We will DAG you

Ignacio Amores-Sesar
University of Bern
ignacio.amores@unibe.ch

Christian Cachin
University of Bern
christian.cachin@unibe.ch

7 November 2023

Abstract

DAG-based protocols have been proposed as potential solutions to the latency and throughput limitations of traditional permissionless consensus protocols. However, their adoption has been hindered by security concerns and a lack of a solid foundation to guarantee improvements in both throughput and latency. In this paper, we present a construction that rigorously demonstrates how DAG-based protocols can achieve superior throughput and latency compared to chain-based consensus protocols, all while maintaining the same level of security guarantees.

1 Introduction

In the ever-evolving landscape of distributed systems, achieving consensus among a set of processes has become a fundamental challenge that has garnered significant attention in recent years. Consensus protocols are a universal primitive in distributed computing, ensuring that a network of interconnected processes can collectively agree on a shared state despite potential failures or malicious actors. However, as the demands on distributed systems continue to grow, the need for consensus protocols that can deliver both higher throughput and lower latency has become increasingly pressing. This need is particularly relevant in permissionless consensus protocols as used by cryptocurrencies and blockchain protocols, which face stringent demands on their throughput and latency.

Traditional consensus protocols have exhibited considerable advancements in both throughput and latency since the first practical consensus protocols [12, 7]. One of the most promising lines of work are DAG consensus protocols as introduced by the "All you need is DAG" paper [10] and subsequently extended by Narwhal and Tusk [8], Bullshark [22], and Cordial Miners [11]. A common characteristic of these protocols is their capacity to enable every participant to generate blocks that reference previous blocks, forming a *directed acyclic graph* (*DAG*). In permissionless protocols like Bitcoin [14], every process (miner) can create a block upon successfully solving the cryptographic puzzle. Therefore, the concept of constructing a DAG that is later ordered, as proposed by Keidar et al. [10], holds the potential to enhance the throughput and latency of permissionless consensus protocols. In essence, DAG protocols may surpass traditional permissionless consensus protocols, which form a chain.

The evident approach to improving the throughput of chain protocols is to increase the block ratio, i.e., the number of block produced per unit of time, effectively accelerating the execution of the protocol as there is less time between created blocks. This goal can be pursued by lowering the difficulty in *Proof-of-Work* (*PoW*) protocols. However, increasing the block ratio may harm the protocol since it elevates the likelihood of forks—situations where two different processes create blocks extending the chain. An abandoned block is one that is never output by the protocol, whenever a chain protocol forks, an abandoned block is produced. Therefore, despite the increased number of generated blocks, the number of abandoned blocks concurrently rises, adversely affecting the protocol's throughput. Moreover, it is imperative to recognize that the block ratio cannot be augmented arbitrarily without compromising the protocol's security.

In this paper, we introduce a construction that takes as input a DAG-based protocol or a chain protocol Π, which may produce abandoned blocks, and produces a new DAG protocol Π' with the property that

1

every created block is eventually output. Specifically, Π' creates the same number of blocks as the base protocol Π and outputs *every* created block of Π. We show that the safety and liveness of Π' reduces to the safety and liveness of Π. In simpler terms, Π' is as safe and live as Π. Furthermore, we establish that Π' has lower or equal *latency* as Π, while achieving strictly higher *throughput*. Our main contribution lies in a formal proof that chain protocols cannot achieve optimal throughput, i.e., for any chain protocol Π, there is a DAG protocol Π' that is safe and life under the same assumptions as Π, with the same or better latency and better throughput.

2 Related work

DAG protocols represent a recent breakthrough within the domain of permissioned consensus protocols [10, 8, 22, 11]. While DAG protocols have been previously introduced in the permissionless context, their adoption and success have been somewhat restrained due to their inherent complexity when compared to traditional chain protocols. Several well-known DAG protocols have exhibited vulnerabilities, highlighting challenges in their success. For instance, IOTA [18], one of the pioneering DAG protocols, has been susceptible to vulnerabilities such as Parasite-chain attacks [18, 17]. Another promising protocol, GhostDAG [20], has also revealed vulnerabilities in its design [13]. Even Avalanche [19], the most successful DAG protocol in terms of market capitalization, originally had vulnerabilities in its design [3].

An intriguing DAG protocol to note is Conflux[13], which leverages the GHOST consensus rule [21] and augments blocks with additional references to transform a chain protocol into a DAG. Li et al. [13] have demonstrated that Conflux's security is directly inherited from the security of GHOST. However, it is worth mentioning that the GHOST protocol has exhibited lower resilience than other consensus protocols in the presence of network malfunctions [15, 4].

Our contribution to this landscape is a formal proof of the superior performance of DAG protocols, facilitated by a construction that can be conceptualized as an extension of the Conflux construction [13]. Specifically, when we instantiate the throughput closure using GHOST [21], we arrive at Conflux [13].

3 Abstractions

We consider a set of n *processes* $\mathcal{P} = \{P_1, P_2, \ldots\}$ that interact with each other by exchanging messages through the network. A protocol Π for $\mathcal{P}$ consists of a collection of programs with instructions for all processes. In particular we are interested in the study of *chain protocol* and *DAG protocol* protocols, i.e., protocol that rely on a chain or a DAG to deliver blocks. These two concepts are formally defined below.

Chain and DAG protocols are pivotal tools employed to establish robust and secure ledgers, and as such, they must adhere to specific fundamental requirements.

Traditionally, the gold standard concept is *atomic broadcast* [6], which ensures that all processes deliver the same set of transactions in the same order. In this paper, we consider a variant of this abstraction that includes the concept of a *block* in the interface and properties [2]. Processes broadcast transactions and deliver blocks using the events *bab-broadcast(tx)* and *bab-deliver(b)*, respectively, where block b contains a sequence of transactions $[tx_1, \ldots, tx_m]$. The protocol outputs an additional event *bab-mined(b, P)*, which signals that block b has been *mined* by process P, where P is defined as the *miner* of b. The event *bab-mined(b, P)* can be understood as the creation of block b by party P. Notice that *bab-mined(b, P)* signals only the creation of a block and not its delivery. In addition to predicate *VT()* that determines the validity of a transaction, we also equip our protocol with a validity predicate *VB()* to be applied to blocks. These predicates and function are determined by the higher-level application or protocol.

Definition 1. A protocol implements *block-based atomic broadcast* with validity predicates *VT()* and *VB()* if it satisfies the following properties, except with negligible probability:

Validity: If a correct process invokes a *bab-broadcast(tx)*, then every correct process eventually outputs *bab-deliver(b)*, for some block b that contains *tx*.
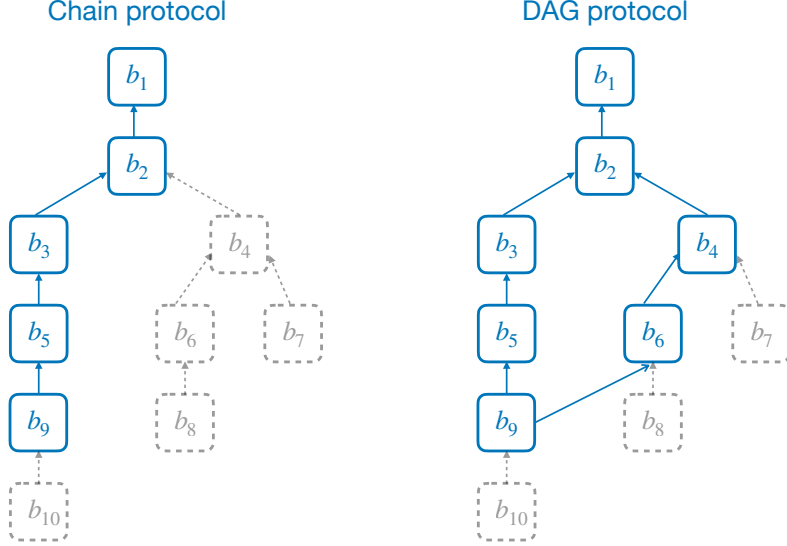
Figure 1. Comparison between a chain protocol and a DAG protocol. Blocks in blue (continuous lines) are the *bab-delivered* blocks, whereas grey (dashed) blocks are *bab-mined* but not *bab-delivered*. The protocol on the left is a chain protocol, each block refers to exactly one block and there is a block (b_9) such that every currently *bab-delivered* block is b_9 or an ancestor of it. The protocol on the right is a DAG protocol, block b_9 references multiple blocks.

No duplication: No correct process outputs *bab-deliver(b)* more than once.

Integrity: If a correct process outputs *bab-deliver(b)*, then it has previously output *bab-mined(b, ·)* exactly once.

Agreement: If some correct process outputs *bab-deliver(b)*, then eventually every correct process outputs *bab-deliver(b)*.

Total order: Let b and b' be blocks, and P_i and P_j correct processes that both output *bab-deliver(b)* and *bab-deliver(b')*. P_i delivers b before b' if and only if P_j delivers b before b'.

External validity: If a correct process outputs *bab-deliver(b)*, then $VB(b) = \text{TRUE}$.

The block-based atomic broadcast abstraction can be implemented by protocols based on different approaches. These difference are not captured in Definition 1, but can relevant for the performance of the protocol. The two families of protocols of interest for this paper are *chain protocol* and *DAG protocol* protocols. The distinguishing factor between them lies in the set of references to previously mined blocks. Specifically, for a given block b, we denote the set of *bab-mined* blocks referenced by b as *parents(b)*, commonly known as the *parents* of b. Furthermore, the set of *bab-mined* blocks reachable through references from b is represented as *ancestors(b)* and is often referred to as the *ancestors* of b. A block b is a *descendant* of its ancestors. A block with no descendants is also called *leaf*.

Definition 2 (Chain protocol, DAG protocol). A block-based atomic broadcast protocol Π is a *DAG protocol* protocol if Π-*mined* blocks contain references to other Π-*mined* blocks, meaning that the set of references is not empty. Π is a *chain protocol* protocol if every Π-*mined* block refers to exactly one Π-*mined* block and for every honest process P_i there is a Π-*delivered* block b such that every Π-*delivered* by P_i is b or in *ancestors(b)*. In essence, Π-*delivered* blocks form a chain.

Figure 1 illustrates an example of both chain and DAG protocols.

To set the stage, we make the assumption that both chain and DAG protocols begin with an initial, hard-coded block referred to as the *genesis* block. This genesis block is special in that it possesses an empty set of references. It is important to note that, according to Definition 2, chain protocols inherently

are DAG protocols. The blocks mined in chain protocols produce a *tree*, a particular kind of DAG. Therefore, for the remainder of this paper, we will use the term "DAG protocol" to encompass both DAG protocol and chain protocols, acknowledging this inclusion.

One significant implication of abstracting DAG protocols as block-based atomic broadcast (Definition 1) is that the protocol must define a function that operates on the directed acyclic graph (*DAG*) that produces a list of delivered blocks. It is worth mentioning that certain DAG protocols, such as the original Avalanche protocol [19, 3], do not output an ordered list of transactions but the list output by different processes may differ up to permutation. While DAG protocols can also be modeled as generic broadcast [16], situations arise where complete transaction ordering, as seen in calls to smart contracts, becomes necessary. For the purposes of this paper, we focus on protocols that can be effectively modeled as block-based atomic broadcast. The results we derive in this context generalize straightforwardly to protocols modelled as generic broadcast.

4 Model

DAG protocols base their security on different techniques such as *proof of work* (*PoW*), *proof of stake* (*PoS*) [9], *proof of space-time* (*PoST*) [1], or *proof of elapsed time* (*PoET*) [5]. For the sake of simplicity, we restrict our model to PoW. Nevertheless, our model can readily be extended to incorporate other techniques.

Processes. Consistent with prior research, our protocol operates without explicit knowledge of the number or identities of the processes. The processes themselves remain unaware of these details as well. We assume a static network consisting of n processes, where up to f processes to be corrupted by the adversary, thereby exhibiting arbitrary behavior.

Blocks. A transaction *tx*, comprises a set of *inputs*, a set of *outputs*, and a collection of digital signatures, as in Bitcoin [14]. Transactions have size $|tx|$, and they are grouped into blocks, as introduced in Definition 1. Each block encompasses a specific number of transactions, denoted as m, a number of references to previously *bab-mined* blocks, quantified as n_{refs}, and further parameters essential for the proper execution of protocol Π. It is noteworthy that the size of a reference, represented as $|ref|$, is significantly smaller than that of a transaction, for simplicity, we consider it to be negligible. We reiterate that protocol Π defines external validity predicates, *VT*() and *VB*(), responsible for determining the *validity* of a transaction or block.

Network. A *diffusion functionality* implements communication among the processes, which is structured into *synchronous* rounds. The functionality keeps a distinct $RECEIVE_i$ string for each process P_i and makes it available to P_i at the start of every round. The purpose of the string $RECEIVE_i$ is to serve as a repository for all the messages received by P_i.

When a process, say P_i, instructs the diffusion functionality to *broadcast* a set of message, it signifies that P_i has "completed its round". In response, the functionality marks P_i as having completed its operations for that specific round. The adversary, whose actions are described in detail below, possesses the ability to access the string of any process at any point during the execution. Additionally, the adversary can observe every message broadcast by any process instantaneously. Furthermore, the adversary has the capability to insert messages directly and selectively into $RECEIVE_i$ for any process P_i, ensuring that only P_i receives the message at the outset of the following round. This behavior models what is often termed a *rushing* adversary.

Once all non-corrupted processes have concluded their respective rounds, the diffusion functionality aggregates all messages that were broadcast by non-corrupted processes during that round. These aggregated messages are then appended to the $RECEIVE_i$ strings for all processes, this is the reason of the name *synchronous* rounds. Subsequently, each non-corrupted process updates its local view at the

conclusion of every round. If a non-corrupted process Π-*mines* a block in round r, all processes receive the Π-*mined* block by the subsequent round $r + 1$.

Furthermore, even if the adversary causes a block to be received selectively by only some non-corrupted processes in round r, the block is received by all non-corrupted processes by round $r + 2$. The update of the local view also encompasses the Π-*delivery* of blocks that meet a given criteria define by protocol Π.

Adversary. The adversary can corrupt up to f processes at the beginning of the execution. These corrupted processes may deviate arbitrarily from the protocol, adhering to the instructions from the adversary. Additionally, the adversary wields control over the *diffusion functionality*. The adversary can schedule the delivery of messages, read the contents of the $RECEIVE_i$ string for every process at any point during the execution, and directly write messages into the $RECEIVE_i$ of any process. The adversary signals the conclusion of her round by transmitting a specially designated message.

Round structure. At the beginning of the round, process P_i reads the messages in its input string $RECEIVE_i$. Then, P_i proceeds to update its internal state in accordance with the received messages and performs a set of actions defined by protocol Π. Such actions include the Π-*delivery* of blocks. P_i concludes the round by broadcasting a set of messages to the other processes.

4.1 Abandoned blocks

Definition 3. An *execution* is a history with an entry for each round containing the actions, a list of received messages, and a list of sent messages by each process in that round.

While an event may be theoretically possible within an execution, its occurrence might have a probability of zero. For instance, consider an algorithm that continuously flips an unbiased coin indefinitely. There could be an execution where all outcomes are heads, but the probability of this specific sequence of events happening is zero, as it is the limit of an infinite execution.

To circumvent these issues, we introduce the concept of a *partial execution*.

Definition 4. Given a protocol Π, the set of λ-*partial executions* Φ_λ is defined to be the set of λ-prefixes of all executions of protocol Π. A *partial execution* is an execution that belongs to Φ_λ for some $\lambda \in \mathbb{N}$.

Definition 5. Given an execution $\mathcal{E}$ of a block-based atomic broadcast protocol Π, an *abandoned* block in $\mathcal{E}$ is is an honestly *bab-mined* block b such that b is not *bab-delivered* in $\mathcal{E}$.

It is important to note that the validity property defined in block-based atomic broadcast (Definition 1) does not guarantee that every *bab-mined* block will eventually be *bab-delivered*. Instead, this property ensures that for each *bab-broadcast* transaction, there exists at least one *bab-delivered* block that contains it. The concept of abandoned blocks is a significant concern in the context of such protocols. Abandoned blocks have been honestly *bab-mined* but are never *bab-delivered*. The existence of abandoned blocks can severely impact the performance of a chain protocol or DAG protocol.

Definition 6. A protocol Π *permits abandoned blocks* if there exist a block b and a partial execution $\mathcal{E}$ such that: b is abandoned in any extension of $\mathcal{E}$.

Remark 1. Note that given a protocol that permits abandoned blocks, the probability, taken over the randomness of the protocol, of having at least one abandoned block in an execution is greater than zero, since partial executions happen with non-zero probability.

Determining whether a given protocol Π permits abandoned blocks or not can be a challenging task and, in some cases, may not be computable due to the need to simulate potentially infinitely long executions. However, for certain protocols like Bitcoin [14], the existence of abandoned blocks is a direct consequence forks occurring among honest miners. This phenomenon is formalized in the following definition.

Definition 7. Given an execution $\mathcal{E}$ of a given protocol Π, a round r *forked* if protocol Π outputs two events *bab-mined*(b, P_i) and *bab-mined*(b', P_j) in round r at two distinct honest processes P_i and P_j. A protocol with a forked round in at least one partial execution is a *forkable protocol*.

Lemma 1. *A forkable chain protocol Π permits abandoned blocks.*

Proof. Given a forkable protocol Π, there exist a round r in which two different honest processes output events *bab-mined*(b, P_i) and *bab-mined*(b', P_j). In particular $b \neq b'$ because their miners are different. Π is also a chain protocol. thus both b and b' have a unique reference to previously *bab-mined* blocks, so they cannot reference each other. Another implication of Π being a chain protocol is that at any point in the execution in the protocol there exists a *bab-mined* block b^* such that every *bab-delivered* is in *ancestors*(b^*). Since every block only contains a single reference and b and b' do not refer each other, we conclude that no honest processes can *bab-deliver* both b and b' simultaneously. $\square$

Transactions that were originally included in abandoned blocks must be re-included in subsequent blocks to maintain the validity property (Definition 1). This re-inclusion consumes space in new blocks and has implications for both latency and throughput, as we formalize below.

4.2 Throughput and latency

Definition 8. Given a block-based atomic broadcast protocol Π, an adversary $\mathcal{A}$, and an execution $\mathcal{E}$, we define the *throughput* of Π in the presence of $\mathcal{A}$ in execution $\mathcal{E}$ as the average number of *bab-delivered* blocks per round and we denote by *throughput*$(\Pi, \mathcal{A}, \mathcal{E})$.

Definition 9. Given a block-based atomic broadcast protocol Π, the *throughput of* Π is defined to be *throughput*$(\Pi) := \inf_{\mathcal{A}} \mathrm{E}[\textit{throughput}(\Pi, \mathcal{A}, \mathcal{E})]$, i.e., the infimum over all the possible adversaries $\mathcal{A}$ of the average over the randomness Π of *throughput*$(\Pi, \mathcal{A}, \mathcal{E})$ over all the possible executions.

Definition 10. The *goodput* of protocol Π is defined to be throughput of Π in the presence of an adversary that follows the instructions of the protocol.

Definition 11. Given a block-based atomic broadcast protocol Π, an adversary $\mathcal{A}$, an execution $\mathcal{E}$, and a transaction *tx*, we define *latency* of *tx* in the presence of adversary $\mathcal{A}$ in execution $\mathcal{E}$ as the number of rounds since *tx* is *bab-broadcast* until the first block containing *tx* is *bab-delivered*, and we denote it by *latency*$(\Pi, \mathcal{A}, \mathcal{E}, tx)$. We define the *latency of* Π to be the average number of rounds, over the transactions *tx* in execution $\mathcal{E}$, since *tx* is *bab-broadcast* until the first block containing *tx* is *bab-delivered* and denote it by *latency*$(\Pi, \mathcal{A}, \mathcal{E})$.

Definition 12. Given a block-based atomic broadcast protocol Π, The *latency* of protocol Π is defined as *latency*$(\Pi) = \sup_{\mathcal{A}} \mathrm{E}[\textit{latency}(\Pi, \mathcal{A}, \mathcal{E})]$, i.e., the supremum over all the possible adversaries $\mathcal{A}$ of the average over the randomness of the protocol of the *latency*$(\Pi, \mathcal{A}, \mathcal{E})$ over the possible executions $\mathcal{E}$.

5 The throughput closure

We introduce a novel construction designed to enhance a given DAG protocol Π. This construction results in a DAG protocol, which we call *the throughput closure of* Π and denote by Π'. Protocol Π' possesses the unique property of ensuring that every honestly *bab-mined* block is eventually *bab-delivered*. The mechanism by which protocol Π' accomplishes this feat involves the incorporation of additional references to blocks. For any given block b, protocol Π' defines the set *abandoned*(b) as the collection of valid blocks that will not be Π-*delivered* if b is to be Π-*delivered*. The block mining and delivery routines of the throughput closure Π' are built on top of their counterparts in Π.

Overview. As shown in Algorithm 1, when an honest process P_i Π-*mines* a block b, process P_i also Π'-*mines* the same block. However, in Π', the block b includes an additional set of references to the blocks in the set *abandoned*(b).

The modified delivery routine operates as follows: when a block b would be Π-*delivered*, all valid blocks in the set *abandoned*(b) are Π'-*delivered* in a fixed topological order immediately before b. This topological sort allows to order non Π-*delivered* blocks with respect to Π-*delivered* blocks deterministically according to the references included in the Π-*delivered* blocks. This is a crucial aspect as establishing a total order in a DAG can be generally challenging due to different processes having different partial views of the DAG. The topological sort τ ensure that all processes that have received block b agree on the same order. A canonical example for *topological sort* τ is to order the blocks in *abandoned*(b) according to their *depth* in the DAG, distance to genesis, breaking the ties according to the hash of the block. Note that if an adversary creates a block with low depth, it will be only Π-*delivered* when deeper block references it, thus the adversarial block is Π'-*delivered* concurrently with deeper blocks.

Constructing the set *abandoned*(b), even when it can be computed, may be challenging task, as we explained above. However, given a chain protocol Π the set *abandoned*(b) becomes trivial to compute as it is formed by every block that is not an ancestor of b. Furthermore, the set of references to *abandoned*(b) are the leaves of the DAG, with the exception of b. As an illustrative example, Figure 2 shows the application of this construction within the context of Bitcoin. If we consider Π to be GHOST protocol [21], we recreate the Conflux protocol [13]. Including references to the leaves in the DAG is the precise method for referring to the set *abandoned*(b) with a chain protocol Π. The same approach can be used with DAG protocols. This approach may be computationally cheaper than than computing the leaves in set *abandoned*(b), however, some blocks may be referenced when there is no need, adding redundancy of references. Further insights into this alternative approach are provided below.

Detailed description. We describe the execution of the protocol from the perspective of an honest process P_i. When honest process P_i Π'-*broadcasts* a transaction *tx*, it invokes Π-*broadcast*(*tx*) (L3–4). Notably, the broadcast of transactions occurs exactly as it does in protocol Π. When P_i triggers event Π-*mined*(b, P_i) (L5–11), it initially computes the set *abandoned*(b) locally. To Π'-*mine* a new block b', P_i augments b by adding extra references to the leaves of the set *abandoned*(b) (L7–9). Subsequently, P_i adds b' to the set of mined blocks $\mathcal{D}'$ (L10) and triggers the event Π'-*mined*(b', P_i) (L11).

When event Π'-*mined*(b', P_j) is triggered, P_i verifies the Π'-validity of b' and incorporates it into its local view (L12–14). So far, the execution of Π' closely parallels that of Π. However, the key distinction lies in the delivery of blocks (L15–19). When event Π-*deliver*(b) occurs, P_i searches for the block b' associated with b. P_i then assembles the set *ready*, which comprises the blocks to be Π'-*delivered* (L16). This set is computed as the set-difference between the ancestors of block b' and the ancestors of the last delivered block b'_l. P_i subsequently updates the last delivered block to be b' (L17). Finally, P_i applies a topological sorting algorithm τ to the set *ready* and Π'-*delivers* them accordingly (L18–19).

A block b' is deemed valid (L22–23) within protocol Π' if it satisfies two conditions: firstly, its associated block b must be Π-*valid*, and secondly, it must contain at least one Π'-valid transaction.

Algorithm 2 presents a greedy version of *abandoned*(b). In this approach, a process P_i adds references to b' for every block that is not already an ancestor of b within protocol Π.

The throughput closure mirrors protocol Π when the set *abandoned*(b) is empty for every block, indicating that the protocol does not permit the existence of abandoned blocks. However, if Π permits abandoned blocks, then there exists some executions of Π with a block b such that *abandoned*(b) $\neq \emptyset$, and the throughput closure diverges from the original protocol. The implementation of the throughput closure does entail an increase in local computation for processes. Specifically, processes need to scan the DAG and append a set of references to all leaves in *abandoned*(b) to the currently mined block b. The computational complexity of determining *abandoned*(b) can vary depending on the protocol, as discussed earlier. However, in the case of chain protocols, this set is relatively straightforward to compute. A process simply adds references to every leaf of a chain that has not been referenced by an ancestor.

Algorithm 1 Protocol Π' for process P_i.

    Implements: block-based atomic broadcast Π'
    Uses: block-based atomic broadcast Π
           topological sort τ

    State:
1:      $\mathcal{D}' \leftarrow \emptyset$
2:      $b'_\ell \leftarrow [\,]$

3: **upon event** Π'-*broadcast*(tx) **do**
4:      **invoke** Π-*broadcast*(tx)

5: **upon event** Π-*mined*(b, P_j) **do**
6:      **if** $P_i = P_j$ **then**
7:          $weak \leftarrow leaves(abandoned(b, \mathcal{D}'))$
8:          $b' \leftarrow b$
9:          $bl'.wrefs \leftarrow weak$
10:        $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{b'\}$
11:        **invoke** Π'-*mined*(b', P_i)

12: **upon event** Π'-*mined*(b', P_j) **do**
13:      **if** $VB'(b')$ **then**
14:          $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{b'\}$

15: **upon event** Π-*deliver*(b) **do**
16:      $ready \leftarrow ancestors'(b') \setminus ancestors'(b'_\ell)$
17:      $b'_\ell \leftarrow b'$
18:      **for** $b^* \in \tau(ready)$ **do**
19:          **invoke** Π'-*deliver*(b^*)

20: **function** *abandoned*($b, \mathcal{D}'$) **:**
21:      **return** $\{b' \in \mathcal{D}' : b' \notin ancestors'(b) \wedge incompatible(b, b')\}$

22: **function** $VB'(b')$ **:**
23:      **return** $VB(b) \wedge \exists\, tx \in b' : undelivered(tx)$

Algorithm 2 Greedy approach for process P_i.

24: **upon event** Π-*mined*(b, P_j) **do**   // Greedy approach
25:      **if** $P_i = P_j$ **then**
26:          $b' \leftarrow b$
27:          $weak \leftarrow leaves(\{b' \in \mathcal{D}' : b' \notin ancestors(b')\})$
28:          $bl'.refs \leftarrow bl'.refs || weak$
29:          $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{b'\}$
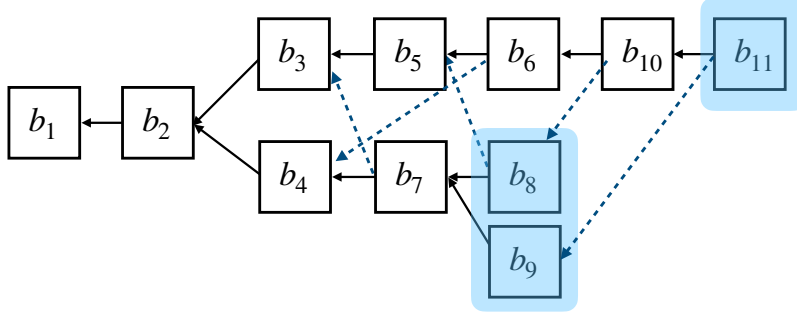30:   **invoke** Π'-*mined*(b', P_i)

Figure 2. An example of our construction applied to Nakamoto consensus. The full lines denote the references of the Nakamoto consensus and the blue dashed lines denote the extra references included by the throughput closure. According to Nakamoto consensus, the main chain is the chain $b_1 \cdots b_{11}$ and blocks $b_4, b_7, b_8,$ and b_9 are abandoned. Looking at b_{11}, the set *abandoned*(b_{11}) is formed by block b_9. Blocks $b_4, b_7,$ and b_8 are not part of the *abandoned*(b_{11}) because b_{10} already references them. When delivering b_{11}, block b_9 would be delivered between b_{10} and b_{11}.

6 Analysis

6.1 Security analysis

Theorem 2. *Given protocol DAG protocol Π implementing block-based atomic broadcast, its throughput closure Π' also implements block-based atomic broadcast.*

Proof. We demonstrate that the throughput closure Π' implements block-based atomic broadcast by leveraging the fact that Π does. Throughout this proof, we assume the perspective of an honest process P_i.

Validity: Assume that an honest process P_j Π'-*broadcasts* a given transaction *tx*. By construction, process P_j does so by invoking Π-*broadcast* transaction *tx* (L3–4). The validity property of protocol Π guarantees that process P_i eventually Π-*delivers* a block b containing transaction *tx*. Process P_i, by definition of the protocol, Π'-*delivers* the block b' consisting of block b with the addition of the extra set of references (L15–19). If every transaction contained in b' is invalid, the block is not Π'-*deliver*. In the case of block b', the validity check can only fail if transaction *tx* fails the validity predicate. Since *tx* is Π'-*broadcast*, the external validity predicate is satisfied unless some block containing *tx* has been Π'*delivered*.

We conclude that for any honestly Π'-*broadcast* transaction *tx*, P_i eventually Π'-*delivers* a block b' containing *tx*, thus validity property of protocol Π' is satisfied.

Integrity: Process P_i only Π'-*delivers* blocks that it Π'-*delivers* or ancestors of those contained in the set $\mathcal{D}$ (L15–19). A block b' enters the set $\mathcal{D}$ only after an invocation of Π-*mined*(b, P_j). We conclude that every Π'-*delivers* has previously been Π'-*mined*.

Agreement: Consider a block b' that is Π'-*delivered* by process P_i. We consider two different cases: when b whether b is Π-*delivered* or not. On the one hand, if b is Π-*delivered* by process P_i, every honest process eventually Π-*delivers* b, thus Π'-*delivers* b' as a consequence (L15–19). On the

other hand, if block bl' is Π'-*delivered* as a consequence of another block b^* is Π'-*delivered*. The same reasoning as above applies to block b^*, which implies the eventual Π'-*delivery* of block b'.

Total order: Consider two Π'*mined* blocks b'_1 and b'_2 and two honest processes P_i and P_j that $\Pi' -$ *deliver* both blocks. We distinguish four cases based on whether blocks b_1 and b_2 are Π-*delivered* or not.

Assume that both b_1 and b_2 are Π-*delivered*. Note that in the view of any honest process the order in which blocks b_1 and b_2 are Π-*delivered* is the same as blocks b'_1 and b'_2 are Π'-*delivered* (L15–19). Due to the total order property of protocol Π, process P_i Π-*delivers* block b_1 and b_2 in the same order as process P_j, thus both processes Π'-*deliver* blocks b_1 and b_2.

If either b'_1 or b'_2 are Π'-*delivered* as a consequence of another block b_3 being Π-*delivered*. Since the set of blocks that are Π'-*delivered* as consequence of block b'_3 are Π'-*delivered* immediately before b'_3, any block b' Π'-*delivered* before (after) b' is also Π'-*delivered* before (after) the set of blocks Π'-*delivered* as a consequence of b'. The same reasoning as above applies to this case. We conclude that P_i also Π'*delivers* both b'_1 or b'_2 in the same order as P_j.

The only case left is when both b'_1 and b'_2 are Π'-*delivered* as a consequence of two blocks b'_3 and b'_4 being Π'-*delivered*. If b'_3 and b'_4 are different the case is the same as before. If b'_3 and b'_4, both P_i and P_j use the topological order to determine in which order to Π'-*delivered*. Since the topological sorting is deterministic and depends only on block b'_3, both P_i and P_j Π'-*deliver* b'_1 and b'_2 in the same order.

External validity: The external validity property is imposed by lines L22–23.

$\square$

6.2 Throughput and latency

Theorem 2 states that the throughput closure Π' maintains the safety and liveness properties the original protocol Π. In this section, we delve into a comparative analysis of the performance aspects, through throughput and latency, between Π' and Π. It is important to note that both throughput and latency definitions take into account adversarial behavior, and the connection between the adversarial behavior of Π' and Π is discussed in the following remark.

Remark 2. Note that given an adversary $\mathcal{A}'$ for protocol Π', an adversary $\mathcal{A}$ for protocol Π can be constructed by merely removing the extra references from any block that $\mathcal{A}'$ Π'-*mines*. Additionally, given an adversary $\mathcal{A}$ for protocol Π, it can also be regarded as an adversary for protocol Π', as every action taken by $\mathcal{A}$ in protocol Π is allowed in protocol Π'.

Definition 13. Given an execution $\mathcal{E}'$ and an adversary $\mathcal{A}'$ for protocol Π', we define the equivalent execution of protocol Π as the execution $\mathcal{E}'$ without the extra references in each block and adversary $\mathcal{A}$ as discussed in Remark 2.

Lemma 3. *Given a DAG protocol Π, its throughput closure Π' achieves the same or lower latency as Π.*

Proof. Consider an execution $\mathcal{E}'$, an adversary $\mathcal{A}'$ for protocol Π', and a transaction tx that has not already been Π'-*delivered*. Denote by $\mathcal{E}$ the equivalent execution (Definition 13) of protocol Π. Note that by definition of Π', tx has not been Π-*delivered* either (L15). Protocol Π' has two different mechanisms to Π'-*deliver*(tx).

On the one hand, if an event Π-*deliver*(b) for a block b containing transaction tx is triggered, then b is Π'-*delivered* (L15). In this case, *latency*$(\Pi', \mathcal{A}', \mathcal{E}', tx)$ is the same as *latency*$(\Pi, \mathcal{A}, \mathcal{E}, tx)$.

On the other hand, if an event Π-*deliver*(b') for a block b' that does not contains tx but is descendent of a block b containing tx., then block b' is Π'-*delivered* immediately before b (L16). In this case, *latency*$(\Pi', \mathcal{A}', \mathcal{E}', tx)$ is strictly smaller than *latency*$(\Pi, \mathcal{A}, \mathcal{E}, tx)$.

Both cases discussed above imply that for every adversary, execution, and transaction, the latency of both protocols satisfy $latency(\Pi', \mathcal{A}', \mathcal{E}', tx) \leq latency(\Pi, \mathcal{A}, \mathcal{E}, tx)$. Hence, $latency(\Pi') \leq latency(\Pi)$ $\qquad \Box$

The next result clarifies the motivation for the term throughput closure.

Lemma 4. *Given a DAG protocol Π, then throughput$(\Pi') \geq$ throughput(Π), and if Π permits abandoned blocks, then throughput$(\Pi') >$ throughput(Π).*

Proof. Consider an execution $\mathcal{E}'$, an adversary $\mathcal{A}'$ for protocol Π', and a transaction *tx* that has not already been Π'-*delivered*. Denote by $\mathcal{E}$ the equivalent execution (Definition 13) of protocol Π.

On the one hand, if there is no abandoned block in the execution $\mathcal{E}'$, then, the set *abandoned*(b') is empty for every block b'. Thus, no extra reference is added at any point in the execution of Π' the executions $\mathcal{E}$ and $\mathcal{E}'$ identical. We conclude that *throughput*$(\Pi, \mathcal{A}', \mathcal{E}) =$ *throughput*$(\Pi', \mathcal{A}, \mathcal{E})$.

On the other hand, if there exists at least one abandoned block b' in execution $\mathcal{E}'$, then, the set *abandoned*(b^*) is not empty for some block b^* that is eventually Π'-*delivered*. When b^* is Π'*delivered* so is b' (L16).

We conclude that *throughput*$(\Pi', \mathcal{A}', \mathcal{E}') \geq$ *throughput*$(\Pi', \mathcal{A}, \mathcal{E})$ for every possible adversary $\mathcal{A}'$ and execution $\mathcal{E}'$, thus *throughput*$(\Pi') \geq$ *throughput*(Π'). Furthermore, if Π permits abandoned blocks, there exists an λ-partial execution with a block b that is abandoned in all its extensions. This means that the probability, over the randomness of the protocol, of having an abandoned block is strictly greater than zero (Remark 1). Thus, $\mathrm{E}[throughput(\Pi', \mathcal{A}', \mathcal{E}')] > \mathrm{E}[throughput(\Pi, \mathcal{A}, \mathcal{E})]$ for at least some adversary $\mathcal{A}'$. We conclude by noticing that if an adversary $\mathcal{A}^*$ prevents the exclusion of abandoned blocks, then *throughput*$(\Pi', \mathcal{A}^*, \mathcal{E}') >$ *throughput*$(\Pi', \mathcal{A}', \mathcal{E}')$. Hence, we conclude that

$$throughput(\Pi') = \inf_{\mathcal{A}'} \mathrm{E}[throughput(\Pi', \mathcal{A}', \mathcal{E}')]$$
$$> \inf_{\mathcal{A}} \mathrm{E}[throughput(\Pi, \mathcal{A}, \mathcal{E})] = throughput(\Pi).$$

$\qquad \Box$

Corollary 5. *Given a DAG protocol Π, then goodput$(\Pi') \geq$ goodput(Π). Furthermore, if Π allows for the existence of abandoned blocks, then goodput$(\Pi') >$ goodput(Π).*

Proof. Consider the proof of Lemma 4 limited to adversaries that follow the instructions of the protocol. $\qquad \Box$

Note that every chain protocol trivially permits abandoned block. We can finally conclude that DAG protocols are strictly better then chain protocols.

Theorem 6. *Given a chain protocol Π, there exists a DAG protocol Π' such that: latency$(\Pi') \leq$ latency(Π) and throughput$(\Pi') >$ throughput(Π).*

Proof. Lemma 1 states that a chain protocol Π permits abandoned blocks. Theorem 2 demonstrates that its throughput closure Π' implements block-based atomic broadcast. Lemma 4 establishes that *throughput*$(\Pi') >$ *throughput*(Π). Finally, Lemma 3 shows that *latency*$(\Pi') \leq$ *latency*(Π). $\qquad \Box$

Acknowledgments

References

[1] "Chia network." `https://docs.chia.net/docs/01introduction/what-is-chia`.

[2] O. Alpos, I. Amores-Sesar, C. Cachin, and M. Yeo, "Eating sandwiches: Modular and lightweight elimination of transaction reordering attacks," *CoRR*, vol. abs/2307.02954, 2023.

[3] I. Amores-Sesar, C. Cachin, and E. Tedeschi, "When is spring coming? A security analysis of avalanche consensus," in *OPODIS*, vol. 253 of *LIPIcs*, pp. 10:1–10:22, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[4] V. K. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *CCS*, pp. 585–602, ACM, 2019.

[5] M. Bowman, D. Das, A. Mandal, and H. Montgomery, "On elapsed time consensus protocols," in *INDOCRYPT*, vol. 13143 of *Lecture Notes in Computer Science*, pp. 559–583, Springer, 2021.

[6] C. Cachin, R. Guerraoui, and L. E. T. Rodrigues, *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[7] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

[8] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient BFT consensus," in *EuroSys*, pp. 34–50, ACM, 2022.

[9] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT (2)*, vol. 10821 of *Lecture Notes in Computer Science*, pp. 66–98, Springer, 2018.

[10] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC*, pp. 165–175, ACM, 2021.

[11] I. Keidar, O. Naor, O. Poupko, and E. Shapiro, "Cordial miners: Fast and efficient consensus for every eventuality," in *DISC*, vol. 281 of *LIPIcs*, pp. 26:1–26:22, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[12] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.

[13] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *USENIX Annual Technical Conference*, pp. 515–528, USENIX Association, 2020.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." Whitepaper, 2009. `http://bitcoin.org/bitcoin.pdf`.

[15] C. Natoli and V. Gramoli, "The balance attack or why forkable blockchains are ill-suited for consortium," in *DSN*, pp. 579–590, IEEE Computer Society, 2017.

[16] F. Pedone and A. Schiper, "Generic broadcast," in *DISC*, vol. 1693 of *Lecture Notes in Computer Science*, pp. 94–108, Springer, 1999.

[17] A. Penzkofer, B. Kusmierz, A. Capossele, W. Sanders, and O. Saa, "Parasite chain detection in the IOTA protocol," *CoRR*, vol. abs/2004.13409, 2020.

[18] S. Popov, O. Saa, and P. Finardi, "Equilibria in the tangle," *Comput. Ind. Eng.*, vol. 136, pp. 160–172, 2019.

[19] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless BFT consensus through metastability," *CoRR*, vol. abs/1906.08936, 2019.

[20] Y. Sompolinsky, S. Wyborski, and A. Zohar, "PHANTOM GHOSTDAG: a scalable generalization of nakamoto consensus: September 2, 2021," in *AFT*, pp. 57–70, ACM, 2021.

[21] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography*, vol. 8975 of *Lecture Notes in Computer Science*, pp. 507–527, Springer, 2015.

[22] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *CCS*, pp. 2705–2718, ACM, 2022.