. 2009. TrInc: Small trusted hardware for large distributed systems. In *NSDI*, Vol. 9. 1–14.
[35] Chenxin Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. 2020. A decentralized blockchain with high throughput and fast confirmation. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 515–528.
[36] Shancang Li, Shanshan Zhao, Po Yang, Panagiotis Andriotis, Lida Xu, and Qindong Sun. 2019. Distributed consensus algorithm for events detection in cyber-physical systems. *IEEE Internet of Things Journal* 6, 2 (2019), 2299–2308.
[37] Juan Liu, Jun Lv, Hasan Dinçer, Serhat Yüksel, and Hüsne Karakuş. 2021. Selection of renewable energy alternatives for green blockchain investments: A hybrid IT2-based fuzzy modelling. *Archives of Computational Methods in Engineering* (2021), 1–15.
[38] Scott Locklin. 2018. Token economics. (2018). https://basicattentiontoken.org/static-assets/documents/token-econ-2018.pdf.
[39] Matt Lockyer, N. Mudge, and J. Schalm. 2015. Erc-998 composable non-fungible token standard. *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland* (2015).
[40] Alexander Löser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. 2003. Semantic overlay clusters within super-peer networks. In *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*. Springer, 33–47.
[41] Daniele Magazzeni, Peter McBurney, and William Nash. 2017. Validation and verification of smart contracts: A research agenda. *Computer* 50, 9 (2017), 50–57.
[42] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 120–130.
[43] Satoshi Nakamoto. 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Technical Report.
[44] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. 2019. FabricCRDT: A conflict-free replicated datatypes approach to permissioned blockchains. In *Proceedings of the 20th International Middleware Conference*. 110–122.
[45] Brice Nédelec, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils. 2013. LSEQ: An adaptive structure for sequences in distributed collaborative editing. In *Proceedings of the 2013 ACM Symposium on Document Engineering*. 37–46.
[46] Vinicius C. Oliveira, Julia Almeida Valadares, Jose Eduardo A. Sousa, Alex Borges Vieira, Heder Soares Bernardino, Saulo Moraes Villela, and Glauber Dias Goncalves. 2021. Analyzing transaction confirmation in Ethereum using machine learning techniques. *ACM SIGMETRICS Performance Evaluation Review* 48, 4 (2021), 12–15.

[47] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 305–319.
[48] Andrea Pinna, Simona Ibba, Gavina Baralla, Roberto Tonelli, and Michele Marchesi. 2019. A massive analysis of Ethereum smart contracts empirical study and code metrics. *IEEE Access* 7 (2019), 78194–78213.
[49] Serguei Popov. 2018. The tangle. *White Paper* 1, 3 (2018).
[50] Thomas Puschmann. 2017. Fintech. *Business & Information Systems Engineering* 59, 1 (2017), 69–76.
[51] Muhammad Raza, Venkatesh Samineni, and William Robertson. 2016. Physical and logical topology slicing through SDN. In *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 1–4.
[52] Team Rocket. 2018. Snowflake to Avalanche: A novel metastable consensus protocol family for cryptocurrencies. (2018). https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV.
[53] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2019. Scalable and probabilistic leaderless BFT consensus through metastability. *arXiv preprint arXiv:1906.08936* (2019).
[54] Muhammad Saad, Songqing Chen, and David Mohaisen. 2021. Root cause analyses for the deteriorating bitcoin network synchronization. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 239–249.
[55] Mohammad Sadoghi, Martin Jergler, Hans-Arno Jacobsen, Richard Hull, and Roman Vaculín. 2015. Safe distribution and parallel execution of data-centric workflows over the publish/subscribe abstraction. *IEEE Transactions on Knowledge and Data Engineering* 27, 10 (2015), 2824–2838.
[56] Ermin Sakic and Wolfgang Kellerer. 2020. Decoupling of distributed consensus, failure detection and agreement in SDN control plane. In *IFIP Networking 2020*. 9.
[57] Johannes Sedlmeir, Hans Ulrich Buhl, Gilbert Fridgen, and Robert Keller. 2020. The energy consumption of blockchain technology: beyond myth. *Business & Information Systems Engineering* 62, 6 (2020), 599–608.
[58] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*. Springer, 386–400.
[59] Voshmgir Shermin. 2017. Disrupting governance with blockchains and smart contracts. *Strategic Change* 26, 5 (2017), 499–509.
[60] James E. Smith and Ravi Nair. 2005. The architecture of virtual machines. *Computer* 38, 5 (2005), 32–38.
[61] Quinten Stokkink, Can Umut Ileri, and Johan Pouwelse. 2022. Reputation-based data carrying for Web3 networks. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. IEEE, 283–286.
[62] Quinten Stokkink, Georgy Ishmaev, Dick Epema, and Johan Pouwelse. 2021. A truly self-sovereign identity system. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, 1–8.
[63] Quinten Stokkink and Johan Pouwelse. 2018. Deployment of a blockchain-based self-sovereign identity. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 1336–1342.
[64] Yuechen Tao, Bo Li, Jingjie Jiang, Hok Chu Ng, Cong Wang, and Baochun Li. 2020. On sharding open blockchains with smart contracts. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1357–1368.
[65] Anjan V. Thakor. 2020. Fintech and banking: What do we know? *Journal of Financial Intermediation* 41 (2020), 100833.
[66] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 67–82.
[67] Lewis Tseng. 2019. Eventual consensus: Applications to storage and blockchain. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 840–846.
[68] F. Vogelsteller and V. Buterin. 2015. ERC-20 token standard. Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland (2015).
[69] Shermin Voshmgir. 2020. *Token Economy: How the Web3 Reinvents the Internet*. Vol. 2. Token Kitchen.
[70] Moritz Wendl, My Hanh Doan, and Remmer Sassen. 2023. The environmental impact of cryptocurrencies using proof of work and proof of stake consensus algorithms: A systematic review. *Journal of Environmental Management* 326 (2023), 116530.
[71] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. (2014). https://gavwood.com/paper.pdf.
[72] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. 2020. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 1432–1465.
[73] Aviv Zohar. 2015. Bitcoin: Under the hood. *Commun. ACM* 58, 9 (2015), 104–113.

Received 14 December 2022; revised 22 May 2023; accepted 20 June 2023