NimbleChain: Speeding up Cryptocurrencies in General-purpose Permissionless Blockchains

PAULO SILVA, MIGUEL MATOS, and JOÃO BARRETO, INESC-ID and IST, Universidade de Lisboa, Portugal

Nakamoto's seminal work gave rise to permissionless blockchains, as well as a wide range of proposals to mitigate their performance shortcomings. Despite substantial throughput and energy efficiency achievements, most proposals only bring modest (or marginal) gains in transaction commit latency. Consequently, commit latencies in today's permissionless blockchain landscape remain prohibitively high.

This article proposes NimbleChain, a novel algorithm that extends permissionless blockchains based on Nakamoto consensus with a fast path that delivers *causal promises of commitment* or simply *promises*. Since promises only partially order transactions, their latency is only a small fraction of the totally ordered commitment latency of Nakamoto consensus. Still, the weak consistency guarantees of promises are *strong enough* to correctly implement cryptocurrencies. To the best of our knowledge, NimbleChain is the first system to bring together fast, partially ordered transactions with consensus-based, totally ordered transactions in a permissionless setting. This hybrid consistency model is able to speed up cryptocurrency transactions while still supporting smart contracts, which typically have (strong) sequential consistency needs.

We implement NimbleChain as an extension of Ethereum and evaluate it in a 500-node geo-distributed deployment. The results show NimbleChain can promise a cryptocurrency transactions up to an order of magnitude faster than a vanilla Ethereum implementation, with marginal overheads.

1 INTRODUCTION

The majority of permissionless blockchains, including mainstream ones such as Bitcoin [52] or Ethereum [70], rely on the foundations of Nakamoto's seminal consensus protocol [52]. In these systems, the probability that a given block has stabilized grows with the number of blocks that succeed it in the chain. Hence, by setting a high-enough threshold to consider a block—and the transactions therein–as committed, one can ensure an arbitrarily

low probability of the block being discarded. This is commonly known as *finality* and has been formalized as a *persistence* property by Garay et al. [20].

Permissionless blockchains can thereby be used as a (probabilistic) totally ordered distributed ledger service with unique features, in particular their resilience to Sybil attacks that characterize permissionless environments. Permissionless blockchains support a wide range of geo-distributed applications, from cryptocurrencies to general-purpose smart contracts, but unfortunately they are also known for their poor performance.

In recent years, the research community has contributed with important improvements to permissionless blockchains, such as to Nakamoto's longest chain rule [40, 42, 65, 66, 70], hierarchical and parallel chains [6, 19, 56, 71], sharded blockchains [17, 37, 46, 72], Layer-2 approaches [14, 27, 28, 49, 59] **Byzantine Fault Tolerance– (BFT)** based blockchains [1, 23, 36, 45, 51, 57], and Proof-of-X alternatives [2, 5, 10, 15, 23, 32, 73]. These approaches have focused mostly on improvements to throughput and/or energy efficiency, bringing only modest improvements to commit latency. Some notable exceptions reduce commit latencies by sacrificing security or scalability (as we discuss in Section 8). In fact, as of today, commit latencies remain notably high in mainstream permissionless blockchains—around 1 hour in Bitcoin and 3 minutes in Ethereum.

Such high commit latencies are especially prohibitive for many applications that require low-latency transactions. Examples range from merchant applications that need to deliver goods quickly (<30 seconds) [30], such as point-of-sale purchases and retail vending machines, take-away stores, online shopping, supermarket checkouts, bike sharing systems [7, 13, 26, 30], and blockchain-backed **Internet of Things (IoT)** devices and applications [27]. To overcome the high commit delays, merchants and service providers frequently adopt risky 0- or 1-confirmation policies [13], which accept transactions as granted well before the underlying blockchain can provide sufficiently strong guarantees of their persistence. Such policies are inherently vulnerable to double-spending attacks.

The commit latencies of mainstream permissionless blockchains reflect two well-known limitations of Nakamoto's consensus: For correctness, blocks need to be generated (on average) at a slow pace with respect to network latency, and a transaction should only be considered as committed (i.e., persistent) after it is followed by a long sequence of (slowly generated) blocks in the chain [20]. To further complicate matters, a recent study [61] concluded that, due to the emergence of powerful mining pools, blockchain systems should wait for even larger blockchain suffixes before committing, which means that commit latency is bound to increase rather than decrease.

This article focuses on improving the latency of general-purpose permissionless blockchains. Our insight is based on the observation that the actual consistency needs of cryptocurrencies, which constitute the bulk of today's most important permissionless blockchains (i.e., almost 100% in Bitcoin [8] and 44% in Ethereum [61]), can be satisfied without resorting to consensus [25]. With this in mind, we set out to build a general-purpose permissionless blockchain that offers a hybrid consistency model where two distinct transaction handling paths gracefully coexist, each one serving applications with differing consistency needs. In detail, a novel fast path that handles weakly consistent cryptocurrency transactions, together with the existing slow path that processes the remaining strongly consistent transactions—such as smart contract transactions—in a consensus-based, totally ordered fashion.

This article comprises contributions to achieve the above vision. *As a first contribution,* we propose an extension to the traditional issue/commit model by introducing the notion of *causal promise of commitment* or simply *promise.* A promise is a new event in a transaction's lifecycle. Informally, when a process promises a transaction t, that means that t will be eventually committed by every correct process, even if t is part of a double-spending attempt by a malicious user, while satisfying any causal dependencies of t. Promises offer weaker consistency guarantees than traditional blockchain consensus—most importantly, promises are only partially ordered. Therefore, promises can be implemented by a faster protocol in a small fraction of the commitment latency. Despite weaker, the guarantees of the promise event are still *strong enough* to fulfil the consistency needs of cryptocurrencies.

Table 1. Comparison of NimbleChain's Promise/Commit Events with Ethereum
(An Instance of BBP)

System	Event	Double-spending-resistant	Consistency	Latency
Ethereum	Commit	Yes	Total order	~3 min
NimbleChain	Promise	Yes	Causal	Typically ~4 s to ~25 s
	Commit	Yes	Total+causal	~3 m

Namely, they are causally ordered and resist double-spending attacks. Table 1 summarizes the key differences between promising and committing a transaction.

As a second contribution, we propose NimbleChain, a novel permissionless blockchain system that extends permissionless blockchains based on the **Bitcoin Backbone Protocol (BBP)** [20] with a *partially ordered promise fast path*. This fast path allows cryptocurrency transactions, which constitute the bulk of today's most important permissionless blockchains (i.e., almost 100% in Bitcoin [8] and 44% in Ethereum [61]) to be promised substantially faster than the original commit, in a large portion of correct processes. Since today's general-purpose blockchains also support smart contracts, which typically have sequential consistency needs, NimbleChain also supports strongly consistent transactions. Together, offers a *hybrid* consistency model that is able to handle transactions supports both types of transactions, as Table 1 outlines. To the best of our knowledge, NimbleChain is the first permissionless blockchain system to bring together fast partially ordered transactions with totally ordered, consensus-based transactions in a permissionless setting.

As a third contribution, we implement NimbleChain as an extension of the Ethereum blockchain, which demonstrates the practicality of our proposal.

As a final contribution, an evaluation with 500 processes in a realistic geo-distributed environment that shows that NimbleChain's fast path reduces the commit latencies of cryptocurrency transactions by an order of magnitude with negligible overhead when compared with Ethereum.

The rest of the article is organized as follows. Section 2 provides background on permissionless blockchains and describes a generic baseline protocol. Section 3 defines the notion of promises, and Section 4 shows how NimbleChain extends the baseline with promises. Section 5 describes how we can leverage NimbleChain's promises to implement low-latency cryptocurrencies. Section 6 evaluates our implementation of NimbleChain as an extension of Ethereum in a large-scale geo-distributed scenario. Section 7 discusses the limitations of NimbleChain. Section 8 surveys related work, and Section 9 concludes the article.

2 BACKGROUND

Today's most valuable permissionless blockchains, such as Bitcoin or Ethereum, are based on the foundations laid by the consensus protocol proposed by Nakamoto [52]. Different studies [20, 21, 33, 55] have formally analyzed the common core of such blockchain protocols. Hereinafter, we adopt the term coined by Garay et al. [20], BBP, to generically refer to the core protocol behind today's permissionless blockchains inspired by Nakamoto's consensus protocol.

Garay et al. [20] established two main properties of BBP, persistence and liveness (further discussed below), and formally studied them under a strict (and unrealistic) network model. Subsequent works showed such properties also hold for more realistic network models [21, 33, 55]. In this work, we consider BBP under the latter models as our starting point, which we extend to build NimbleChain.

In the next sections, we present the system model, the main properties of BBP, and conclude with an overview of the protocol.

2.1 Assumptions

The BBP runs on a peer-to-peer network of processes and relies on the following assumptions [32, 33, 55, 58]. First, Byzantine adversaries control less than 50% of the total mining power that is used to produce blocks and

Table 2. Notation Used in the Paper

Symbol	Meaning
D	Maximum message delivery delay
BBP	Bitcoin Backbone Protocol [20]
B	Average block generation time
C	BBP commit threshold (minimum chain suffix height)

do not have computational power to subvert the standard cryptographic primitives. Any message broadcast by a correct process is delivered to every correct process with high probability, and with a maximum delay of D [32, 55, 58, 60]. This is often a hidden assumption of permissionless blockchains systems, but it is in fact a critical aspect of the system that we not only expose but also embrace. In fact, in Nakamoto consensus, the difficulty of the **Proof-of-Work (PoW)** puzzle is based on the maximum delivery delay D [55, 60], and hence this is an assumption shared by Bitcoin and Ethereum. The same applies to proof-of-stake blockchains such as Ouroboros [32] and Algorand [23], which assume a bounded delay. The correctness of BBP, and systems such as Bitcoin or Etherum, thus depends on the premise that the system takes much longer to generate a new block than to propagate it [20]. Let us denote B as the overall average block generation time. Hence, we assume that D/B is relatively small [20, 31, 33, 55].

For presentation simplicity, we assume that all processes can produce (i.e., mine) blocks and act as clients by submitting transactions to the system. Moreover, we also assume that each correct client is collocated with a correct process participating in the protocol. Devising robust solutions to support correct clients that interact with possibly Byzantine processes running the protocol is an open problem, which is orthogonal to our work [29].

A transaction is signed by the process that issues it. The body of a transaction is an application-dependent payload (e.g., the target account and amount to transfer in a cryptocurrency transaction or, in a smart contract transaction, the target smart contract and its method and arguments).

Each transaction also carries a local sequence number, which is consecutively unique for that issuer.[1] Note that a Byzantine process p can submit two or more distinct transactions with the same sequence number, with the intention of having some correct processes commit one transaction (and, hence, discard the other one(s) as invalid), and other corrects processes decide in the opposite direction. For any pair of distinct transactions that share the same sequence number, we say that each transaction is a *double spending* of the other (and vice versa).

We summarize the notation used in the article in Table 2.

2.2 Properties

We start by defining the event of committing a transaction as follows. If a correct process p has a block b in its blockchain followed by at least C other blocks, then we say that every transaction included in b is *committed* at p.

Garay et al. [20] define the following two main properties of the commit event of BBP, with high probability:[2]

Persistence Property. *If a correct process p commits a block b, and consequently the transactions in b, then any correct process has block b in the same position in the blockchain, from this moment on.*

Liveness Property. *If a correct process submits a transaction t, then all correct process eventually commit t.*

Different works have formally studied the above properties, using different methods. Starting from Garay et al.'s initial analysis, which considered simplified network assumptions [20], subsequent works [21, 33, 55] have

---

[1]This is the case, for instance, with Ethereum's transaction nonces. Still, we note that BBP can generically abstract blockchain protocols without per-transaction sequence numbers.

[2]For presentation simplicity, we present a formulation that is simpler than Garay et al.'s original one. Namely, we omit some parameters that are orthogonal to our contributions in this article and explicitly use the term *commit*.

formally shown that BBP guarantees such properties under the more realistic assumptions that we consider in Section 2.1.

Two important corollaries can be easily drawn from the above properties. The first one is that if a correct process p commits a transaction t, then all other correct processes will eventually commit t. A second corollary is that if a correct process p commits transaction t_A and, later, transaction t_B, then any other correct process q will commit both transactions in the same order—in other words, the commit event is totally ordered.

Together, the above properties and the associated corollaries constitute strong consistency guarantees, which BBP provides with high probability. These enable many geo-distributed applications to operate even when operating under an adversarial permissionless environment. For instance, state-machine replication can be built on top of BBP through smart contracts. Yet, these guarantees are provided at the cost of a high latency.

2.3 BBP Algorithm

At a high level, the algorithm works as follows. Each process holds a local ledger of transactions that consolidates two components: a local copy of the blockchain (or, simply, local chain) and the *mempool*. The local chain is organized as a cryptographically linked list of blocks. Blocks have a monotonically increasing gap-free sequence number, and each block includes a totally ordered sequence of transactions. The mempool is a local queue of individual transactions not yet included in the local chain. The transactions in the mempool are ordered after the transactions in the local chain. When receiving a transaction, a process performs a series of validations such as checking for funds, checking whether another transaction with the same identifier already exists in the local ledger (i.e., a double-spend attempt) and verifying the transaction's digital signature, among others. If the transaction is valid and not yet in the local chain, then it is inserted in the mempool.

Processes initially share a genesis block b_0 and produce blocks by selecting a subset of transactions in the mempool and creating a proof that depends on these transactions and on the last block in the local chain. The proof is a Sybil-proof leader election mechanism, such as PoW, that ensures the process legally produced the block. The puzzle difficulty of PoW is a function of D [58].

Upon producing a new block, a process appends it to its local chain and broadcasts it. When a block is delivered, it is validated before being included in the local chain. Due to concurrency, it is possible that two or more processes produce two competing blocks b_{i+1} and b'_{i+1} that extend the sequence $(b_0, \ldots, b_i)$. Processes will select either sequence $(b_0, \ldots, b_i, b_{i+1})$ or $(b_0, \ldots, b_i, b'_{i+1})$, using a **chain selection rule (CSR)**. In Nakamoto consensus, the CSR corresponds to selecting the chain with greatest PoW effort. By relying on the CSR, the protocol will make the system converge by gradually agreeing on an increasing *common blockchain prefix*. Blocks, and the transactions therein, are considered committed, when followed by a sequence of C blocks.

2.4 The Weak Consistency Needs of Cryptocurrencies

Our work hinges on the observation that cryptocurrency transactions, also known as asset transfer transactions, do not need to be totally order [25]. The intuition behind this observation is the following. In an asset transfer scenario, the system maintains a set of accounts where each account maintains the number of assets held, for instance the amount of cryptocurrency in a given account. Each account has a single owner that can withdraw funds from this account and transfer them to other accounts. The other users in the system can only read the accounts balance and transfer funds to it (from their own accounts). Therefore, it is the sole responsibility of an account owner to order the assets withdrawn from that account.

Intuitively, this removes the need to order a transaction that withdraws assets from a given account relatively to every other transaction that is concurrently withdrawing from distinct accounts. Therefore, instead of a total order across asset transfer transactions, Guerraoui et al. [25] have shown that the semantics of asset transfer only require causal order and some mechanism that prevents double-spending attempts.

This observation unveils a notable opportunity to implement faster cryptocurrencies that escape the high cost of agreeing on a total order. This opportunity has been explored by recent proposals for weakly
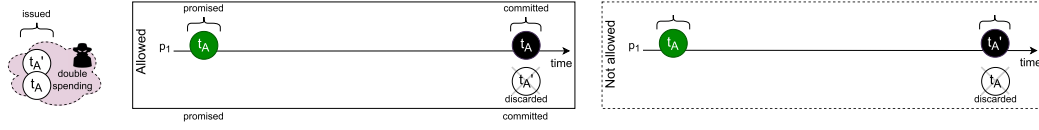
Fig. 1. An example illustrating the *eventually committed upon promised* property. Transaction t_A, which belongs to a double-spending attack (along with t'_A) is promised by correct process p_1. By the above-mentioned property, every correct processes will eventually commit t_A and, consequently, discard its double-spending counterpart, t'_A.

consistent cryptocurrency protocols for permissionless systems, not just by Guerraoui et al. [11, 25] but also by others [38, 54, 60, 63, 64](more details in Section 8).

However, such proposals are not compatible with mainstream permissionless blockchains, whose general-purpose nature requires also serving transactions issued by applications with strong, sequential consistency requirements, such as those issued to/by smart contracts. To the best of our knowledge, bringing weakly consistent cryptocurrency transactions to mainstream permissionless blockchains that also support strongly consistent smart contracts remains an open problem. We address such problem in the next sections.

3 PROMISES

NimbleChain extends the lifecycle of a transaction, t, with an additional event, a *promise of commitment of t*, which we abbreviate to *promise of t*. Whenever a process p is able to anticipate—through some means, as we discuss later—that t will eventually commit (by BBP), p can immediately promise t. The following property formally captures this notion.

Eventually Committed upon Promised Property. *If a correct process promises a transaction t, then all correct process eventually commit t.*

Figure 1 illustrates this property with an example. Process p_1 promises transaction t_A, and thus every correct process (including p_1) will eventually commit t_A.

This example also illustrates that promises resist double-spend attacks, similarly to commits. In our example, a malicious user issues two conflicting transactions, t_A and t'_A in an attempt to double-spend. Recall that BBP ensures that if one of such transactions commits, then the other conflicting transaction(s) will not, with high probability. Hence, since correct process p_1 promises t_A, then that process can immediately infer that, even if a double-spending attempt threathens t_A, the system will eventually commit t_A and, thus, discard any conflicting transactions (t'_A in the example). This implies that an alternative example where the system instead committed t'_A (Figure 1 (right)) is not allowed.

To be useful, promises must be delivered faster than commits. To accomplish that, we hinge on the observation that an important class of applications do not require the strong consistency guarantees of commits. Therefore, promises offer weaker, but still *strong enough*, guarantees for such applications. As we show in Section 5, cryptocurrencies are one notable example that can be implemented by the weak guarantees of promises.

Compared to commits, promises are fundamentally weaker along two dimensions, which Figure 2 illustrates with an example. As a first relaxation, promised transactions are only partially ordered by causal dependencies among them. In other words, promises occur in causal order.

For instance, each process in Figure 2 promises transactions in different orders, which all satisfy the causal dependencies between transactions. In contrast, BBP commits the transactions in total order, as depicted on right-hand side of Figure 2. Causally ordering events in a distributed system is well-studied to be easier than totally ordering them [3], hence implementing promises is an easier problem than the one solved by BBP.

As a second relaxation, correct processes do not need to agree on promising a given transaction. A transaction t may be promised at some correct process, but not necessarily at all other correct process. This relaxation allows
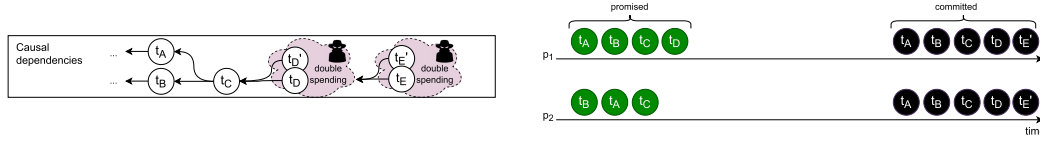
Fig. 2. An example in a 2-process system illustrating how promises are weaker than commits. Only promise and commit events are shown (we omit issue events). The example illustrates the two ways in which the promise order is more relaxed than the commit order: (i) Promise order is partial (more precisely, a causal order), not total, and (ii) not every transaction that commits needs to have been promised at every process.

best-effort implementations that, under worst-case scenarios, can simply give up their efforts to promise some transaction at a subset of processes.

For example, transaction t_D in Figure 2 is promised at process p_1 but not at p_2. Although p_2 does not promise t_D, p_2 later commits t_D. Since commits ensure strictly stronger properties than promises, the commit of t_D properly replaces the missing promise. Concretely, for a client at p_2 with weak consistency demands that wishes to know when t_D has been promised, the fallback is to wait longer until t_D commits (or, instead, is discarded by BBP).

At an extreme scenario, transactions t_E and t'_E (a double-spending pair) are not promised at any process. In this case, processes will eventually learn that one of such transactions has committed (t'_E in our example). This scenario is different than the previous ones because, since no process promises neither t_E nor t'_E, the *eventually committed upon promised* property does not hold and, thus, a correct system is free to agree on either transaction to be committed at every correct process—concretely, an alternative example where every correct process committed t_E would also be correct.

On a related note, suppose that, after t'_E commits at p_2, that process receives a new transaction, t_F, that causally depends on t'_E. By the same rationale as above, we allow p_2 to promise t_F since its causal dependency has already committed (instead of promised) at p_2. Leveraging the above observations, we can finally formulate the (partial) causal order of promises, as follows.

Causal Promise Order Property. *No correct process promises a transaction t before promising or committing t's causal dependencies.*

Hereafter, we assume that transactions carry a metadata field that denotes its causal dependencies (using some suitable causality tracking mechanism). Also, a transaction t issued by correct process p causally depends on every transaction previously issued by p[3].

As a final requirement, which has been implicit so far, we must enforce that BBP's commit order also respects the causal dependencies that determine the causal promise order. This ensures the commit total order is a linear extension of the causal promise orders observed at the different processes. This requirement prevents that the commit order introduces causality anomalies that are absent in the promise order.

Causal Commit Order Property. *No correct process commits a transaction t before committing t's causal dependencies.*

Finally, we remark that, in general-purpose blockchains, different applications or operations within an application, might have distinct consistency demands. This explains why we decided to add the promise event to the original issue/commit model rather than replacing the commit event with the promise event. The richer issue/promise/commit event set offers a hybrid consistency model. It allows different applications, or different operations from the same application, to choose whether to wait for a transaction to be promised or committed, depending on the required guarantees.

---

[3]This is a usual definition in distributed algorithms with causal order [3, 39].
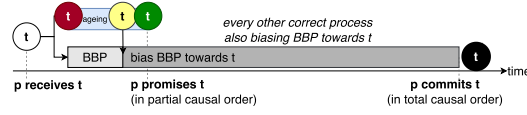
Fig. 3. An example where NimbleChain's ageing fast path promises a transaction, t. Later, the biased BBP slow path eventually commits t, even if t was issued by an attacker that issued a double-spending transaction that is competing with t.

4 NIMBLECHAIN

The main goal of NimbleChain is to provide a fast path that promises transactions much sooner than BBP commits them. To accomplish that goal, we exploit the fact that the *partial* causal order across promised transactions is fundamentally easier to achieve than the *total* causal order that is required across committed transactions.

Besides the promise fast path, NimbleChain still relies on BBP, for two main reasons. First, NimbleChain's fast path is best-effort, since some unlikely scenarios caused by double-spending attempts prevent it from promising specific transactions. To handle such cases, NimbleChain uses BBP to let clients determine whether an unpromised transaction is effectively committed or not. The BBP slow path also serves to provide a total causal order when application semantics have strong consistency demands. Table 1 summarizes the essential differences between both paths.

Next, we present NimbleChain's design. Section 4.1 starts with an overview of NimbleChain. Section 4.2 then details the main algorithm of NimbleChain. Section 4.3 discusses how NimbleChain can be tuned to maximize its resilience, and Section 4.4 details how NimbleChain enforces causal order.

4.1 Overview

In NimbleChain, any transaction that is received by a process (either issued locally or received from another issuer) is fed simultaneously into two modules, which handle the transaction in parallel. We first present the high-level semantics of each module (we defer their algorithms to the following section). Next, we describe how both building blocks are used together to support fast transaction promises. For presentation simplicity, we here assume that transactions have no causal dependencies. We lift this simplification in Section 4.4.

4.1.1 Fast Path. The fast path runs a transaction *ageing* algorithm. As a transaction t ages at some process p, its current age is labelled as a color (red, yellow or green), which we denote by $color_p(t)$. A transaction is initially red (when received at the process) but may later transition to yellow and, finally, to green, as Figure 3 illustrates.

Not all transactions reach the green state; in fact, double-spending attempts may cause some transactions to stop their aging in red or yellow (see Section 4.2). In any case, it takes a short time (considerably shorter than BBP's latency) for any process to determine the final age of a transaction.

The ageing is determined locally at each process, with no cross-process coordination taking place. A consequence is that it cannot guarantee that every correct process reaches the same final age for a given transaction. Instead, it ensures a weaker guarantee:

Bounded Age Consistency Property. *If at least one process, p_i, ages some transaction, t, up to green ($color_{p_i}(t) = green$), then any other correct process p_j must have aged t up to yellow or green ($color_{p_j} \in \{yellow, green\}$).*

4.1.2 Slow Path. The second module consists of BBP augmented with a novel feature, called *biased BBP commit*, which we will describe shortly. Upon receiving some transaction t as input, the BBP's consensus protocol eventually decides whether t commits within the total order of the blockchain or not. Since this takes a long time, BBP constitutes the slow path of NimbleChain.

For any transaction t, each process has the option to *bias BBP toward t*. By default, this option is disabled for new transactions that are received by NimbleChain. If activated by every correct process for a common transaction, then the biased BBP option provides a powerful guarantee:
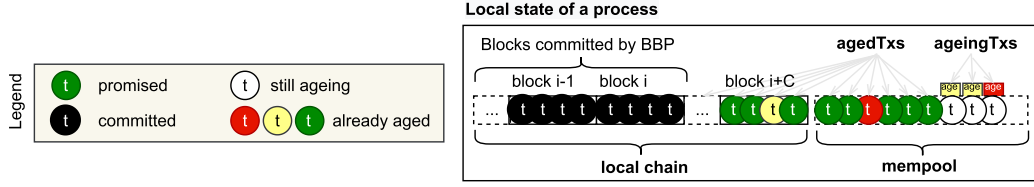
Fig. 4. Local state of a process in NimbleChain.

Biased Transaction Selection Property. *If every correct process biases BBP toward a given transaction t, then BBP will eventually commit t, even if double-spending transactions (of t) have also been issued.*

4.1.3 Putting It All Together. NimbleChain uses the ageing protocol as a best-effort mechanism to trigger promises. When a process p ages some transaction t up to green, p can immediately promise t.

Of course, NimbleChain needs to ensure that, if at least one correct process promises t, then BBP eventually commits t at every correct process. This is not trivial since t may have been issued by an attacker that, concurrently, also issued a double-spending transaction, t'; hence, if no additional care was taken, then BBP might end up committing t' and, thus, discarding t. To address this, NimbleChain connects the ageing protocol and the biased BBP feature as follows: As soon as a process p has aged a transaction up to (at least) yellow, p biases BBP toward t. This is depicted in Figure 3.

From the above-mentioned properties (bounded age consistency and biased transaction selection), one can conclude that this approach ensures that if at least one process p promises a transaction t in the fast path, then t will be eventually committed (by BBP) at every correct process. In fact, if p promises t (hence, t has aged up to green in p), then t will at least age up to yellow at every other correct process. Therefore, every correct process will bias BBP toward t. Consequently, BBP will eventually commit t—even in the presence of a double-spending transactions (conficting with t).

It is worth noting that there are scenarios, caused by double-spending attempts, where a process p cannot promise a transaction t and, therefore, needs to wait for BBP's slow path to learn whether t's outcome. We discuss such scenarios in the next section.

4.2 Main Algorithm of NimbleChain

NimbleChain comprises two main modules, the transaction ageing and the biased BBP commit. Both run on top of an instance of BBP, intercepting specific routines of BBP. We take advantage of the extensible design of BBP, customizing it with relatively simple extensions that affect the incoming transaction and chain validation routines.[4] We provide details on our concrete extension of Ethereum in Section 6.1.

At each process, both modules share the state depicted in Figure 4. This includes BBP's state; namely, a local chain and a mempool. Moreover, each process also maintains each transaction's age (or color), stored in the *agedTxs* and *ageingTxs* sets, which comprises transactions that have already reached their final age, and transactions whose ageing is still ongoing, respectively.

We now present the algorithms of each module.

4.2.1 Fast Path: Transaction Ageing. The transaction ageing fast path runs Algorithm 1. A transaction t can be ageing, be already aged, or in none of such states. The age of a transaction t at a given process is an integer

---

[4]More precisely, inside the *content validation predicate ($V(\cdot)$)* and the *environment* (more precisely, where the *input tapes* of each process are determined) according to Garay et al.'s terminology [20], which are external to BBP and, thus, can be modified without hurting the correctness of the underlying BBP.

ALGORITHM 1: Ageing protocol (NimbleChain's fast path)

```
1  Initially
2  │   ageingTxs ← ∅;
3  │   agedTxs ← ∅;

4  Function age(transaction t):
5  │   (t_aged, ageval) ← lookup(agedTxs, t.issuer, t.seqno);
6  │   if t_aged then
7  │   │   return ageval;
8  │   (t_ageing, ts) ← lookup(ageingTxs, t.issuer, t.seqno);
9  │   if t_ageing then
10 │   │   return (now − ts)/D;
11 │   else
12 │   │   return ⊥;

13 Procedure AgeingMonitor:
14 │   while true do
15 │   │   for each t in ageingTxs do
16 │   │   │   if age(t) = AT then
17 │   │   │   │   remove(ageingTxs, t);
18 │   │   │   │   add(agedTxs, (t, AT));
19 │   │   │   │   promise(t);

20 Upon receiving transaction t:
21 │   t_prev ← lookup(agedTxs ∪ ageingTxs, t.issuer, t.seqno);
22 │   if t_prev = null then
23 │   │   add(ageingTxs, t, now);
24 │   else
25 │   │   if t_prev ≠ t then
26 │   │   │   reject t from mempool;
27 │   │   │   if isAgeing(t_prev) then
28 │   │   │   │   add(agedTxs, (t_prev, age(t_prev)));
29 │   │   │   │   remove(ageingTxs, t_prev);

30 Function color(transaction t):
31 │   switch age(t) do
32 │   │   case ⊥
33 │   │   │   return ⊥
34 │   │   case AT
35 │   │   │   return green
36 │   │   case AT − 2
37 │   │   │   return yellow
38 │   │   otherwise  return red ;
```

---

value, determined by function *age* in Algorithm 1. This value is translated to a color, as we explain later. Before a transaction starts to age, we simply say that its age/color is undefined (denoted by ⊥).

When receiving a transaction t, a process p assigns a local physical timestamp to t and adds t to *ageingTxs* (Line 23, Algorithm 1). From that instant onward, we say that t is *ageing*. While a transaction t is ageing, its age is given by how much time has already passed since t's timestamp, measured in units of D (Line 10). The ageing of t stops in two situations:

(1) When the age of t reaches a system-wide **ageing threshold (AT)** (Line 16). For presentation simplicity, in this section we assume $AT = 4$. In Section 4.3, we show that lower AT values are not appropriate and, more importantly, higher values can achieve higher resiliency under targeted attacks at the expense of latency.

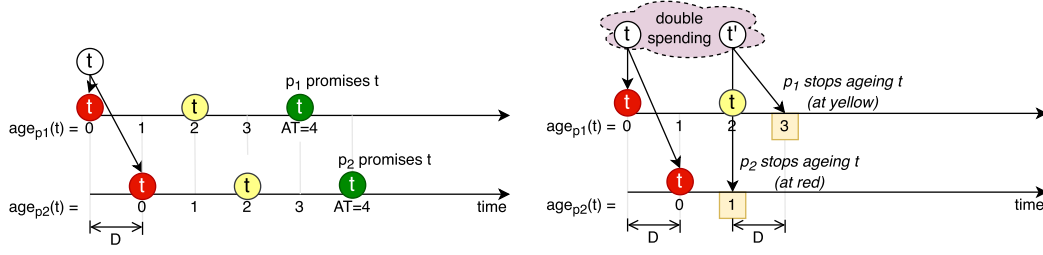(2) When the local process receives a double-spend of t (Line 29).

Fig. 5. Examples illustrating that the age of some transaction t at different processes can diverge up to 2.

In either case, we say that t has aged, and its age is final. In the case of double-spending attempts, where at least two conflicting transactions t and t' are received by a process p, only the first one of them that reaches p is accepted into the mempool and ages at p. The remaining (double-spending) transactions are rejected from the mempool (Line 26) and, consequently, will always have an undefined age ($\perp$) at p.

The age values are translated into a color (Line 30). When a transaction t starts ageing, its color is red. As soon as its age reaches $AT - 2$, its color changes to yellow. Finally, when its age is AT, t's color changes to green and, consequently, the local process promises t (Line 19).

Therefore, transactions issued by correct processes (which, by definition, do not have double-spends) will always age up to green and hence be promised in the fast path at every correct process. In contrast, a transaction t that is issued as part of a double-spend attempt and is the first (among its conflicting transactions) to be received by a process p may age up to red, yellow, or green—depending on how much time it takes until a double-spend t' is subsequently received at p. In any case, the first transaction received at p, t, will be added to the local mempool, and thus p will try to include t in new blocks that p tries to mine; in contrast, p will reject t' from the local mempool, hence p will never mine blocks with t'.

Since transactions are delivered at different times at distinct processes, their final ages or colors are not guaranteed to be consistent across the system. Still, since the behaviour of the broadcast layer is bounded by a maximum delivery delay (D), the potential divergence is bounded and, as we prove next, always meet the *bounded age consistency* property (see previous section).

This is illustrated in Figure 5. It may occur that t ages up to green (thus, promised in the fast path) in some correct processes but only up to yellow in others. Or that t ages up to yellow at some correct processes but only up to red in others. Generalizing these examples, we can formulate and prove the following lemma.

LEMMA 1. *If a correct process p ages a transaction t up to green (i.e., $age_p(t) = AT$), then (a) for any correct process q, t ages at least up to yellow ($age_q(t) \geq AT - 2$); further, (b) for any transaction t' that conflicts with t, then $age_q(t') = \perp$ for every correct process q.*

PROOF. (1) Suppose, by contradiction, that either (a) $age_q(t) < AT - 2$ or (b) $age_q(t) = \perp$; this implies that either (a) q received t' either less than $2D$ after q first received t or (b) that q received t' before t, respectively. Since p received t first and t' more than $4D$ after (since $age_p(t) = AT$), then hypothesis (a) and (b) only occur in executions where t or t' was delivered at q or p (respectively) more than D after they were broadcast by their issuer, which contradicts the assumption of a bounded delivery delay, D. (2) Suppose, by contradiction, that $age_q(t') \neq \perp$. This implies that q received t' before t, which corresponds to hypothesis (b) above, which is impossible under the bounded delivery delay assumption. □

From sub-lemma (a) above, we directly obtain that the proposed ageing algorithm satisfies the *bounded age consistency* property. Due to space constraints, we ommit the correctness arguments for the remainder of this section.

4.2.2 Slow Path: Biased BBP Commit. We now focus on the slow path of NimbleChain, the biased BBP commit feature. As outlined in Section 4.1, the biased BBP commit is activated by each process for a given transaction, t, as soon as a t ages up to yellow (at least). According to the *biased transaction selection* property (see Section 4.1), we expect that if every correct process biases BBP toward t, then BBP eventually commits t.

To implement the biased BBP commit feature, we add two restrictions on BBP's operation. A first restriction is already described in Algorithm 1. From a set of double-spending transactions, only the first one that is received at a process p will be included in p's mempool and aged at p; any subquently received double-spends are rejected from p's mempool.

We now introduce a second restriction, which affects BBP's CSR and is shown in Algorithm 2. For each new block b_i in a new chain that a process p receives, p checks whether block b_i has any transaction that is a double-spending of another transaction t that p had already received. If so, the age of t (at p) determines whether the offending block b_i can be accepted or must be rejected. If t has aged at least to yellow at p, then b_i is rejected, together with any subsequent blocks.

ALGORITHM 2: Biased BBP commit (NimbleChain's slow path)

---

1 **Upon receiving** *chain $b_0, \ldots, b_n$*:
2   **for** *each $b_i (0 \geq i \geq n)$ not yet in the local chain* **do**
3   **for** *each transaction $t_i \in b_i$ that conflicts with a previously received transaction t* **do**
4   **if** RRS(t) > $n - i$ **then**
5   reject chain suffix $b_i, \ldots, b_n$;

```
/* Simplest implementation                                                          */
```
6 **Function** RRS(*transaction t*):
7   **if** $color(t) \in \{yellow, red\}$ **then**
8   **return** C;
9   **else**
10   **return** 0;

```
/* Progressive variant (see Section 4.3)                                            */
```
11 **Function** RRS(*transaction t*):
```
        /* Assuming AT = 2(C + 1)                                                    */
```
12   **return** $\lfloor age_p(t)/2 \rfloor$;

There is, however, an exception to this rule. When b_i is followed by at least C blocks in the received chain, NimbleChain no longer rejects b_i. We present the rationale behind this exception shortly. Algorithm 2 captures the above rule (and its exception) with the **required replacement suffix (RRS)** function. This function returns C for those transactions that the local process has aged to at least yellow and 0 otherwise.

To illustrate, let us consider the first example in Figure 6 (top). In this example, some correct processes (such as p_A) promise t, as soon they age t up to green (③). From the *bounded age consistency* property, we know that even if, due to a double-spending transaction t', other correct processes (such as p_B) might not age t up to green, they are guaranteed to age t up to yellow (④). Therefore, at that point, every correct process (either p_A or p_B) biases BBP toward t. According to the restrictions presented above, this means that

(1) t has been included in every correct processes' mempool, thus they are all trying to mine a new block with t (unless a block with t is already in their local chains), and any double-spend of t is excluded from their mempools; and

(2) all correct processes are rejecting any chains that contain a double-spend of t *unless the block containing such double-spend is suffixed by at least C blocks.*

Together, both observations imply that the whole mining power of correct processes will be used toward appending t to their chains and continuing building on a chain with t. The probability that an attacker is able to generate a chain containing a double-spending of t followed by at least C blocks is arbitrarily low. Therefore, the
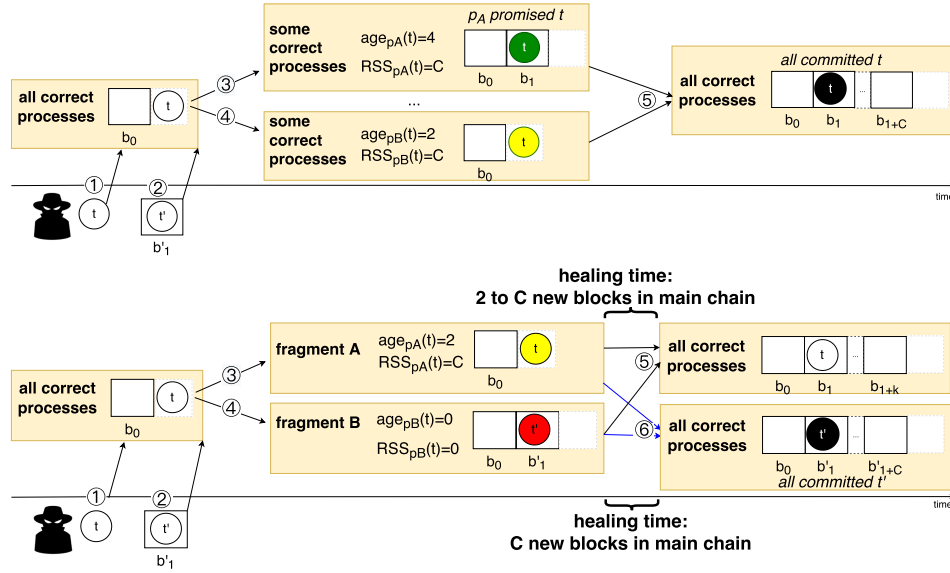
Fig. 6. Example of NimbleChain handling a transaction that is part of a double-spending attempt, with $AT = 4$. Top: transaction ages up to green (thus, is promised) in at least one correct process. Bottom: transaction does not age up to green in any correct process.

correct processes biasing BBP toward t will never replace the chain with t, which they are collectively building, by a fork with a double-spend of t. Hence, every correct process will eventually commit t by BBP.

We also need to take other scenarios into account. These are illustrated in Figure 6 (bottom). Here, different correct processes only age t up to either red or yellow. Those correct processes that age t up to yellow, such as p_A (③), bias BBP toward t, thus reject chains with any double-spending transaction t' since $RRS(t) = C$. In contrast, other correct processes that only age t up to red, such as p_B (④), remain open to accept chains with either t or t', since $RRS(t) = 0$.

This situation can be problematic if the Byzantine process that issued the double-spend further manages to extend the current main chain with a new block comprising t'. In this situation, processes like p_A will reject such new chain, while processes like p_B will accept it (by the longest chain rule of BBP).

We call this a *fragmentation attack*. It introduces an artificial fork that divides the mining power of correct processes across two *fragments*, denoted A and B in Figure 6. While the fragmentation persists, the attacker's mining power will be temporarily closer to the largest correct fragment's mining power, which may harm the robustness of BBP.

To mitigate the impact of a fragmentation attack, NimbleChain is able to self-heal upon any successful fragmentation attack. To understand how, let us again focus on the bottom of Figure 6. A first scenario (⑤) is when the fragment of correct processes biasing BBP toward t are able to grow their chain faster than the opposite fragment (holding a chain with t'), but not vice-versa. This fragment will be naturally healed by BBP's longest chain selection rule as soon as the main chain grows to be larger than the opposite fragment's chain.

The second scenario (⑥), however, may take longer to heal. In this case, the fragment holding a chain with t' happens to have the majority of correct processes. Therefore, it is likely that this chain will tend to grow faster than the chain with t. This is where the C bound on RRS becomes useful. The fragment holding a chain with t will accept the chain with t' as soon as it becomes suffixed by C blocks, which heals the fragmentation attack. It should be noted that, in the scenario we are considering, no correct process has promised t. Therefore, the system of correct processes can safely converge toward committing t' and, thus, discard t.
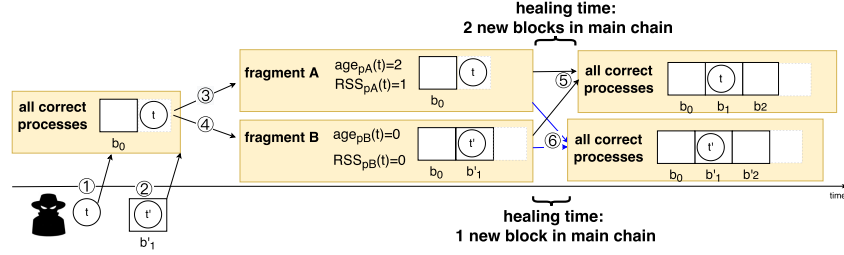
Fig. 7. Example of NimbleChain healing from a fragmentation attack, with its most robust configuration ($AT = 2(C + 1)$).

4.3 Trading Fast Path Latency for Robustness

The previous section showed that, with the minimal $AT = 4$ value, a single fragmentation attack may require waiting up to C block generation rounds to heal. In this section, we explain how NimbleChain can be configured to substantially mitigate the chances of success of the above attack. The key insight is that, by increasing AT, NimbleChain can reduce the time it takes to heal fragmentation attacks from C to the time it takes for the fastest fragment's chain to generate 2 blocks, with high probability. This improved robustness comes at the cost of a higher fast path latency, since a higher AT implies that transactions take longer to age up to green.

Before describing how NimbleChain can reduce the above vulnerability window, let us recall the two fragmentation scenarios in Figure 6, which assume $AT = 4$. In both scenarios, every correct process has received transaction t and an attacker is mining an alternative chain that contains a double-spend transaction t'. While some correct processes will accept the alternative chain as long as it meets the standard BBP requirements, others will be more reluctant and impose the additional constraint that, in the alternative chain, the block with t' must be followed by at least C blocks. As shown in Figure 6, with $AT = 4$, $RRS_{p_i}(t)$ may differ between 0 and C for different correct processes. Intuitively, this upper bound on the divergence across $RRS_{p_i}(t)$ directly determines the time—in terms of block generation rounds—that a fragmentation attack may take to heal.

If we can lower this upper bound, ideally down to 1, then we can minimize the healing time. We achieve this by extending the algorithm described in the previous section with a *progressive* variant (Line 11, Algorithm 2) where each process gradually increments RRS in $2D$ steps. More precisely, we define $RRS_p(t) = \lfloor age_p(t)/2 \rfloor$. To enable this progressive variant, while ensuring that p only promises transaction t once every correct process p_i is biasing BBP toward t (i.e., $RRS_{p_i}(t) = C$), we need to redefine $AT = 2(C + 1)$.

As an example, suppose that process p receives a transaction t. Initially, $RRS_p(t) = 0$. If p does not observe any double-spend of t, then $RRS_p(t)$ will grow to 1, 2, . . . every $2D$, until it reaches $RRS_p(t) = C$ once $age_p(t) = C \times 2D$. Finally, $2D$ later, t ages up to green at p and, hence, p promises t (at $age_p(t) = 2(C + 1)$).

To understand the impact of this new configuration, Figure 7 revisits the fragmentation scenarios from Figure 6. As before, a successful fragmentation attack has split the set of correct processes into two divergent fragments: fragment A, whose processes hold t in their local ledger and now have $RRS_{p_A}(t) = 1$ (instead of $RRS_{p_A}(t) = C$ as per the original formulation), and fragment B, whose processes have $RRS_{p_B}(t) = 0$ and, consequently, have accepted a chain with transaction t', a double-spend of t, which the attacker produced and disseminated. On the one hand, a process in fragment A does not accept the chain held by fragment B, since it holds a block with a double-spend of t that is not suffixed by at least one block (due to $RRS_{p_B}(t) = 1$). On the other hand, a process in fragment B will not accept the chain of fragment A since it has a lower height than fragment B's main chain (due to the longest-chain rule).

Let us first consider the scenario where fragment A is faster in extending its chain (⑤) than fragment B. In this case, as soon as A's chain grows 2 blocks, processes in B will start accepting that (according to the longest-chain rule). Let us, instead, analyse the reverse situation, where fragment B's chain grows faster (⑥). The fragmentation may now heal as soon as fragment B appends *a single* new block to its chain. These represent

Table 3. Trade-off between Promise Performance and
Fragmentation Healing Time in Function of AT

AT	Upon ageing to green		Upon fragmentation
	Promise latency	Speedup (vs. BBP commit)	Healing time
4	$4D$	$C/(4r)$	up to $C \times B$
$2(C + 1)$	$2D(C + 1)$	approx. $1/(2r)$	up to $2B$

The speedup is relative to BBP's commit latency of $C \times B$. The D/B ratio is
denoted by r, which, by definition, is expected to be small [20].

substantial improvements in healing time relatively to the $AT = 4$ configuration, which could take up to C blocks to heal in worst-case scenarios.

We remark that these upper bounds are also observed in concurrent scenarios, where more than one block is produced either at the same fragment or across different fragments. Furthermore, we highlight that the 1 or 2 blocks needed to heal a fragmentation can be produced by any process in either fragment, which means that the healing speed is determined by the aggregate resources of the correct nodes (i.e., not restricted to the largest correct fragment's resources). These upper bounds on the healing time can be generalized to scenarios where (i) the fragmentation attack occurs when t had a higher age; and, (ii) the fragmentation attack simultaneously aims at multiple double-spending transactions.

Summing up, NimbleChain can be configured between two extremes and offer a tradeoff between fast path latency and robustness against fragmentation attacks. Table 3 summarizes the tradeoff between both extremes of AT.

4.4 Ensuring Causal Order

We now describe how NimbleChain handles causal dependencies consistently across the slow and fast paths, with the goal of ensuring promise and commit causal order (Section 3). We hereafter assume that causal dependencies are specified in an additional transaction field using some suitable causality tracking mechanism.

Let us start by addressing promise causal order. This is easy to ensure by adding a simple condition that checks causal dependencies before a transaction is promised upon ageing (Line 19, Algorithm 1). Concretely, when a transaction t that has aged up to green at a given process, that process checks every causal dependency of t before promising t. If every such causal dependency is already promised or committed at that process, then it promises t. Otherwise, t is temporarily held in a local set of green transactions that still have causal dependencies awaiting to be promised/committed.

We also need to enforce commit causal order, which we achieve by extending NimbleChain with two additional restrictions to how each process manages its local chain.

The first restriction requires that the mempool comprises two distinct queues: a *ready* and a *pending* queue. When a given process p receives a transaction t and is about to be included in the local mempool, an additional validation procedure will check every causal dependency of t. If at least one causal dependency of t is still absent from the *ready* queue at p, then t is inserted in the *pending* queue. Otherwise, t is added to the *ready* queue, and any transactions in the *pending* queue that causally depend on t have their causal dependencies re-evaluated and, accordingly, moved to the *ready* queue if t was their last missing dependency. When generating blocks, p will only select transactions from the *ready* queue. Therefore, any transaction in a block generated by a correct process is preceded (in the corresponding chain order) by its causal dependencies.

As a second restriction, causal dependencies are checked for transactions in chains received from processes. If the causal dependencies are not satisfied in a received chain, that chain is discarded.

It is easy to show that the two above restrictions ensure that any transaction t in a process' local chain is ordered after every causal dependency of t. Consequently, the committed prefix is causally ordered, which implies that transactions commit in causal order.

We remark that, while the algorithm described in the previous sections ensured that every transaction issued by a correct process would be promised at every correct process, this no longer holds when transactions have causal dependencies. As an example, suppose that a correct process p has promised a transaction t_1 that was issued by a Byzantine process as part of a double-spend attempt; and, after observing t_1, p issues t_2, which causally depends on t_1. Recall that, since t_1 is part of a double-spending attempt, t_1 may not be promised at some (or all) process. In that case, some (or all) processes will only promise t_1. Hence, even though t_2 was issued from a process that had promised t_1 via its fast path, the processes that had not done so will delay t_2's commit until the outcome of t_1 is determined by the slow path.

Finally, we note that, even though Nakamoto consensus defines a total order of transactions, it does not prescribe any specific transaction order to miners. In particular, miners are free to select which transactions to include in a block and the order in which they appear in the block. Our approach skews the order in which transactions are included in a block such that this order satisfies the causal order defined by the promises. But because Nakamoto consensus is itself oblivious to the concrete order of transactions within a block, our approach does not affect the correctness of the original protocol.

5 LOW-LATENCY CRYPTOCURRENCIES WITH NIMBLECHAIN

As discussed in Section 2.4, a notable application that can benefit from the weaker guarantees of the promise event provided by NimbleChain is a cryptocurrency. We now detail how.

A cryptocurrency can be abstracted as an instance of the asset-transfer object type defined by Guerraoui et al. [25]. An asset transfer object maintains a set of accounts, where each account is associated with an owner client that is the only one allowed to issue transfers withdrawing from this account. To do so, the owner client of an account a can invoke a *transfer(a,b,x)* to transfer x from account a to account b. There is a second operation, *read(a)*, which every process can invoke to read the balance of account a.

Traditional permissionless blockchains, such as BBP, implement the asset-transfer object type by relying on the consensus-based commit event, as follows.

- **transfer(a,b,x)**. When the process that owns account a wishes to execute *transfer(a,b,x)*, it reads the current balance of a (see next) and, if the balance is sufficient, issues a new transaction whose payload transfers x from account a to account b.
- **read(a)**. The *read(a)* operation is implemented by returning a's balance from the state that results from the ordered execution of every committed transaction in the local chain.

Guerraoui et al. [25] prove that, in fact, the asset-transfer object type can be correctly implemented in a consensusless fashion. They also present (and prove correct) an actual consensusless implementation of the asset-transfer object type based on message passing. The algorithm they propose relies on a secure broadcast layer that exposes a *broadcast* and a *deliver* event (for messages), while offering uniform reliable delivery with *source order* guarantees despite Byzantine faults. The complete algorithm by Guerraoui et al. defines which state each process maintains to know, which outgoing transactions have been issued by that process, as well as which incoming transactions have been delivered and validated at that process. Further, it defines how, based on that state, the causal dependencies field of a newly issued transaction can be efficiently encoded.

We can port their approach to NimbleChain by replacing the underlying secure broadcast layer with NimbleChain. Concretely, by simply replacing the *broadcast* and *deliver* events in Guerraoui et al.'s algorithm with the *issue* and *promise* events of NimbleChain's interface. The key insight to this transformation is that the properties that Guerraoui et al. require from the secure broadcast layer (namely, integrity, agreement, validity and source order [25]) are also satisfied by NimbleChain's promises with high probability. We prove this later in this section.

Below, we present a high-level summary of the algorithm that results from porting Guerraoui et al.'s to rely on NimbleChain's interface. For lower-level details, we refer the reader to Reference [25].

- ***transfer(a,b,x)***. When a processes *p* that owns account *a* executes the *transfer(a,b,x)* operation, it confirms that account *a* has enough funds and, if so, issues a transaction *t*, whose payload describes the requested operation. Further, the causal dependencies field of the new transaction *t* is the set of transactions comprising: (i) every previously issued outgoing transfer transaction (i.e., transferring funds from *a*) and (ii) every incoming transaction (i.e., transferring funds to *a*) already promised by process *p*.
- ***read(a)***. The *read(a)* operation is implemented by returning *a*'s balance from the state that results from the ordered execution of every *promised* transaction in the local chain.

Recall that, for most transactions, NimbleChain's fast-path ensures that most correct processes are able to promise such transactions much sooner than the time that BBP slow-path takes to commit them. Therefore, the above implementation of the asset-transfer object achieves important latency improvements.

The above promise-based implementation is correct according to the specification of the asset-transfer object type [25]. In other words, the above implementation correctly supports a cryptocurrency. The following lemma states this.

LEMMA 2. *The proposed promise-based implementation of* transfer(a,b,x) *and* read(a) *is a correct implementation of an asset-transfer object type [25].*

PROOF. To show that the above implementation is equivalent to Guerraoui et al.'s message passing asset-transfer object implementation, which was originally proposed and proved correct in Reference [25], we prove that NimbleChain's issue and promise events satisfy, with high probability, the properties of the secure broadcast layer underlying Guerraoui et al.'s implementation. Next, we take the properties of the broadcast layer that underlies Guerraoui et al.'s algorithm (namely, integrity, agreement, validity and source order [25]) and reformulate them by renaming the *broadcast* and *deliver* events by the *issue* and *promise* events. Then, we prove that the resulting properties are satisfied by NimbleChain.

Integrity: A correct process promises a transaction t from a process p at most once and, if p is correct, only if p previously issued t. This is ensured since transactions in BBP are digitally signed and carry a unique identifier.

Agreement: If processes p and q are correct and p promises t, then q promises t. Let us recall that if a correct process *p* promises some transaction *t*, then *p* eventually commits *t* (by the *eventually committed upon promised* property). Hence, any other correct process *q* also eventually commits *t* (by BBP's persistence property), thus, by definition of promise, *q* also promises *t*.

Validity: If a correct process p issues t, then p promises t. This is ensured since any transaction *t* issued by a correct process *p* is eventually committed by *p* (by BBP's liveness property) therefore, by definition of promise, *p* also promises *t*.

Source order: If p and q are correct processes and both promise transactions t and t′, both issued by the same process r, then they do so in the same order. Let us first consider the case where *t* and *t′* have distinct sequence numbers. Therefore, the promise causal order property guarantees that both *p* and *q* will promise both transactions in order defined by their sequential numbers. Let us, instead, assume by contradiction that *t* and *t′* have the same sequence number and are promised by *p* and *q*. Therefore, by the *eventually committed upon promised* property, *p* and *q* will eventually commit both transactions. Still, by definition, distinct transactions with the same sequential number are conflicting and, therefore, at most one can commit. This contradicts the initial assumption. □

We conclude with two final remarks. First, although Guerraoui et al.'s algorithm was originally proposed in the context of permissioned systems, adapting it to exploit the primitives of NimbleChain enables it the work in permissionless environments. Furthermore, while the original proposal supported a stand-alone cryptocurrency system, the above adaptation to NimbleChain integrates the low-latency cryptocurrenty in a richer ecosystem where other applications with stronger consistency requirements may also co-exist. For instance, this hybrid consistency ecosystem enables processes to issue smart contract transactions (via the issue/commit interface),

Table 4. Summary of Experimental Parameters

Number of processes	500
Mining power	24.0%, 21.3%, 13.2%, 12.1%, 5.7%, 1.9%, 1.8%, 1.5%, 1.4%, 1.3%, 1.1%, 1.0%, 1.0%, and 0.026% for each remaining process
Average block generation time (B)	20 s
Average transaction generation time	125 ms
Commit threshold (C)	12
Average message delivery delay	120 ms
Maximum delivery delay (D)	960 ms

whose execution costs are charged from cryptocurrency accounts that may receive incoming transfers as defined above (via the issue/promise interface).

6 EVALUATION

In this section, we evaluate NimbleChain with the goal of answering the following questions: (i) what are the latency improvements that NimbleChain brings to applications with different consistency needs and (ii) what is the impact of fragmentation attacks?

As we further detail below, our experimental evaluation is performed in an environment as close as possible to a real deployment. Namely, it relies on the real code of the reference Geth implementation of Ethereum [16], which we extend to implement NimbleChain, running in a system of 500 processes. Adopting a similar methodology as related works (e.g., References [19, 41, 71]), the workload is derived from a real trace of Ethereum, PoW mining is simulated, and a realistic geo-distributed network is emulated using a state-of-the-art emulator [24].

Next, we detail the evaluation scenario, metrics and discuss our results.

6.1 Implementation, Deployment, and Workload

We implemented NimbleChain as an extension of Ethereum using the reference Geth implementation [16]. The implementation required ≈3,000 new lines of code to implement NimbleChain and changing ≈300 lines of code in Geth.

For experimental purposes, we also developed a custom client that, using the regular API, injects transactions in the system according to real transaction traces from Etherscan [18] (a sample of transactions from block 5306612 up to block 6222336). The workload has no information about causality (aside from the implicit dependencies among transactions issued from the same account) and hence we set each transaction to depend on the most recently promised transaction and, transitively, from the causal dependencies of that transaction. Note that this is a conservative choice, since it increases the probability that some processes will have to wait for causal dependencies.

Table 4 summarizes the main experimental parameters. To run experiments with a large number of processes we replaced the PoW component with a probabilistic mining selection process that follows a Poisson distribution and mimics the block production distribution. To reflect the non-uniform mining power distribution of today's mainstream permissionless blockchains, we allocated the estimated mining power of the top-13 most powerful mining pools of Ethereum to a subset of 13 processes, according to Reference [18] (as detailed in Table 4). The remaining mining power was uniformly distributed across the remaining 487 processes.

We adjusted the block production probability to mimic Ethereum's rate of three blocks per minute [18]. Moreover, we extended both NimbleChain and Ethereum implementations to log transaction events such as generation, reception, dissemination and block events (such as insertion to the local chain and forks) to allow *a posteriori* offline processing for our evaluation. We use the same codebase, client and PoW component for the NimbleChain
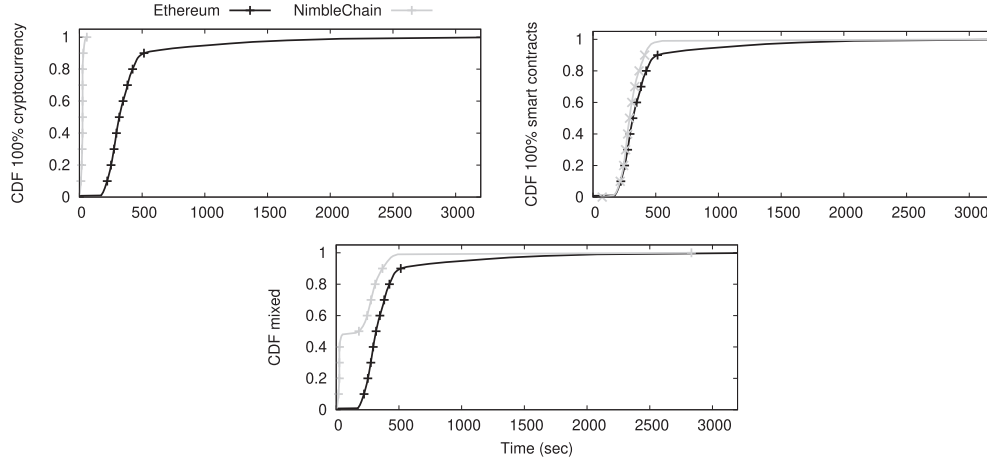
Fig. 8. Transaction latency: 100% cryptocurrency transactions (above); 100% smart contract transactions (middle); mixed cryptocurrency and smart contract transactions.

and Ethereum. Every correct process is parameterised with $C = 12$, the current standard commit threshold in Ethereum [70].

We ran each experiment for one hour with 500 processes for both NimbleChain and Ethereum, using five machines equipped with a mix of Intel Xeon CPUs. We empirically found this configuration of machines to be able to accommodate 500 processes without becoming overloaded and hence compromising the fidelity of the results. The processes run on an emulated network using Kollaps [24] with internet latencies that model the geo-distributed nature of permissionless blockchains. As suggested by previous measurement studies [22, 61], we used an average latency value of 120 ms, and a conservative value for $D = 8 \times 120$ ms $= 960$ ms. Each node started with the same local chain, consisting of a single genesis block. We injected eight transactions per second, as common in Ethereum [18]. All results are the average of 5 runs.

6.2 Promise and Commit Latencies

In this section, we study *transaction latency* as perceived from a process p, which we define as the time from the moment a given transaction t is issued (at some process, not necessarily p) and the moment p triggers the event that is required by the application semantics associated with t (i.e., either promise or commit). We evaluate transaction latency for two transaction types, with distinct consistency needs: cryptocurrency transactions, which only require promises, and smart contract transactions, which need to be committed.

We consider three scenarios: (i) 100% cryptocurrency transactions, (ii) 100% smart contract transactions, and (iii) and a mixed ratio of 44% cryptocurrency and 66% smart contract transactions as observed in Ethereum [61]. We evaluate these three scenarios considering no Byzantine behaviour. (We study Byzantine behaviour in the next section.)

We consider NimbleChain configured with $AT = 2(C + 1)$, the most robust configuration. For brevity, we do not evaluate NimbleChain configured with $AT = 4$, which is a less robust configuration that would present even lower values for promise latency.

Figure 8 presents our results. As expected, the average transaction latency for cryptocurrency transactions is around one order of magnitude lower with NimbleChain (promise latency) than with Ethereum (commit latency), as shown in Figure 8 (top).

Smart contract transaction commit times with Ethereum are very similar to NimbleChain's, which suggests that the overhead of NimbleChain on the underlying protocol is negligible, as shown in Figure 8 (middle).

NimbleChain seems to slightly outperform Ethereum at the head, due to slightly different transaction ordering criteria in the mempool. As an example, causal dependencies are ordered before a dependent transaction regardless of their price. Hence, such dependencies may reach the blockchain earlier in NimbleChain than in Ethereum.

Figure 8 (bottom) depicts the mixed scenario and shows an interesting trend. Cryptocurrency transactions continue to perform much faster in NimbleChain (promise latency) than in Ethereum (commit latency), while smart contracts commit at roughly the same pace with the results showing a clear inflection point between each transaction type.

6.3 Fragmentation Attacks

To perform a fragmentation attack for some period T, an attacker must allocate a portion x of its mining power to that purpose during T. As a return from such investment, the attacker will reduce the average mining power of the largest correct fragment by y during T. To be profitable to the attacker, y must be higher than x. Otherwise, a rational attacker will not have any tangible benefit, and thus will not carry on the attack. The main goal of this section is to study the impact of fragmentation attacks, and consequently answer the above question.

Recall that a fragmentation attack involves broadcasting a transaction t and immediately starting to produce a block with a double-spend t'. Later, the attacker broadcasts t' in an attempt that different correct processes will receive it at different ages of t. This will result in some correct processes choosing $RRS_p(t) = a$ and others $RRS_q(t) = a + \delta$, where a denotes the age that t had when the former received t' and δ depends on the AT parameter (as discussed in Section 4.3).

We empirically found that the most advantageous case for the attacker was with $a = 0$, i.e., some correct processes p receive t' when $age_p(t) = 0$, while other processes q receive t' when $age_q(t) > 0$ (i.e., either 1 or 2). Hence, in this analysis, we consider $a = 0$.

As detailed in Section 4.3, the divergent fragments of correct processes eventually converge when at most two new blocks are generated from either fragment (and delivered), when $AT = 2(C + 1)$. In contrast, when $AT = 4$, the divergent fragments of correct processes may require C blocks to converge. In the meantime, the attacker may perform a burst of consecutive fragmentation attacks with the intention to keep the system fragmented for a longer period, by creating a new fragmentation before the current one is healed.

In these experiments, we assumed the attacker holds 24% of the resources, which corresponds to the largest mining pool in Ethereum [61]. The attacker continuously performs a fragmentation attack to weaken the aggregate resources owned by correct processes, by making them adhere to distinct fragments. The attacker uses its total mining power to perform the fragmentation attack series ($x = 24\%$). For brevity, we omit scenarios where the attacker only spends a fraction of its total mining power ($x < 24\%$). Our analysis of lower values of x yielded similar observations as the ones that we present next.

Further, we artificially instrumented the delivery protocol to ensure a *fragmentation ratio* that is favorable to the attacker: Eighty percent of nodes randomly considered $RRS_p(t_1) = 0$ and could accept a malicious block immediately, while the remaining 20% considered $RRS_p(t_1) = 1$ for $AT = 2(C + 1)$ or $RRS_p(t_1) = C$ for $AT = 4$. In a real scenario, it would be very unlikely for the attacker to be able to reach such precise fragmentation, since he does not control the network latency. We tested other values for the fragmentation ratio (e.g., 50–50%, 70–30%, and 60–40%), which yielded similar conclusions and hence we omit those results.

Figure 9 shows the results as stacked histogram. The y-axis expresses the percentage of resources owned by each fragment and the x-axis represents time, in seconds. The light colored layer represents the percentage of resources owned by honest processes in the largest (i.e., with the highest mining power) fragment at a given moment. It is worth noting that, even in the baseline, the mere generation of a new block by a correct process (with or without a concurrent fork) causes short periods where the correct processes are not perfectly synchronized (i.e., in the same local chain state), which quickly end as the system synchronizes again. Note that this is intrinsic to BBP.
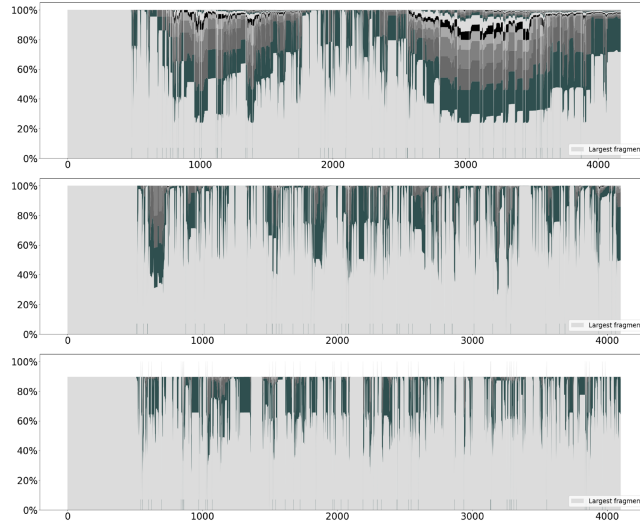
Fig. 9. Fragment adoption during a fragmentation attack, comparing NimbleChain with $AT = 4$ (top) and $AT = 2(C + 1)$ (middle) against the Ethereum baseline (bottom, no fragmentation attack) The y-axis expresses the percentage of resources owned by each fragment of correct processes, where 100% in the same colour (light grey) means that every correct process' local chain is identical. The x-axis represents time, in seconds.
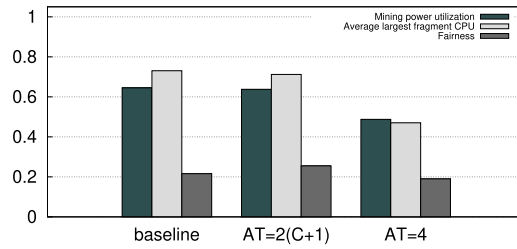

Fig. 10. Mining power utilization, average largest fragment CPU and fairness upon fragmentation attack.

As expected, the results for $AT = 4$ (Figure 9 (top)) show a long-lasting fragmentation since processes require a long suffix of C blocks before accepting a chain with t'_1. For $AT = 2(C + 1)$ (Figure 9 (middle)), we observe a much faster healing since processes require only 1 block before accepting a chain with t'_1. This confirms that $AT = 4$ trades a decreased latency in the fast path for less robustness against fragmentation attacks, while $AT = 2(C + 1)$ provides more robustness at the expense of a higher latency in the fast path.

To conclude our analysis of the fragmentation attack, we evaluate its impact on two metrics originally proposed by Eyal et al. [19]. The **mining power utilization (MPU)** is the ratio between the aggregate work of the main chain and all produced blocks. Fairness is the ratio between the number of blocks generated by the largest honest miner and all produced blocks. In a fair system, the fairness ratio should be identical to the mining power of the reference miner, which holds 21.3% mining power in this experiment (corresponding to the second most resourceful miner). We also measure the average CPU power of the largest fragment.

The results are presented in Figure 10. As it is possible to observe, the MPU decreases from its baseline value when an attack is carried out. Further, MPU decreases as AT decreases. The average CPU owned by the largest

fragment during the attacks also decreases as AT decreases. These are expected results, since the mining power is scattered among fragments.

Most importantly, the decrease in both metrics is considerably lower than the mining power the attacker invested to put the attack in practice. This is true even for the least robust variant of NimbleChain, $AT = 4$. This means that the attacker does not reach a *break even* point. Therefore, the analyzed attack is not profitable for a rational attacker. Instead of using his mining power to slow down the correct system's ability to advance the main chain (through fragmentation), it would be more profitable for the attacker to employ the same mining power to accelerate the generation of his malicious chain (an attack vector that is possible in standard BBP).

Finally, and as expected, the fragmentation periods cause *fairness* deviations in NimbleChain. While vanilla Ethererum is the closest to the desired fairness target (ideally, 21.3%), the fairness of NimbleChain's variants are still within a 10% distance from the ideal target.

7 DISCUSSION

Assumptions on network propagation. The design of NimbleChain depends on the assumption of a well-known maximum delivery delay, D. Of course, NimbleChain may behave incorrectly if the D assumption is not met by the underlying network. More precisely, a period of arbitrary propagation delays (beyond D) can violate our assumption that the age and RSS that two correct processes assign to some transaction do not diverge by more than 2 and 1 (respectively). An arbitrary divergence across the ages each process sees may lead to pathological situations where two double-spending transactions are able to successfully age at distinct processes. Further, an arbitrary divergence in RSS may compromise the ability of NimbleChain to heal upon a fragmentation attack.

We remark that a maximum network propagation time is a standard assumption in permissionless blockchains. For instance, every permissionless blockchain mentioned in Section 8 relies on this assumption. Further, different works have studied vulnerabilities that are possible if the D assumption is violated. For instance, an unexpectedly high block propagation delay may slow down the time the system converges upon forks, which provides an advantage to a resourceful attacker to temporarily benefit from lower mining power utilization from the correct processes [20]. An eclipse attack [48] can isolate a subset of correct processes from the remaining correct system—which can be translated to arbitrarily increasing D from the outside to the eclipsed partition—allowing a resourceful attacker to control the evolution of the blockchain within the partition.

Still, we acknowledge that NimbleChain might be more sensitive than BBP to smaller sporadic violations of D, and/or smaller periods where D is violated, during which NimbleChain's properties are violated but BBP's hold. This is a natural consequence of a protocol that reaches a decision over a shorter time window. This consequence is shared with every proposal in Section 8 that provides lower commit latencies than BBP.

Performance with higher delivery delays. In absolute terms, NimbleChain depends on D being low enough to ensure a low fast path latency. Hence, one may rush to the conclusion that the speedup that NimbleChain introduces relatively to the baseline BBP-based permissionless blockchain depends on D. However, that is not correct. Let us suppose that NimbleChain was used in a low-quality network whose D was much higher than the one that previous studies find in real permissionless blockchains [34, 35]. In that case, the PoW difficulty (and, hence, the block generation time, B) would need to be adjusted accordingly, to keep the rate D/B low enough to ensure the (probabilistic) correctness of BBP. This readjustment of D and B would not only increase the fast path latency of NimbleChain, but also the commit latency of BBP. Therefore, the speedup of NimbleChain would remain the same. This conclusion is in line with Table 3, which shows that the speedup of NimbleChain depends on f and C, not on D.

8 RELATED WORK

In recent years, many proposals have arisen that improve the performance of permissionless blockchains based on Nakamoto consensus. In the vast majority of such proposals, meaningful consistency guarantees are only

provided once a block is followed by enough (i.e., C) blocks in the ledger, similarly to Nakamoto's original proposal. Therefore, such proposals do not escape the well-studied lower bound of commit latency in Nakamoto consensus [20]. Instead, they provide improvements to throughput and/or energy efficiency, but not on commit latency. In contrast, NimbleChain offers a hybrid consistency model in which weaker consistency guarantees are provided by a new promise event, enforced by a different protocol that coexists with Nakamoto consensus. Consequently, promise latency is not dictated by the lower bounds of Nakamoto consensus' commit latency.

In this section, we start by surveying related work that improves permissionless blockchains based on Nakamoto consensus. Later, we describe work on more disruptive approaches that, while falling outside the domain of Nakamoto-based permissionless blockchains, are related to our work.

Improvements over longest chain rule. GHOST [66], partially implemented in Ethereum [70], improves Nakamoto's original longest chain rule by allowing all blocks generated by honest participants to contribute to the commit of the main chain. This enables convergence even with higher block generation rates. A different approach, followed by inclusive blockchain protocols [40] and PHANTOM [65], organizes blocks as **directed-acyclic graphs (DAG)** of blocks instead of a totally ordered list to optimize performance. More recently, Conflux [42] combines the main principles behind GHOST and DAG-based solutions in an adaptive fashion to provide them with better liveness guarantees. Despite the significant throughput gains (e.g., Conflux is able to improve GHOST's throughput by 32×), these approaches still rely on consensus as the only path to commit. Consequently, they only bring modest commit latency savings (e.g., 25% latency gains in Conflux with respect to GHOST). NimbleChain can be plugged to any of these systems and enhance them with substantially lower commit latencies, while retaining their throughputs.

Hierarchical and Parallel Chains. Alternative blockchain organizations include hierarchical and parallel chains. In Bitcoin-NG [19], key blocks are generated at a similar rate as Bitcoin. Still, in-between two key blocks, the miner of the previous key block can generate many microblocks that contain transactions. FruitChain [56] adopts a similar hierarchical approach. OHIE uses parallel instances of BBP and then deterministically sorts blocks to reach a total order [71]. In all these proposals, the total order of the main chain is determined by only a fraction of blocks (key blocks). Hence, the remaining blocks (microblocks), which carry the actual transactions, can be safely generated at much higher rates than BBP allows, thus increasing throughput. Unlike NimbleChain, these proposals focus on improving throughput, not commit latency. For example, Bitcoin-NG does improve the time it takes the system to agree on Bitcoin-NG's microblocks but does not improve commit latency [19], while OHIE's average commit latency is around 10 minutes [71]. One exception is Prism [6], which supports low-latency and high-throughput honest transactions by resorting to parallel voting chains, which determine the total order of blocks in the main chain. Still, their simulation-based evaluation results are around 2× higher than the causal commit latency of NimbleChain's most robust configuration (40 to 58 s with $\beta = 0.3$ [6]). Since all the above proposals are optimizations over BBP's foundations, NimbleChain can supplement any of them with the low-latency of our promise fast path.

A notable alternative is to organize transactions as a DAG, which has been proposed in IOTA's Tangle system [62] to achieve improved performance, including lower latency. However, the safety of this solution depends on an honest central point of control, which is at odds with the decentralized and permissionless nature of blockchains.

Sharding. Systems such as Elastico [46], OmniLedger [37], Rapid-Chain [72], Monoxide [68], Ethereum 2.0 [17] or Tao et al.'s proposal [67] rely on multiple parallel blockchains cooperating via sharding, where a small committee maintains each shard. This approach achieves substantial throughput gains (up to thousands of transactions per second), but at the cost of security, since the smaller shards are vulnerable to powerful attackers. Since sharded proposals typically assume (multiple) BBP-based blockchain instances, NimbleChain can be generalized to supplement sharded proposals with a low-latency promise fast path. Some sharded proposals have also been shown to achieve comparable commit latency savings as NimbleChain (e.g., References [37, 72]).

Still, such results are possible in networks with much lower RTT than the one considered in our article and only in specific best-case workloads.

Proof-of-X alternatives. Another research avenue has proposed permissionless consensus algorithms that replace PoW with energy-efficient alternatives, such as Proof-of-Stake [5, 23, 32, 73], Proof-of-Space [2, 15], or Proof-of-Elapsed-Time [10]. Among such proposals, some are still based on a variant of BBP, despite replacing the PoW leader election component by a PoX alternative (e.g., References [2, 15, 32, 73]). Therefore, NimbleChain's fast path can be integrated onto such proposals.

Blockchains based on BFT. A relevant body of work leverages BFT protocols, executed among small committees of processes, to improve the performance of permissionless blockchains. Proposals such as ByzCoin [36], Thunderella [57], and Solida [1] combine BBP with BFT protocols. Other proposals such as Algorand [23], HoneyBadger [51], and Stellar [45] are more disruptive. These approaches can achieve comparable commit latencies as NimbleChain's most robust configuration (e.g., in Algorand [23]'s best-case setting, 22 seconds). However, they significantly change the trust assumptions of permissionless blockchains, as BFT consensus requires that 2/3 of the validators must be trusted. Further, most of these proposals are highly disruptive and, as such, cannot incrementally extend existing mainstream permissionless blockchains.

Layer-2 proposals. Layer-2 proposals employ an additional protocol layer that handles (and commits) transactions and use the permissionless blockchain as a backend anchor to ensure consistency in the presence of malicious behaviour. In that sense, NimbleChain fits into this broad category. Among the most relevant proposals, so-called off-chain solutions such as the Lightning Network [59] and FastPay [27] rely on a separate network of payment channels and allow two or more parties to exchange currency without committing in the blockchain. However, these proposals have important shortcomings. They work at the expense of temporarily locking payment guarantees (often called *collaterals*) in the blockchain if a party misbehaves. While proposals based on payment networks are not tailored to unidirectional payment flows (as typical in retail payments from customers to merchants [49]), alternatives based on payment hubs [14, 28] either impose trusted entities or increased locked funds.

More recently, Snappy [49] proposes a novel on-chain smart-contract-based alternative that mitigates the above-mentioned shortcomings and achieves payment commit latency in the order of a few seconds. Still, Snappy has important scalability limitations in the number of payment recipient processes (up to 200 statekeeping merchants [49]). Further, since Snappy payments require smart contract invocations, they cost 8× more than simple transactions (in Snappy's Ethereum-based implementation [49]). In contrast, NimbleChain neither requires collaterals, nor restricts scalability, nor increases transaction cost.

Weakly consistent blockchains. Like NimbleChain, some proposals attempt to obtain partial orders instead of total orders for cryptocurrency transactions. In the permissionless world, notable proposals include SPECTRE [64], TrustChain [54], ABC [63], Avalanche [60], and Pastro [38]. In the context of permissioned blockchains, Astro [11] exploits Byzantine reliable broadcast [47] to build a payment system. All these proposals exploit the fact that cryptocurrecy transfer transactions do not need to be totally ordered, hence can be managed by weaker primitives than consensus. Like NimbleChain's promise fast path, the above proposals can serve the weak consistency needs of some applications such as cryptocurrencies. Still, these proposals cannot directly support general-purpose blockchains, whose application ecosystem comprises applications with weaker consistency demands (such as cryptocurrencies) and strong sequential consistency (such as most smart contracts). In contrast, NimbleChain's hybrid consistency model is tailored to such mixed ecosystems.

Permissioned blockchains. Permissioned blockchains (e.g., References [4, 12, 53]) have emerged as a considerably more efficient alternative to the permissionless counterpart. In contrast to the latter, permissioned protocols do not support public systems in which anyone can participate without a specific identify. Hence, permissioned blockchain protocols target a different trust model and thus are out of the scope of our article.

Hybrid-consistency replication. The dichotomy between weak and strong consistency is well studied in the context of traditional geo-replicated systems [9]. It is well established that one needs to forfeit strong consistency

to obtain the high availability, low latency, partition tolerance and high scalability that geo-replicated systems demand [9]. It is also known that many geo-distributed applications do not require strong consistency for *every* operation [44] and that many such applications are dominated by operations that are correct even if executed over a weakly consistent view. This observation has motivated the advent of geo-replicated systems supporting hybrid (or mixed) consistency models [43, 50]. To the best of our knowledge, NimbleChain is the first to introduce hybrid consistency models to permissionless environments.

9 CONCLUSIONS AND FUTURE WORK

This article proposes NimbleChain, which extends standard permissionless blockchains with a fast path that delivers *partially ordered promises of commitment*. This fast path supports cryptocurrency transactions and only takes a small fraction of the original commit latency, while providing consistency guarantees that are *strong enough* to ensure correct cryptocurrencies. Since today's general-purpose blockchains also support smart contract transactions, which typically have (strong) sequential consistency needs, NimbleChain implements a *hybrid* consistency model that also supports strongly consistent applications. To the best of our knowledge, NimbleChain is the first system to bring together fast partially ordered transactions with totally ordered, consensus-based transactions in a permissionless setting.

Our evaluation conducted in a realistic geo-distributed environment with 500 processes shows that the average latency to promise a transaction is an order of magnitude faster than consensus-based commit. Furthermore, our empirical evaluation of fragmentation attacks show that, even considering very favourable conditions for the attacker, an attacker cannot achieve any tangible benefits in exploring this kind of attack. A formal analysis of fragmentation attacks is out of the scope of this article, and left for future work.

Overall, we believe that our approach of bringing fast transactions to permissionless blockchains as an extension to existing blockhains, rather than proposing a new system from scratch, is a step in the direction of bringing these results closer to adoption by *de facto* blockchain systems such as Ethereum or Bitcoin, thus benefiting both the academic and industry communities.

Our work unveils new research avenues. Although this article focuses on cryptocurrencies as the obvious application to benefit from the promise fast path of NimbleChain, our proposal can also provide important benefits to smart contracts that have (a subset of) transactions with weaker consistency needs. As an example, smart contracts that employ the ERC20 Token Standard [69] to transfer some asset may have weaker consistency needs. However, providing a hybrid consistency model to smart contract programs requires a careful integration of this model into smart contract execution runtimes, as well as providing programmers with the adequate abstractions to help them build smart contract methods that can safely run with weaker guarantees.

ACKNOWLEDGEMENTS

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. 2018. Solida: A blockchain protocol based on reconfigurable byzantine consensus. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS'17), LIPIcs,*Vol. 95, James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 25:1–25:19. https://doi.org/10.4230/LIPIcs.OPODIS.2017.25

[2] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. 2017. Beyond Hellman's time-memory trade-offs with applications to proofs of pace, 357–379. https://doi.org/10.1007/978-3-319-70697-9_13

[3] Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. 1995. Causal memory: Definitions, implementation, and programming. *Distrib. Comput.* 9, 1 (March 1995), 37–49. https://doi.org/10.1007/BF01784241

[4] 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 Vukolic, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the 13th European Conference on Computer Systems (EuroSys'18)*, Rui Oliveira, Pascal Felber, and Y. Charlie Hu (Eds.). ACM, 30:1–30:15. https://doi.org/10.1145/3190508.3190538

[5] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. 2018. *Helix: A Scalable and Fair Consensus Algorithm Resistant to Ordering Manipulation*. Cryptology ePrint Archive, Paper 2018/863.

[6] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*. Association for Computing Machinery, New York, NY, 585–602. https://doi.org/10.1145/3319535.3363213

[7] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. 2013. Have a snack, pay with Bitcoins. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P'13)*. 1–5. https://doi.org/10.1109/P2P.2013.6688717

[8] Stefano Bistarelli, Ivan Mercanti, and Franceso Santini. 2018. An analysis of non-standard bitcoin transactions. *Crypto Valley Conference on Blockchain Technology (CVCBT'18)*, 93–96. DOI:10.1109/CVCBT.2018.00016

[9] Eric A. Brewer. 2000. Towards robust distributed systems (abstract). In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*. Association for Computing Machinery, New York, NY, 7. https://doi.org/10.1145/343477.343502

[10] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. 2017. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 282–297.

[11] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. 2020. Online payments by merely broadcasting messages. In *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'20)*. IEEE, 26–38.

[12] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards scaling blockchain systems via sharding. In *Proceedings of the International Conference on Management of Data (SIGMOD'19)*. Association for Computing Machinery, New York, NY, 123–140. https://doi.org/10.1145/3299869.3319889

[13] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. 2020. SOK: Cryptocurrency networking context, state-of-the-art, challenges. In *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES'20)*. Association for Computing Machinery, New York, NY, Article 5, 13 pages. https://doi.org/10.1145/3407023.3407043

[14] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual payment hubs over cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'19)*. IEEE, 106–123. https://doi.org/10.1109/SP.2019.00020

[15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of space. In *Annual Cryptology Conference*. Springer, 585–605.

[16] ETHDEV and the Ethereum community. 2017. Geth. Retrieved from https://github.com/ethereum/go-ethereum/wiki/geth.

[17] ETHDEV and the Ethereum community. 2019. Ethereum 2.0: Sharding Roadmap. Retrieved from https://github.com/ethereum/wiki/wiki/Sharding-roadmap.

[18] Etherscan. 2021. Etherscan–Ethereum Blockchain Explorer. Retrieved from https://etherscan.io/.

[19] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. 2016. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. 45–59.

[20] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 281–310.

[21] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology (CRYPTO'17)*, Jonathan Katz and Hovav Shacham (Eds.). Springer International Publishing, Cham, 291–323.

[22] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. 2018. Decentralization in bitcoin and ethereum networks. Financial Cryptography and Data Security, Lecture Notes in Computer Science, Vol. 10957.

[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.

[24] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. 2020. Kollaps: Decentralized and dynamic topology emulation. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys'20)*. Association for Computing Machinery, New York, NY, Article 23, 16 pages. https://doi.org/10.1145/3342195.3387540

[25] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The consensus number of a cryptocurrency. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 307–316.

[26] Hanyue Guo, Jiting Zhou, Jiaqi Wang, and Xiaodong Wang. 2018. A bike sharing system based on Blockchain platform. In *MATEC Web of Conferences*, Vol. 232. EDP Sciences, 01027.

[27] Zijiang Hao, Raymond Ji, and Qun Li. 2018. FastPay: A secure fast payment method for edge-IoT platforms using blockchain. In *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC'18)*. IEEE, 410–415.

[28] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS'17)*. The Internet Society.

[29] Yiwen Hu, Sihan Wang, Guan-Hua Tu, Li Xiao, Tian Xie, Xinyu Lei, and Chi-Yu Li. 2021. Security threats from bitcoin wallet smartphone applications: Vulnerabilities, attacks, and countermeasures. In *Proceedings of the 11th ACM Conference on Data and Application Security and Privacy (CODASPY'21)*. Association for Computing Machinery, New York, NY, 89–100. https://doi.org/10.1145/3422337.3447832

[30] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in Bitcoin. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'12)*. Association for Computing Machinery, New York, NY, 906–917. https://doi.org/10.1145/2382196.2382292

[31] Aggelos Kiayias and Giorgos Panagiotakos. 2015. *Speed-Security Tradeoffs in Blockchain Protocols*. Cryptology ePrint Archive, Paper 2015/1019.

[32] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*. Springer, 357–388.

[33] Lucianna Kiffer, Rajmohan Rajaraman, and abhi shelat. 2018. A better method to analyze blockchain consistency. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. Association for Computing Machinery, New York, NY, 729–744. https://doi.org/10.1145/3243734.3243814

[34] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security'16)*. 279–296.

[35] Eleftherios Kokoris-Kogias. 2019. *Robust and Scalable Consensus for Sharded Distributed Ledgers*. Cryptology ePrint Archive, Report 2019/676.

[36] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16)*. USENIX Association, 279–296.

[37] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*. IEEE, 583–598.

[38] Petr Kuznetsov, Yvonne-Anne Pignolet, Pavel Ponomarev, and Andrei Tonkikh. 2021. Permissionless and asynchronous asset transfer. In *Proceedings of the 35th International Symposium on Distributed Computing (DISC'21), LIPIcs*,Vol. 209, Seth Gilbert (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:19. https://doi.org/10.4230/LIPIcs.DISC.2021.28

[39] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (July 1978), 558–565. https://doi.org/10.1145/359545.359563

[40] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive block chain protocols. In *Proceedings of the 19th International Conference on Financial Cryptography and Data Security (FC'15), Lecture Notes in Computer Science*, Vol. 8975, Rainer Böhme and Tatsuaki Okamoto (Eds.). Springer, 528–547. https://doi.org/10.1007/978-3-662-47854-7_33

[41] Chenxing Li, Peilun Li, Dong Zhou, Wei Xu, Fan Long, and Andrew Yao. 2018. Scaling Nakamoto consensus to thousands of transactions per second. arXiv 2018. arXiv:1805.03870 [cs.DC]. Retrieved from https://arxiv.org/abs/1805.03870.

[42] Chenxing 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 *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'20)*. USENIX Association, 515–528.

[43] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. 2012. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. USENIX Association, Berkeley, CA, 265–278. http://dl.acm.org/citation.cfm?id=2387880.2387906.

[44] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2011. Don'T settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. ACM, New York, NY, 401–416. https://doi.org/10.1145/2043556.2043593

[45] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. 2019. Fast and secure global payments with stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP'19)*. Association for Computing Machinery, New York, NY, USA, 80–96. https://doi.org/10.1145/3341301.3359636

[46] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. Association for Computing Machinery, New York, NY, 17–30. https://doi.org/10.1145/2976749.2978389

[47] Dahlia Malkhi and Michael Reiter. 1997. A high-throughput secure reliable multicast protocol. *J. Comput. Secur.* 5, 2 (1997), 113–127.

[48] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. 2018. *Low-resource Eclipse Attacks on Ethereum's Peer-to-peer Network*. IACR Cryptol. ePrint Arch., 236.

[49] Vasilios Mavroudis, Karl Wüst, Aritra Dhar, Kari Kostiainen, and Srdjan Capkun. 2020. Snappy: Fast on-chain payments with practical collaterals. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society.

[50] Matthew Milano and Andrew C. Myers. 2018. MixT: A language for mixing consistency in geodistributed transactions. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 226–241.

[51] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42.

[52] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

[53] 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 (SOSP'21)*. Association for Computing Machinery, New York, NY, 35–48. https://doi.org/10.1145/3477132.3483584

[54] Pim Otte, Martijn de Vos, and Johan Pouwelse. 2020. TrustChain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*. 107 (2020), 770–780. https://doi.org/10.1016/j.future.2017.08.048

[55] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 643–673.

[56] Rafael Pass and Elaine Shi. 2017. FruitChains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'17)*. Association for Computing Machinery, New York, NY, 315–324. https://doi.org/10.1145/3087801.3087809

[57] Rafael Pass and Elaine Shi. 2017. Hybrid consensus: Efficient consensus in the permissionless model. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC'17), LIPIcs*, Vol. 91, Andréa W. Richa (Ed.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 39:1–39:16. https://doi.org/10.4230/LIPIcs.DISC.2017.39

[58] Rafael Pass and Elaine Shi. 2017. Rethinking large-scale consensus. In *Proceedings of the IEEE 30th Computer Security Foundations Symposium (CSF'17)*. IEEE, 115–129.

[59] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Retrieved from https://lightning.network/lightning-network-paper.pdf.

[60] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. 2020. Scalable and probabilistic leaderless BFT consensus through metastability. arXiv:1906.08936 [cs.DC]. Retrieved from https://arxiv.org/abs/1906.08936.

[61] Paulo Silva, David Vavřička, João Barreto, and Miguel Matos. 2020. Impact of geo-distribution and mining pools on blockchains: A study of ethereum. In *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'20)*. IEEE Computer Society, 245–252. https://doi.org/10.1109/DSN48063.2020.00041

[62] Wellington Fernandes Silvano and Roderval Marcelino. 2020. Iota tangle: A cryptocurrency to communicate Internet-of-Things data. *Fut. Gener. Comput. Syst.* 112 (2020), 307–319. https://doi.org/10.1016/j.future.2020.05.047

[63] Jakub Sliwinski and Roger Wattenhofer. 2019. ABC: Asynchronous blockchain without consensus. arXiv:1909.10926. Retrieved from https://arxiv.org/abs/1909.10926.

[64] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. *SPECTRE: A Fast and Scalable Cryptocurrency Protocol*. Cryptology ePrint Archive, Paper 2016/1159.

[65] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. 2018. *PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus*. Cryptology ePrint Archive, Report 2018/104.

[66] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in Bitcoin. In *Proceedings of the 19th International Conference on Financial Cryptography and Data Security(FC'15), Lecture Notes in Computer Science*, Vol. 8975, Rainer Böhme and Tatsuaki Okamoto (Eds.). Springer, 507–527. https://doi.org/10.1007/978-3-662-47854-7_32

[67] Yuechen Tao, Bo Li, Jingjie Jiang, Hok Chu Ng, Cong Wang, and Baochun Li. 2020. On sharding open blockchains with smart contracts. In *Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE'20)*. 1357–1368. https://doi.org/10.1109/ICDE48307.2020.00121

[68] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out blockchains with asynchronous consensus zones. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*. 95–112.

[69] Bitcoin Wiki. 2018. The ERC20 Token Standard. Retrieved from https://en.bitcoinwiki.org/wiki/ERC20.

[70] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.

[71] H. Yu, I. Nikolic, R. Hou, and P. Saxena. 2020. OHIE: Blockchain scaling made simple. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'20)*. IEEE Computer Society, Los Alamitos, CA, 90–105. https://doi.org/10.1109/SP40000.2020.00008

[72] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling blockchain via full sharding. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. Association for Computing Machinery, New York, NY, 931–948. https://doi.org/10.1145/3243734.3243853

[73] Vlad Zamfir. 2017. *Casper the Friendly Ghost: A Correct by Construction Blockchain Consensus Protocol*. White Paper. Retrieved from https://github.com/ethereum/research/blob/master/papers/CasperTFG/CasperTFG.pdf. (2017).