Blockchain and BlockDAG protocols

Chains to DAGs

Master Thesis

Renato Claudio Rao

Faculty of Science, University of Bern

Prof. Christian Cachin
Ignacio Amores-Sesar

December 8, 2023

Abstract

BlockDAGs have garnered significant attention in the realm of distributed ledgers as a potential solution to address the low transaction throughput of the longest-chain rule, as introduced in the ground-breaking Bitcoin whitepaper by Satoshi Nakamoto. We introduce the intricacies of BlockDAG protocols and highlight the utilization of BlockDAG protocols to achieve higher throughput compared to more traditional blockchain consensus protocols.

The core objectives are (i) the assessment of inefficiencies stemming from the longest-chain selection rule, (ii) the implementation of a modified longest-chain protocol transformed into a BlockDAG protocol, and (iii) the analysis of the implementation under various parameters.

This paper outlines the transformation of the NAKAMOTO consensus protocol into a DAG-based version, referred to as DAG-NAKAMOTO. The transformation is based upon a generic construct to transform a blockchain into a BlockDAG protocol, the transformation leads to increased throughput. Furthermore, the DAG-NAKAMOTO protocol is implemented in Python in the form of a simulation.

In conclusion, we present a detailed analysis of the DAG-NAKAMOTO protocol's performance through simulations, indicating that lowering the mempool similarity and optimizing the transaction selection strategy leads to increased throughput. The thesis suggests future work involving additional messaging mechanisms to further optimize transaction selection. In summary, this thesis offers a comprehensive exploration of a BlockDAG protocol and the potential to enhance throughput in distributed ledger technologies.

Keywords: Blockchain, Directed Acyclic Graph, Throughput, Latency, BlockDAG

Contents

1 Introduction

BlockDAGs are a topic of interest in the field of distributed ledgers. The longest-chain rule in Satoshi Nakamoto's famous bitcoin paper [9] suffers from low-transaction throughput by design. Participants in decentralized finance wish to transact values with safety and wish for fast settlements of funds. To address these inefficiencies in throughput, various BlockDAG protocols have been proposed [8] [10] [2]. Directed acyclic graphs (DAGs) are used to achieve a higher throughput than conventional blockchains. DAGs have been implemented at various levels of abstraction (e.g. at level of blocks or transactions, where transactions are connected by references) in distributed ledgers. Various DAGs offer different levels of settlement speeds and safety guarantees.

The core objectives of this thesis are (i) the assessment of inefficiencies stemming from the longest-chain selection rule, (ii) the implementation of a modified longest-chain protocol transformed into a BlockDAG protocol, and (iii) the analysis of the implementation with various parameters.

In Chapter 2, we introduce the theoretical background and notation that can be used to reason about single-chain protocols and BlockDAGs. Furthermore, we show an overview of various BlockDAG protocols. Then we introduce the proposed algorithm by [2] which provides a blueprint to increase throughput in existing blockchain consensus protocols by converting them to a DAG protocol. Measurements of efficiency are introduced to compare blockchain to BlockDAG protocols.

Furthermore, Chapter 3 shows the implementation of a BlockDAG protocol based on NAKAMOTO protocol. We highlight some attempts at evaluating existing protocols by recording propagated transactions and blocks. We introduce a simulation framework for simulating distributed systems in the context of blockchains. We outline the implementation of the proposed BlockDAG protocol.

Chapter 4 goes into depth on how parameters were chosen for the simulation. We present a simulation pipeline used to rigorously evaluate the implemented BlockDAG protocol with various parameters. Moreover, we show some outputs of the implemented simulation. The chapter ends with the results and the interpretation thereof.

2 Background

Consensus protocols ensure that a network of processes can agree on a common value, proposed by a participant, despite failing or malicious behaviour by a fraction of participating processes. A *safety* property ensures that nothing bad happens during the execution of a distributed algorithm. A *liveness* property ensures that eventually, something good happens. In the context of consensus protocol, an example of a safety property could be that all nodes agree on the same order of transactions. An example of a liveness property could be that any suitable transaction for inclusion might eventually be confirmed.

A blockchain protocol or distributed ledger technology (DLT) is *robust* if it satisfies liveness and safety against a Byzantine adversary ***at the same time***. The performance of a DLT is measured by the delivered transactions per second (*throughput*) and the time-to-delivery of a transaction (*latency*). In this work, we are interested in public DLT (as opposed to permissioned DLT). A DLT is *decentralized* if there is no central point of control or failure.

An ideal consensus protocol aims to build a robust ledger optimizing the following qualities: robustness, performance and decentralization. Consensus protocols are frequently evaluated by comparing their performance to that of centralized payment systems like Visa.

We will introduce the NAKAMOTO consensus protocol, then show how the generic throughput closure can be constructed [2]. Furthermore, we describe how we transform NAKAMOTO consensus protocol to a DAG version, called the DAG-NAKAMOTO consensus protocol. The DAG-NAKAMOTO protocol has higher throughput and lower latency with the same robustness as NAKAMOTO protocol. Furthermore, we will highlight the inefficiencies caused by the *longest-chain rule*.

2.1 Modeling distributed algorithms

In distributed algorithms, one way to model processes is to differentiate between *Byzantine* and *honest* nodes within a network based on how they behave. We are interested in how they execute some algorithm or protocol.

Byzantine nodes may behave arbitrarily or even maliciously. They may send incorrect, conflicting or malicious data to other nodes. Byzantine nodes may collude and execute an attack together. Collusion is modeled as an *adversary* that controls one or multiple executions of Byzantine nodes in an attack.

Honest nodes follow the protocol and do not deviate from the specified protocol. Honest nodes do not purposely interrupt the protocol being carried out.

The tolerance of Byzantine nodes is a central topic in DLT. Byzantine nodes should not be able to compromise safety or liveness of a DLT.

Another way to model processes is to model the participants as behaving *rational*. A rational node behaves in its self-interest on the rational assessment of potential cost and reward of an action. Rational nodes may deviate from the protocol, if it allows them to maximize their profit. The rational model is often used to study game-theoretic aspects of a protocol.

Note that rational behaviour may not necessarily imply Byzantine behaviour. Rational nodes are not inherently malicious. DLT must align protocol designs with rational behaviour of nodes.

2.2 Bitcoin protocol

In 2008, Satoshi Nakamoto [9] proposed a DLT maintained by an anonymous peer-to-peer network of nodes, which together agree on transactions to be included and the order in the ledger. Transactions are grouped into blocks identified by a SHA-256 hash [9], each block stores a *reference* to its parent block, effectively forming a chain of blocks, or a *blockchain*, and thus an order of blocks.

A node selects a set of transactions to include in a block, to create a valid block the node has to perform some computational workload - called proof-of-work.

The local view of the blockchain in a node is in reality a blocktree. This happens when multiple blocks reference the same parent. To ensure a globally consistent view NAKAMOTO consensus extracts a chain out of the tree of blocks.

To ensure robustness in the case of Byzantine nodes, the *longest-chain rule* (LCR) is used to select the *main chain* denoted by MC. The longest chain is determined by the computational work used to create it, which will be explained in Section 2.2.1. Blocks become *finalized* or *delivered* [1] when a certain block height k has been reached. We refer to k as the confirmation depth. When a block is delivered, so are the transactions contained within the block.

More formally, we define a block as a tuple $B = (h, T, \sigma, n)$, where h denotes the hash of a parent block, T a set of transactions to be included in the block, σ is a hash called the *Merkle Root* and n denotes a nonce. Elements within a block B will be denoted using a dot, for instance the hash h of block B will be represented as $B.h$. The Merkle Root, denoted by σ, is obtained through a hash function applied to the set T using a Merkle Tree structure, where each node is a hash of its children hashes. The leaves of the Merkle Tree are (transaction) hashes. The top or root of the Merkle Tree is also referred to as the Merkle Root. It is used to efficiently verify the integrity of a set of (transaction) hashes [9].

2.2.1 Proof-of-work

Proof-of-work (PoW) is a protection against post-factum modification of the order of transactions in a blockchain. Furthermore, PoW protects against Sybil attacks, where an attacker tries to influence a protocol by creating an overwhelming amount of pseudonymous identities. Due to the high cost of influencing the protocol, the attack becomes prohibitively expensive. Proof-of-work asks a node or *miner* to solve an intractable problem which is believed to take an exponential amount of steps. On the other hand, a node verifying the solution will use constant time. Parameter d denotes the *difficulty* a larger d makes it harder (longer) to solve the problem, a smaller d makes it easier (faster). The expected amount of steps to solve the problem is 2^d computational steps.

[1] Depending on the consensus mechanism, finalization is a probabilistic guarantee of the irreversibility of block-ordering in a security parameter ϵ

Concretely, in the NAKAMOTO consensus protocol [9] the intractable problem is calculating the input x to a hash function $H : \{0,1\}^* \to \{0,1\}^{256}$ (in this case concretely SHA-256) such that: (1) the output $y = H(x)$ and (2) that y has a prefix of d zeros. The input is structured as follows: $x := a||n$, where a is a given input string and n is a nonce. The input string a is a string representation in a deterministic format: $a := B.h||\sigma$, where B is the block that is being extended.

$|\textit{zeroPrefix}(y)|$ is the length of the zero prefix. Thus the probability of finding a x that is correct is:

$$P[H(x) = y, |\textit{zeroPrefix}(y)| = d] = \frac{1}{2^d}$$

Finding a correct hash results in a new block $\hat{B} = (B.h, T, \sigma, n)$, with $y = H(B.h||\sigma||n)$ and $|\textit{zeroPrefix}(y)| = d$. An honest node will propagate $\hat{B}$ containing the found nonce.

The hash function H is assumed to generate uniformly random outputs, thus the probability of collisions is negligible and will not be further considered. If the developer of a protocol was interested in controlling the rate at which correct hashes were found, there are two ways: by lowering the difficulty d or by lowering the run-time complexity of the hashing function.

2.2.2 Node

A node or miner (also called a *process*) is a participant in running the consensus protocol. Depending on the consensus mechanism, the nodes have varying roles. In NAKAMOTO consensus, a node propagates messages and mines new blocks, extending upon other blocks.

Definition 1 (Memory Pool or mempool)**.** *The memory pool of node u, denoted by M_u, is a data structure used for the temporary storage of transactions until they are selected to be included in a block. Each node within the protocol maintains its own memory pool. M_u is a set that contains undelivered transactions.*

A node u in the set of all participating nodes $\mathcal{N}$ has a local view V_u^t of a *blocktree* at time t and mempool M_u^t at time t. If time t is irrelevant it will be omitted from the notation. Every node needs to manage the local blocktree to apply the longest-chain rule.

When mining a new block, a node selects a block B to reference for generating the input string a. Let $\mathcal{B}$ be the set of all blocks. We model the selection of the block B that is being extended as a function $sel : \mathcal{B}^n \to \mathcal{B}$ which extracts a block $B \in V_u^t$ from the blocktree V_u^t, where n is the size of V_u^t.

2.2.3 Longest-chain rule

The longest-chain rule (LCR) dictates which blocks a node considers to be part of the *main chain*. The *longest chain* is the longest path from genesis to a leaf block in the blocktree V_u^t. In this work, we will assume constant difficulty d, otherwise the heaviest chain (weighed by computational effort used) may not be the longest (measured by the number of blocks in the chain). There might be two chains C_1 and C_2 with C_1 being longer than C_2, but C_2 may have had a higher difficulty and thus more computational effort was needed to create it. With this assumption, the *longest chain* will also be the *main chain*.

The goal of the longest-chain rule is to create a global and consistent blockchain among all nodes $u \in \mathcal{N}$ participating in the protocol.

The LCR introduces a trade-off in security versus scalability. The LCR introduces inefficiencies by constraining all nodes to mine to extend the same block. All honest nodes ideally receive newly mined blocks before being able to extend the chain. The scalability becomes limited because more forks would lead to less computational power in the main chain and as such worsen the security of the protocol against adversarial behaviour [11]. A lower mining rate leads to less computational power being wasted on side chains, which will be abandoned eventually, due to the LCR. Thus artificially suppressing the throughput

of the system by selecting an arbitrary, but secure delay for the amount of time passed between newly mined blocks ($\approx 600s$).

In NAKAMOTO consensus blocks become finalized or confirmed after reaching a certain *confirmation depth*, denoted by k, in the main chain. The *confirmation depth* denotes how many blocks k have to be mined after a block B, for B to be confirmed. In the case of NAKAMOTO consensus $k = 6$ [9]. The probability of an attacker reversing the order of transactions in the chain drops exponentially with higher k.

2.2.4 Forks

A *fork* is created when two honest nodes generate (by solving PoW) and propagate valid blocks B_1, B_2 through the network, referencing the same parent, that is, when $B_1.h = B_2.h$. This occurs when both blocks are mined at roughly the same time. The two blocks both contain a set of transactions $B_1.T, B_2.T$ with potentially *conflicting* transactions. Conflicting means that the transactions may spend the same resources. Since both blocks are valid, the chain is split into two competing sides. Furthermore, blocks B_1, B_2 can not both be included by the protocol because they do not form a blockchain, thus the processes have to decide on including either one of the competing blocks.

Another reason to not include both blocks B_1, B_2 is that the inclusion of a block in the blockchain generates a reward transaction, thus we would have two competing reward transactions.

Thus, the longest-chain rule forces the nodes to select one of the competing blocks and start a new PoW process.

2.2.5 Network layer

Miners exchange messages such as blocks and transactions through the *network*. The most important information nodes share in the form of *messages* are blocks and transactions. Messages are *gossiped* between nodes in the network. When a node joins the network, it retrieves a list of bootstrap peers and contacts them[2].
After joining, the node learns about other peers by querying its bootstrap peers and by receiving transactions from new peers [4]. Every node randomly selects a number $\geq p$ peers to randomly communicate with. Possible partitions in the connection graph are not actively detected. If such a partition occurs, the protocol will continue to operate independently. Sharing blocks and transactions is a three-step process:

1. A node will advertise its *inventory*, which indicates that the node has a block or transaction to share
2. Another node interested in the information may request it from the advertising node.
3. The advertising node will deliver the information unit or respond with *not found*.

In this work, we will simplify the three-step process to a two-step process. Nodes *send* a message and eventually, an event *receive* will be triggered. The time that passes between the two events is called *transaction delay*.

A node u verifies the information it receives against the local view V_u^t, independently from the other nodes before advertising it again. If the information is not consistent with the local state of the node it is not further propagated.

The network can be modeled as a graph with degree $\geq p$ with nodes and links $G_{Network} = \langle N, L \rangle$.

[2]A hardcoded list of IPs stored in the source code[4]

2.2.6 Transaction selection

More formally, we define a transaction as a tuple $tx = (id, sender, receiver, value, size, fee)$

A miner u selects valid transactions from its mempool M_u^t to include in the next block. Nodes that behave rationally will aim to maximize the fees attached to a transaction $tx.fee$ to optimize the reward:

$$reward(txSelection(M_u)) = \sum_{tx \in txSelection(M_u)}^{\infty} tx.fee$$

The amount of transactions $|B.T|$ in a block B is limited by the size of the transactions *tx.size* that fit in the maximum size of a block, $BMAXSIZE = 1$ MB, defined by the protocol:

$$\sum_{tx \in txSelection(M_u)}^{\infty} t.size \leq BMAXSIZE$$

The goal is to maximize the reward through transaction selection, while respecting the size limitation:

$$\arg\max_{txSelection(M_u)} reward(\cdot)$$

This problem of optimizing a reward while respecting a limit reduces to *0/1 knapsack problem* and is thus NP-hard. In practice, a greedy algorithm is employed with polynomial running time.

Definition 2 (Greedy selection strategy)**.** *The greedy selection strategy γ is achieved by ordering transactions in mempool M_u by their fee in descending order. A node iterates through the ordered list and selects transactions until BMAXSIZE is reached.*

Additionally, the validity predicate $VT(tx)$ must hold for every transaction to be selected. A transaction is valid if its signature is valid and if it has not been delivered by another node and the participants have the necessary assets they wish to transact on.

2.2.7 Validity of transactions and blocks

In this work, we use the following properties to determine the validity of blocks and transactions. The transaction validity is determined based on the assumption that a determined balance exists for the sender and receiver.

Definition 3 (Transaction Validity)**.** *A transaction tx in the* NAKAMOTO *consensus protocol is valid if:*

- *Sender has sufficient balance.*
- *The tx.id has not been delivered already.*

For a valid transaction tx, it holds that: $VT(tx) =$ TRUE.

To check the validity of received transaction a node has to store all transactions and iterate through them. This can incur significant cost in terms of both (disk) space and time. Similarly a block B in NAKAMOTO Consensus is valid if:

Definition 4 (Block validity)**.** *A block is valid if:*

- *All transactions included in a block must be valid. $\forall tx \in B : VT(tx) =$* TRUE.
- *The sum of the size of transactions is smaller than the block limit.*
- *The same block with hash $B.h$ has not been delivered before.*

For a valid block B the block, it holds that: $VB(B) =$ TRUE.

2.2.8 Attack on bitcoin

In a *double-spend attack* (DS Attack), an adversary tries to replace the existing main chain *MC* with some chosen set of blocks *HC*. In this attack, a malicious miner may silently mine blocks (without broadcasting them) and release them in a burst of blocks at once. It is computationally very expensive to do so, the attacker needs to outpace the growth of the main chain with his hidden chain *HC*. Upon the release of *HC*, the honest nodes would continue to extend *HC* due to the LCR and overtake the existing, honestly generated chain.

2.3 BlockDAG protocols

Different protocols that address the inefficiencies of the *LCR* while maintaining security. The protocols can be broadly divided into three kinds: Parallel Nakamoto Chains [5], Variations of GhostDAG and Inclusive protocols [2].

Parallel nakamoto chains aim to avoid the scalability-security trade-off all together. The ledger is divided into k separate Nakamoto chains. Mined blocks are randomly (for instance, by using the generated hash as source of randomness) assigned to one of the k chains. The random selection process leads to prohibitively expensive costs associated with a double-spend attack. Since an attacker would have to generate enough blocks on all the parallel chains to influence the block ordering. A total-order over all k chains can be achieved, for instance, by ordering blocks by timestamp (which assumes synchronized clocks). Prism [3] enables higher throughput by deconstructing the basic blockchain into its atomic functionalities. This is done by dividing blocks into different block types by their purpose: transaction blocks, proposal blocks and voter blocks. Transaction blocks hold transactions and can be created in very short intervals. Proposer blocks reference transaction blocks. Voter blocks are used to order the proposer blocks and thus the transactions. The proposer and voter blocks are stored in separate chains and are created through proof-of-work (with independent difficulty for both chains). Furthermore, the voter chain can be divided into many parallel chains which together vote on a single proposer block, this leads to very fast transaction confirmation and higher throughput.

Variations of GHOST: GHOSTDAG, PHANTOM GHOSTDAG: Generally, for protocols in the GHOSTDAG family, the goal is to tolerate more forks and achieve a higher throughput by doing so.

The work GHOST [11] is an acronym for "Greedy Heaviest-Observed Sub-Tree"[3] GHOST is a fork-choice rule that is used instead of the LCR. The idea is that blocks that are not in the main chain still contribute to the weight of the chain. Weight here refers to the computational effort that was used to create the sub-chain. The GHOST fork-selection rule itself is only distantly related to a DAG consensus protocol.

GHOSTDAG [10] aims to avoid the security-scalability trade-off of which NAKAMOTO consensus suffers from. The blocks in GHOSTDAG are stored in a DAG structure, every block has a set of references to multiple other blocks (there is no dedicated parent). The tips of the DAG are the latest mined blocks. In GHOSTDAG a mined block will reference all the tips (latest mined blocks) the miner has "seen". The goal of the GHOSTDAG protocol is to classify mined blocks as either red or blue; red blocks have been likely mined by malicious nodes; blue blocks have likely been mined by honest nodes. The coloring algorithm is fairly involved, the idea is to analyze how the blocks are connected and if a certain block A has not been "seen" or referenced by other blue blocks that should have seen the block, then A was withheld by an attacker and thus colored red.

Inclusive protocols aim to change the LCR such that blocks that were forked can be included in the main chain. The local blocktree of every node is transformed into a DAG. A deterministic version of a topological sort is used to finally order the blocks. The analysis of Inclusive Protocols [8] shows that the *probability* of a successful double-spend attack on an inclusive protocol is the same as the non-inclusive

[3] It is suspected that the acronym was chosen *a priori*.

protocol. Additionally, they show that the *cost of an attack* in the rational model can be made prohibitively high.

This thesis will analyze and implement the protocol proposed by Amores-Sesar and Cachin [2], which can be thought of as a black-box recipe to transform a given blockchain protocol Π to an inclusive protocol Π' using a DAG.

2.3.1 Directed Acyclic Graphs

For directed graphs to be useful in the context of DLT, they need to fulfill some properties to enable the protocol to carry out its essential tasks. A DLT has to establish a unique total-order over all transactions. For this, a directed graph needs to be acyclic and, furthermore, have a certain structure. We present how a DAG is defined and the conditions under which a DAG can be ordered topologically.

Definition 5 (Directed Graph)**.** *A directed graph is a Graph $G = (V, E)$, with V denoting the set of vertices and E the set of directed edges.*[4] *[12]*

Definition 6 (Cycle in a Graph)**.** *A path is an ordered set of directed edges connecting a set of vertices, e.g. $a, b, c \in V$ and $(a, b), (b, c) \in E$. A path forms a cycle if it is non-empty and the starting vertex of the first edge equals the ending vertex of the last edge. A graph is said to be cyclic if it contains one or more cycles.*

Definition 7 (Directed Acyclic Graph (DAG))**.** *A DAG is a directed graph with no cycles.*

A DAG G induces a partial order ($\preceq$) on the vertices. Two vertices $a, b \in G$ are said to be partially ordered $a \preceq b \Leftrightarrow \exists path\ (a \to b) \in G$. [12]

Definition 8 (Hamitlonian path)**.** *A **Hamiltonian path** is a path in a directed or undirected graph that visits each vertex exactly once.*

Definition 9 (Topological sorting)**.** ***Topological sorting** is the algorithmic problem of finding a total ordering.*

The term *total ordering* is used slightly differently in the context of graphs than in distributed systems. In a distributed system a total ordering implies that all processes achieve the same ordering, whereas in a graph there may be multiple total orderings.

Theorem 1. *A total ordering is possible if and only the Graph G is a DAG [14].*

Note that in undirected graphs or cyclic directed graphs, no such total ordering is possible.

Lemma 1. *A DAG may have multiple total orderings if it has no Hamiltonian path; otherwise, the total ordering is unique [14].*

Topological sorting algorithms have a time complexity in the order of vertices and edges $O(|V| + |E|)$.

2.4 Construction of a BlockDAG protocol

The *throughput closure* is a construction that takes as an input a blockchain protocol Π with abandoned blocks and creates a new inclusive BlockDAG protocol Π' which aims to include abandoned blocks in the ledger. This is achieved by modifying the way Π' delivers blocks compared to Π [2].

The following construction will be made on the example of a PoW protocol as Π. The construction can be applied to other forms of consensus, such as Proof-of-Stake or Proof-of-elapsed-Time [2]. The sub-sections subsequent to the construction will consider the throughput closure based on NAKAMOTO protocol as Π.

[4] The graph may not contain multiple directed edges with the same source and target vertice.

2.4.1 Throughput closure

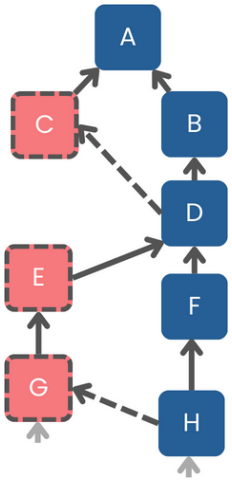

Figure 2.1: An example of the throughput closure of NAKAMOTO consensus. Full arrows denote a reference to the parent of a block. Dashed arrows denote the references to abandoned blocks, defined by the throughput closure. The main chain is marked by blue blocks (full-lined border). Red blocks (dashed border) are abandoned blocks. For block D the set of abandoned blocks is exactly the block C, written $\textit{abandoned}(D) = \{C\}$.

The throughput closure, as the name suggests, is a construct that improves throughput by converting a blockchain protocol Π to an inclusive BlockDAG protocol denoted by Π'. This is done by modifying the block delivery. More formally, every honestly mined block is eventually delivered in Π'. However, this is not the case in a blockchain protocol Π, since blocks may be forked.

We will explain a transformation of NAKAMOTO consensus protocol as Π to the BlockDAG protocol Π' which will be implemented in this thesis.

The throughput closure allows blocks to reference multiple parents by adding extra references to abandoned blocks. For each block B, there is a set of $\textit{abandoned}(B)$ which is a set of blocks (which must contain at least one valid transaction) that cannot be included in the main chain if B is included. In NAKAMOTO consensus the set $\textit{abandoned}(B)$ is easy to construct. Recall a Block in Π is a tuple $B = (h, T, \sigma, n)$. Thus a block B' in Π' is defined as an extension of a block B in PoW with a set of references denoted by R. Formally, $B' = (h, T, \sigma, n, R)$. Where $R = \{H(b) | b \in \textit{abandoned}(B)\}$ is the set of references to/hashes of abandoned blocks. The acceptance of blocks of Π is modified to accept blocks with conflicting transactions.

When a block B' is delivered, the set of references R is delivered before B' in a deterministic order. The order needs to be deterministic, such that the local block-trees of the nodes do not diverge. The inclusion of the set of references effectively converts the blockchain in Π to a BlockDAG in Π'.

To construct the set of $\textit{abandoned}(B)$, a node has to listen and store all blocks that are propagated, even blocks that may not be part of the main chain. Figure 2.1 shows an example of the throughput closure.

When the set of $\textit{abandoned}(B) = \emptyset$ is empty for all blocks in the execution, Π' reduces to Π. On the other hand, if there exists a block in the set $\textit{abandoned}(B)$ the throughput closure Π' does not reduce to Π. In that case, the original paper formally shows that the throughput of Π' will be strictly higher than the throughput of Π.

2.4.2 Topological ordering

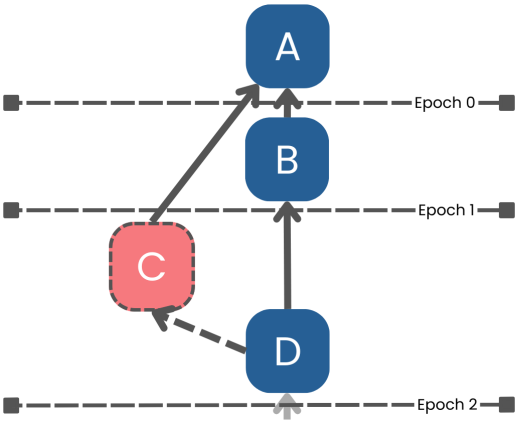

Figure 2.2: Consider a BlockDAG with four blocks A,B,C,D. Where blue blocks A, B, D (full-lined border) are blocks in the main chain and red (dashed border) block C is a forked block. The delivery of the blocks is done according to the rules defined. Here an epoch can be thought of, as all the blocks that are delivered due to a block in the main chain. Thus the block C referenced by D in Epoch 2 is delivered before D. The unique total ordering is (from first to last delivered) is: A, B, C, D.

In a blockchain, the ordering of blocks is trivial, given a deterministic sorting function. To sort a DAG we use a topological sorting. Depending on the DAG structure, the topological sorting may not be unique. As seen in Definition 9 if a DAG contains a Hamiltonian path, then the topological sort order is unique.

The protocol must define rules to reduce a BlockDAG to a unique total ordering. In this case, we define the following rules to break the ties and produce a unique total order:

(1) Block B' references a set of abandoned blocks $R = \{b' \in B'.R\}$. The blocks in R are delivered before the parent block.

(2) The blocks in R are delivered in an order determined by a deterministic ordering function, e.g. by ascending order of their respective block hashes.

An abandoned block $b' \in B'.R$ may further contain references to abandoned blocks. Block b' might reference them as parent or as abandoned blocks. To deliver blocks that were referenced by abandoned blocks in a unique order, we follow the path of references (if there are multiple references to follow, the same deterministic ordering as in rule (2) is used) until we reach a block that references the main chain. Finally, the blocks are delivered by following the rules (1) and (2) until all paths have been resolved.

We define an *epoch* as all the blocks that are delivered due to a block in the main chain. As such an epoch contains all abandoned blocks referenced by the block in the main chain and recursively referenced blocks.

The construction shown in Figure 2.2 shows that the DAG forms a Hamiltonian path from the last epoch to the genesis block A.

2.4.3 Validity of transactions and blocks

In the transformed protocol Π' the validity of transactions (**Definition 3**) remains unchanged.

The block validity predicate from **Definition 4** has to be relaxed since a forked block may contain conflicting transactions and is still deemed valid. Thus the first condition of the validity predicate

$\forall tx \in B \implies VT(tx) = \text{TRUE}$ for a block B' becomes:

$$\exists tx \in B' \text{ such that } VT(tx) = \text{TRUE}$$

The relaxed block validity predicate is denoted by $VB'(B')$.

2.4.4 Implicit changes

Converting a blockchain protocol Π to a BlockDAG protocol Π' has some implications for the implementation of the protocols.

On the network layer of a BlockDAG protocol Π' the message complexity will be higher for consensus. Consider a fork with two blocks B_1, B_2 in Π and the identical blocks B'_1, B'_2 in Π'. For example in Nakamoto Consensus, an honest node that receives block B_1 before the forked block B_2 will not further propagate B_2 to other nodes [4], since the block is not deemed valid anymore. Conversely in Π' an honest node must propagate both blocks to all other honest nodes; otherwise, the honest nodes may never receive the forked block B'_2 that might be referenced by a subsequent block $B'_3.R = \{B'_2\}$.

Similarly, in NAKAMOTO consensus, a transaction tx_1 that has become invalid will not further be gossiped by an honest node.

In Π', the same transaction is gossiped, assume the transaction became invalid due to the inclusion in a block in the main chain and is also included in a forked block. If nodes in Π' wish to verify the block hash of a forked block that builds on the Merkle root, they may need the invalid transactions as well. Thus the nodes may need to gossip invalid transactions as well.

2.5 Measures of efficiency

We want to compare the two protocols with different measures to assess the inefficiencies introduced due to the LCR. While NAKAMOTO and DAG-NAKAMOTO both use the LCR, DAG-NAKAMOTO does so without excluding forked blocks.

Measurement	Defined in NAKAMOTO	Defined in DAG-NAKAMOTO
Fork Rate	✓	✓
Latency	✓	✓
Throughput	✓	✓
Mempool Similarity	✓	✓
Conflict inclusion rate	–	✓
Reference inclusion rate	–	✓

Table 2.1: Metrics that were defined to assess the efficiency of the protocol. Inclusion rates can only be calculated for the DAG-NAKAMOTO protocol

To measure the inefficiencies arising from excluded forked blocks, the metrics listed in Table 2.1 were introduced. All the measurements are calculated per process and then averaged over the entire set of processes. Recall the main chain of node u is denoted by MC_u and blocktree by V_u. The blocktree contains all the blocks a node has received.

Definition 10 (Fork rate)**.** *The* fork rate *for node u is defined as*

$$\mathit{forkrate}(u) = 1 - \frac{|MC_u|}{|V_u|}$$

The fork rate is then averaged over the entire set of processes participating in protocol Π*:*

$$\mathit{forkrate}(\Pi) = \frac{1}{|\mathcal{N}|} \sum_{u \in \mathcal{N}} \mathit{forkrate}(u)$$

Latency measures the time from when a transaction tx was initially broadcasted until the transaction is delivered. The $\mathit{latency}(\Pi)$ is averaged over all transactions. For this, we extend the model of a transaction with two additional attributes: *creation_time* and *delivery_time*. The *creation_time* denotes the time at which the transaction was created by a participant in the protocol. The *delivery_time* denotes the time at which a block containing the transaction has reached the confirmation depth $k = 6$.

Definition 11 (Latency)**.** *The latency of a transaction tx is defined as:*

$$\mathit{latency}(tx) = \mathit{tx.delivery\_time} - \mathit{tx.creation\_time}$$

Definition 12 (Throughput)**.** Throughput *is measured in transactions delivered per second (TPS).*

Definition 13 (Mempool similarity)**.** Mempool similarity *is used to measure how similar the mempools of different nodes are, using the Jaccard similarity coefficient. The Jaccard similarity coefficient is a statistic to measure the similarity between two finite-sized sets [13]. By comparing which transactions are in both sets (intersection) divided over all the transactions in both sets (union).*

$$\mathit{mempoolSimilarity}(u, v) = J(M_u^t, M_v^t) = \frac{|M_u^t \cap M_v^t|}{|M_u^t \cup M_v^t|}$$

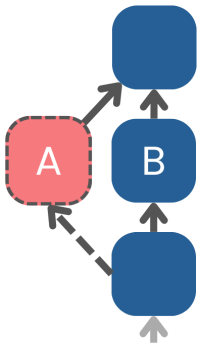

Figure 2.3: Simple execution of DAG-NAKAMOTO. Blue blocks are part of the main chain, and red blocks are forked blocks. The arrows with full lines are references of NAKAMOTO protocol, whereas the arrows with dashed lines are references introduced by DAG-NAKAMOTO.

To measure the performance of DAG-NAKAMOTO we introduce two extra metrics. Consider an execution of DAG-NAKAMOTO as shown in Figure 2.3. The two blocks A, B with following sets of transactions: $A.T = \{tx_1, tx_2, tx_3\}$, $B.T = \{tx_2, tx_3, tx_4\}$. In the execution of NAKAMOTO transactions of either block A or block B would be included, but not transactions of both. In DAG-NAKAMOTO, both sets of transactions are included.

Definition 14 (Conflict inclusion rate)**.** *The conflict inclusion rate determines how many of the transactions in a referenced (dashed line) block are included by a competing forked block and thus do not contribute to a higher throughput.*

$$\mathit{conflictInclusionRate}(A, B) = |A.T \cap B.T|/|A.T|$$

The conflict inclusion rate of blocks A and B, shown in Figure 2.3, would thus be:

$$\textit{conflictInclusionRate}(A, B) = \frac{|A.T \cap B.T|}{|A.T|} = \frac{|\{tx_2, tx_3\}|}{3} \approx 66\%$$

Note that instead of the set $B.T$ in the simulation we use all the transactions delivered before a forked block A is delivered, to more accurately represent the performance. The transactions delivered before A are denoted as *deliveredTxBefore(A)*.

$$\textit{conflictInclusionRate}(A) = |A.T \cap \textit{deliveredTxBefore}(A)|/|A.T|$$

Definition 15 (Reference Inclusion Rate)**.** *The reference inclusion rate defines the rate of transactions, in a forked block A that were included due to the references and would not have been included in the* NAKAMOTO *protocol.*

$$\textit{referenceInclusionRate}(A, B) = \frac{|A.T \setminus B.T|}{|A.T|}$$

In the simulation, we calculate the reference inclusion rate based on the transactions not contained in the set of already delivered transactions $deliveredTxBefore(A)$.

$$\textit{referenceInclusionRate}(A) = \frac{|A.T \setminus deliveredTxBefore(A)|}{|A.T|}$$

Going back to the example with blocks A and B, shown in Figure 2.3, the reference inclusion rate would be:

$$\textit{referenceInclusionRate}(A, B) = \frac{|A.T \setminus B.T|}{|A.T|} = \frac{|\{tx_1|\}}{3} = \approx 33\%$$

Note that there may also be a percentage of invalid transactions due to conflicts with earlier transactions.

In practice, there are multiple models to track the tokens that a participant in the protocol can transact with. In the NAKAMOTO consensus protocol the "Unspent Transaction Output" (UTXO) model is used. In this model, tokens are stored as a list of UTXOs, where each UTXO represents a reference to an output (similarly to the "change" received when spending a 50$ bill) that can be spent in the future. In the account-based model, each participant has a balance associated with their account. Transactions in this model decrease the balance of the sender and increase the balance of the receiver.

Consider two transactions tx_1, tx_2 and two participants u, v with u having a balance of 10 and v having a balance of 0. Written as $[u : 10, v : 0]$.

$$tx_1 : u \rightarrow_{10\ USD}\ ^{5} v$$

$$tx_2 : v \rightarrow_{10\ USD}\ u$$

If tx_1 is delivered before tx_2:

$$tx_1 : [u : 0, v : 10]$$
$$tx_2 : [u : 10, v : 0]$$

On the other hand if tx_2 is delivered before tx_1:

$$tx_2 : [u : 20, v : -10] \implies VT(tx_2) = \text{FALSE}$$
$$tx_1 : [u : 0, v : 10] \implies VT(tx_1) = \text{TRUE}$$

As a result, the order of transaction inclusion can influence their validity. If tx_1 is delivered before tx_2, both transactions are valid. If tx_2 is delivered before tx_1, tx_2 becomes invalid. It's important to note that we do not consider this form of invalidity in the simulation. This type of invalidity can occur in both the UTXO and account-based models.

[5] u sends 10 USD to v

Remark 1 (Upper- and lower-bounds for TPS in DAG-NAKAMOTO)**.** *The throughput of an inclusive BlockDAG protocol can be upper- and lower-bounded as follows:*

The lower bound on TPS is the execution of the original protocol Π*, thus no transactions from forked blocks are included - in the case of* DAG-NAKAMOTO *protocol this would be the TPS bitcoin achieves.*

The upper bound on TPS is an execution where all transactions from forked blocks are included - denoted as the optimal TPS.

2.6 Security against block reordering

By construction, the security of the throughput closure Π' reduces to Π [2].

Thus, the security against an attacker attempting to forcefully reorder blocks (and thus transactions) is the same for its throughput closure Π' as for the original Π.

A successful attacker cannot arbitrarily change transactions. The attacker aims to: (1) in the case of NAKAMOTO protocol exclude blocks from the total-ordering (2) in the case of DAG-NAKAMOTO protocol change the ordering of the blocks in the total-ordering.

In DAG-NAKAMOTO protocol, if an attacker chain HC surpasses the honest chain, the transactions in the honest chain would still be included by honest blocks that will be eventually mined. This results in a re-ordering of the transactions and may invalidate some transactions. Thus in an attack against DAG-NAKAMOTO protocol, transactions may be excluded due to their invalidity highlighted in Section 2.5. Nonetheless, in a successful attack, this would possibly result in double spending.

As the analysis of the Bitcoin backbone protocol [6] shows, the security of the NAKAMOTO protocol in the synchronous and rushing adversary setting with $f < |\mathcal{N}|/2$ Byzantine nodes holds, as long as the expected amount of blocks mined from Byzantine nodes remains below the number of non-forked blocks, a double-spend attack is improbable.

Thus, the throughput closure implemented in DAG-NAKAMOTO guarantees strictly higher throughput with the same security as NAKAMOTO [2].

Nakamoto in his original paper [9] shows that the probability of an attacker overtaking the honest chain, drops off exponentially with higher confirmation depth k.

Furthermore, the Bitcoin backbone paper [6], shows that the probability of at least an honest party mining a block in a round (denoted by α)[6] has a strong impact on the share of Byzantine nodes that can be tolerated by the protocol.

It is shown that if α in a round is close to 0 ($\alpha = 0$, implies that the protocol is not life), the protocol can tolerate $f < |\mathcal{N}|/2$. On the other hand, if we increase the rate at which a correct proof-of-work is found, with $\alpha > 0.1$ the adversarial bound drops below 50%. In practice, the aim of the protocol is to calibrate α, such that: $\alpha \in (0, 1)$. In Bitcoin α is in the range of 0.02 - 0.03 with a round taking on average $20s$ [6].

[6] In the original paper denoted as f

3 Implementation

In this chapter, we introduce BlockSim [1] the simulation framework. Furthermore, we show the base model of BlockSim has been adapted to a BlockDAG protocol. Formally, the following sections address instantiations of the NAKAMOTO protocol Π and DAG-NAKAMOTO protocol Π'. The first section shows some other attempts that were made at evaluating a BlockDAG protocol Π' in this thesis.

3.1 Attempts at evaluating a protocol Π'

The initial idea was to set up a node in a public DLT and listen to messages in the network. In the unpermissioned DLT case, this should be fairly straightforward. With the recorded blocks and transactions of a protocol Π, one could simulate the throughput closure Π' and measure various efficiency metrics.

Two options were considered: (1) finding recorded data about broadcasted blocks and transactions (2) running a node that monitors the broadcasts in the network.

For the latter, there are different approaches to accomplish listening to the network traffic that differ in complexity and cost. It involves hosting a node:

Self-hosting a node is actually fairly complex in the case of Ethereum. The hardware requirements are costly depending on which Ethereum network, one would like to host a node. To run a node on Goerli (testnet) the server needs to have at least $16GB$ RAM and $\geq$ $100GB$ SSD and $10MB/s$ bandwidth. Depending on the cloud provider this can cost around $50 - 100$ USD per month. Furthermore, configuring the consensus and execution layers to interact correctly turned out to be more difficult than anticipated. This is contrary to what one would hope to be true for a public DLT.

Dedicated node as a service is a more scalable approach in the sense that it allows booting a dedicated node configurable for various blockchain protocols with a few clicks. It is fairly costly, at around 50 USD/day.

Hosted node with a REST-API has either a public or private REST-API that is exposed to query transactions. For instance, the REST-APIs of Infura[1] in the free tier allows for 25'000 archive requests on Ethereum. One request in Ethereum is, for example, a function call to *eth_getTransactionByHash(h)* in the API.

[1] `https://infura.io`

The first approach has been evaluated thoroughly. The first problem was that the University network is behind a firewall, which prohibits nodes from exposing ports to the public (that should be available to peers outside the university network). Subsequently, an attempt was made to use a Virtual Machine (VM) hosted on DigitalOcean. While this solution allowed the VM to accurately record blockchain blocks, a new issue arose in the form of rapidly depleting storage space. In fact, the utilization of hard drive space escalated to such an extent that it eventually rendered even the basic SSH login functionality inoperable.

A third approach was also evaluated: a listener was written in Golang by using a web socket and listening to new blocks announced on the Ethereum test network. The recorded fork rate was very low and thus leaves little space for improvement by an inclusive protocol. In the end, we opted for BlockSim to have more fine-grained control over parameters in the protocol.

3.2 BlockSim

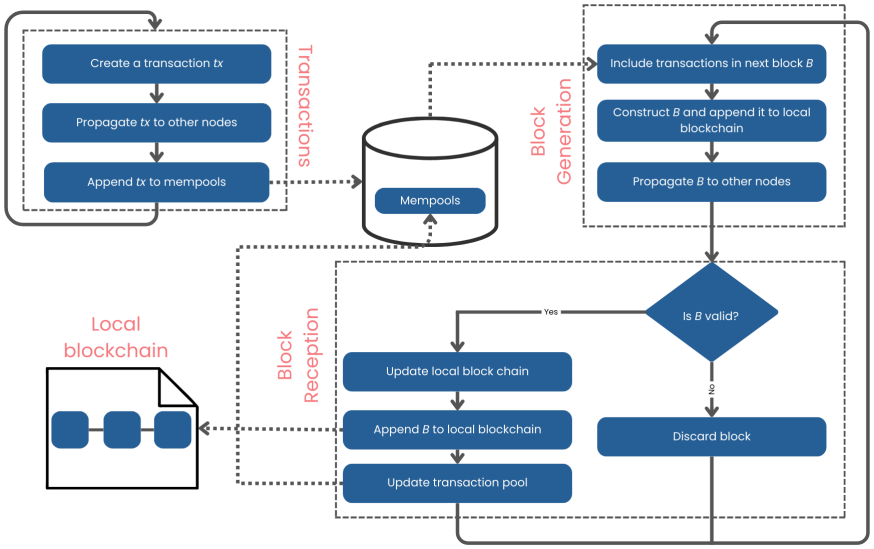

Figure 3.1: Workflow for the consensus activities within the Base Model of BlockSim [1]. The essential activities involve transaction generation, block generation, block reception and the management of the local blockchain and the mempools.

BlockSim [1] is an extensible simulator for blockchain systems. BlockSim defines a Base Model and its parameters that can be easily adjusted for various studies of interest. BlockSim is structured into three abstract layers the (1) network , (2) consensus and (3) incentives layer. Furthermore, BlockSim aims to provide similar results, for a variety of metrics, in the simulation models as their real-world counterparts.

BlockSim allows us to easily to simulate a number of nodes without the complexity of orchestrating and administrating a cluster of nodes.

Thus, Blocksim simplifies much of the underlying complexity of implementing the different layers. For example, in a productive implementation of a blockchain protocol, the network layer would have to implement several functionalities, such as: Node discovery, Serialization of messages and Deserialization and Reliable Broadcast.

In the simulation, it suffices to calculate the propagation delay for transactions and hide other details of a real execution.

Figure 3.1 shows a more detailed overview of the activities involved in the consensus layer. More formally, the simulation can be thought of as an environment $\mathcal{Z}$ in [6] which controls all nodes $u \in \mathcal{N}$. The simulation employs a *discrete-event paradigm*; each event occurs at a particular instant in time and makes a state change. Between events, the system remains unaltered. After simulating one event at t_1, the clock can directly jump to the next event at t_2. All measurements are based upon the passing of simulated time; it is important to note that the simulated time may not necessarily align with the real-time passage. The relationship between simulated time and real time depends largely on the running time of the events. This study did not prioritize real-time measurements, which would involve highly optimized implementations of the protocol.

The environment $\mathcal{Z}$ does not simulate any adversary other than delays. Thus all nodes are assumed to behave honestly.

Broadcast The network follows a full-mesh topology. The nodes can be said to use perfect point-to-point links for communication. The environment $\mathcal{Z}$ copies messages from one node to another node and never loses any message.

Node A node has the following attributes within the simulation: id, hashPower, local blocktree , local mempool and a reward balance.

Transactions A transaction has the following attributes: id, creation timestamp (*creation_time*), list of reception timestamp (*rx_time*), sender, receiver, value, size and a fee. Transactions are generated by the environment $\mathcal{Z}$ a priori and given a timestamp for reception (for each node). $\mathcal{Z}$ creates T_n transactions at every second of the simulation time, as defined per the simulation parameters. Then the transaction is propagated. Essentially, for every node $\mathcal{Z}$ calculates a reception time and stores it in the list of reception timestamps. The reception timestamp for a node u is calculated as $\textit{rx\_time}(u, tx) = \textit{creation\_time} + X[u, tx]$ with random variable (r.v.) $X[u, tx] \sim Exp(1/\textit{propagation\_delay})$. $X[u, tx]$ can be thought of as a lookup table.

Thus a node u has tx in its mempool M_u^t, if $t \geq rx\_time(u, tx)$.

Block Generation A block B has the following attributes: id, previous, timestamp, miner, list of transactions and block size. The simulation does not explicitly use hashes; instead, they rely on IDs but for all intents and purposes, the ID can be thought of as a hash h.

$\mathcal{Z}$ initially creates block-creation events for all nodes. The block-creation events are distributed according to the node's mining power with $\textit{block\_creation\_time}(u, B) = M$, $M \sim Exp(1/\textit{minig\_power}_u)$

Whenever the $\mathcal{Z}$ simulates a block-creation event at time t it will use the transaction selection strategy γ of Definition 2 to fill the set of transactions.

The block propagation is simulated by creating a block-reception event with a transmission delay. The transmission delay does not scale with block size but is a r.v. that is exponentially distributed.

Block Reception

Algorithm 1 Block reception

```
1:  function RECEIVEBLOCK(E, NODE_RECIPIENT)
2:
3:      if not V(E.Block) then return
4:
5:      B ← E.Block
6:      miner ← B.miner
7:      lastBlockHash ← sel(node_recipient)
8:      if B.h = lastBlockHash then
9:          node.blockTree = node.blockTree ∪ {B}
10:     else
11:         if miner.calculateDepth(B) > sel(miner.blocktree).depth then ▷ Block is further ahead
12:             node.update_local_blockchain(miner)
13:         else
14:             node.forkedBlocks = node.forkedBlocks ∪ {B}
15:             return
16:     node.update_mempool(B.T)
17:     node.generate_next_block()
```

Block-reception is triggered by the block-reception event. Algorithm 1 *lines 1 - 3* show how a newly received block is first checked for its validity. If and only if, every transaction $tx \in B$ in the block is valid, the block may be included in the local blocktree. In *line 7- 9* the receiving node r checks if the receiving block extends the tip of its longest chain; if so the block is added to the blocktree. If B does not extend the tip, node r may either fetch blocks it has missed or not received yet from the sender until r has caught up with the miner (*lines 11, 12*), or the block is stored as a forked block (*lines 14 - 15*). Finally, the received transactions are removed from the local mempool, and the node generates a new block event, based upon the new state of the blocktree (*lines 16, 17*).

3.3 Implementation of DAG-Nakamoto protocol

The implementation of the DAG-Nakamoto protocol includes several entities and workflows at different layers of the protocol. Most of the work presented here has been focused on the consensus layer. The network layer has been largely left untouched. The incentives layer has remained largely untouched, if one wishes to examine rational nodes, this layer would most likely need to be adjusted.

Figure 3.2 shows how the base model was extended to facilitate the implementation of the BlockDAG model. Parts highlighted in light blue have been altered noticeably and we will explain what had to be implemented and where simplifications were made.

Block Generation Block generation is largely the same as in the original protocol. A block B' as defined in 2.4.1 is an extension of the original block structure with a set of references R to abandoned blocks. This leads to differences in how the blocks are generated. A node must manage a set of abandoned blocks. In the most simple case, a node may have received a block $B'_A \in \textit{abandoned}(B')$ which the node deems abandoned and adds it to the set R. Due to the topological ordering defined in Definition 9 the block B'_A will be delivered before B'. In the more complex case, with blocks B' trying to reference multiple abandoned blocks $|\textit{abandoned}(B')| > 1$, we have to individually ensure that after referencing a block B'_{A_1} a second block B'_{A_2} still contains at least on the valid transaction. Similarly, we have to ensure that

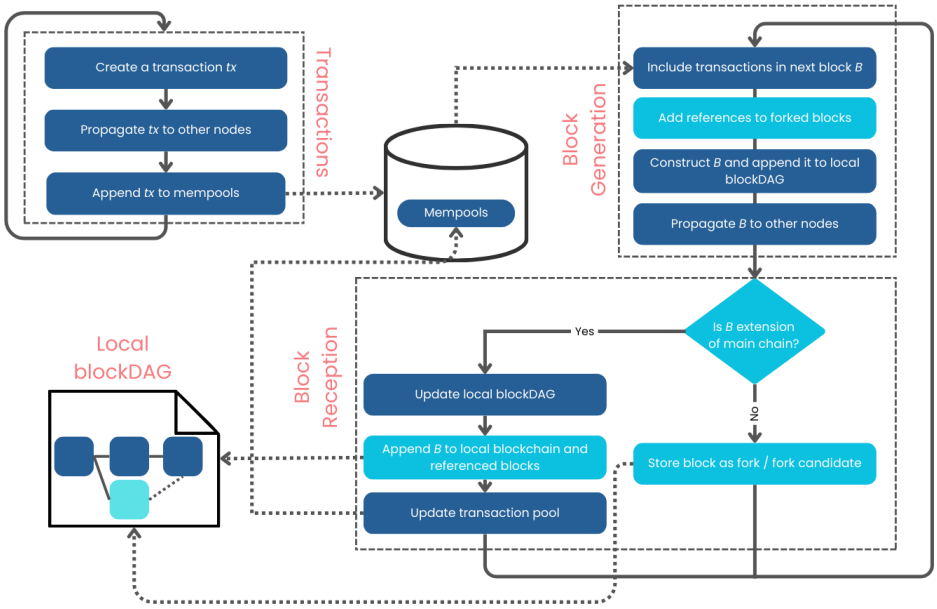

Figure 3.2: Workflow for the consensus activities implemented within the DAG-NAKAMOTO consensus model of BlockSim, with changes to the base model (based on NAKAMOTO consensus) highlighted in clear blue.

the blocks $B'_{A_1}.R$ and $B'_{A_2}.R$ do not contain the same blocks, otherwise, we cannot ensure a unique total order. If we have more than two blocks we need to check $O(|abandoned(B')|^2)$ possibilities.

Block Reception Note that checking if a block is valid has been excluded from the diagram 3.2 for the sake of simplicity but it is still a part of the protocol execution. In the Nakamoto-DAG protocol the receiving node checks if a received block B' extends the main chain. If not, the block is stored as *fork candidate / abandoned block* to be potentially referenced at block generation. If the received block B' extends the main chain, the local BlockDAG is updated. If the received block B' is ahead of the main chain, the node attempts to fetch the missing blocks from the miner or another node. Similarly, if the node is missing blocks in $reachable(B'.R)$ the node attempts to fetch them from another node. The mempool is updated by excluding all transactions that are included in the set of reachable blocks from $B'.R$, $\{B'.T\} \cup \{b'.T | b' \in reachable(B'.R)\}$.

Remark 2. *Note that when a node receives a block that extends the main chain it will stop the current PoW*[2]*. Furthermore, the protocol will accept blocks that reference blocks below the tip of the main chain. This implies that the protocol design must be aligned with the incentives that may arise by allowing nodes to mine at lower levels (reference blocks below the tip of the main chain).*

[2]In the discrete-event paradigm, this is equal to removing all future block creation events and creating new ones

3.4 Simulation parameters

Type	Parameter	Description
Blocks	creation interval $[s]$	*BLOCK_CREATION_INTERVAL* : Average time for creating a block in the protocol
	size $[mb]$	*BMAXSIZE* : Maximum block size
	delay $[s]$	*BLOCK_DELAY* : Block propagation delay parameter for exponential distribution
	reward	Reward for mining a block
Transactions	number	T_n : Number of transactions that are generated per second
	size $[mb]$	Transaction size parameter for exponential distribution
	delay $[s]$	Transaction broadcast delay parameter for exponential distribution
	fee	Transaction fee parameter for exponential distribution
Nodes	number	$\mathcal{N}$: Number of nodes
	hashPower $[\%]$	Fractional hashing power of the protocol
Simulation	SimTime $[s]$	Duration of the simulation
Additional	Creation Seed	Used to seed the exponential distribution for the block creation interval
	Plot Chain	Controls if the BlockDAG of all nodes is plotted
	Plot Similarity Matrix	Controls if the mempool similarity of all nodes is calculated and plotted
	Print progress	If set to TRUE, broadcasts are logged to console

Table 3.1: The table shows an exhaustive list of parameters that can set to run the simulation [1]. The parameters are divided into four types of entities that they affect: Transactions, Blocks, Nodes, Simulation and some additional parameters. The additional parameters were implemented, to create some visualizations for the BlockDAGs and their execution.

Table 3.1 shows an overview of all the parameters that can be tuned before running the simulation. The parameters were extended to implement plotting functionality to show the progress of the consensus protocol. The number of nodes can be configured and their respective hashing power as a fraction of all nodes hashing power. In this work, we are most interested in the parameters affecting the measurements of efficiency as outlined in Section 2.5. The parameters are divided into four groups: block, transaction, node, general simulation parameters and additional parameters. The additional parameters were implemented to generate graphs of the BlockDAGs and show the progress of the execution.

3.4.1 Parameter relationships

In this section, we show how the block rate parameter of the simulation influences other measurements. Figure 3.3 shows some of the trade-offs that are true for blockchain and BlockDAG protocols.

The throughput of both blockchain and BlockDAG protocols can be naively scaled by increasing the block rate[3]. In a PoW-based protocol, a higher block rate is achieved by tuning the difficulty parameter d. In the simulation, we can directly influence the block rate parameter.

An increased block rate will have an influence on the fork rate and the throughput. At the same time, the increased fork rate will lower the security against double-spend attacks [11]. Note that this holds for

[3]Another way to increase throughput is by increasing the maximum block size, this approach is not examined in the thesis.

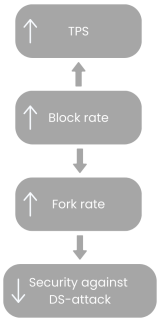

Figure 3.3: Tradeoffs in blockchain and BlockDAG models between different parameters. The white (thin) arrows denote an increase in the measurement or parameter. The grey (thick) arrow shows the direction of influence of the parameter or measurement [11]. Notably, a higher fork rate will lead to lower security against double-spend attacks (DS-attack).

both NAKAMOTO and DAG-NAKAMOTO protocols, since the security of Π' reduces to the one of Π as shown by [2].

We will show how DAG-NAKAMOTO throughput can be increased with the greedy selection strategy in comparison to the NAKAMOTO protocol. Note that the mempool similarity across all nodes of the network influences the throughput of DAG-NAKAMOTO. If all nodes have the same transactions in the mempool at time t and a fork is created, then the fork will contain **exactly** the same transactions as the block in main chain. As such the forked block will not contribute to the throughput of the protocol in DAG-NAKAMOTO.

Lemma 2. *A higher block rate leads to a higher fork rate*

Proof. This follows trivially from how forks are defined. With a higher probability of finding a solution to PoW, the probability of two nodes finding a solution to PoW at the same time increases. □

Lemma 3. *Higher fork rate can lead to higher TPS*

Proof. Consider an execution of NAKAMOTO ε and DAG-NAKAMOTO ε' where

$$\mathit{forkRate}(\varepsilon) = \mathit{forkRate}(\varepsilon') = 0$$

In this case the throughput closure reduces to the original protocol thus the expected throughput is the same for both protocols [2].

Conversely if the $\mathit{forkRate}(\varepsilon) = \mathit{forkRate}(\varepsilon') > 0$ the throughput closure can include more blocks than the original protocol. The throughput is strictly greater or equal than the original protocol, provided that a forked block in Π' always contains at least one valid transaction [2]. □

Lemma 4. *High mempool similarity leads to similar transaction selection*

Proof. Consider an execution of a BlockDAG protocol with two nodes u, v at time t. If

$$\mathit{mempoolSimilarity}(u, v) = 1 \implies J(\mathit{txSelection}(M_u), \mathit{txSelection}(M_v)) = 1$$

Similarly, it holds that

$$\mathit{mempoolSimilarity}(u, v) = 0 \implies J(\mathit{txSelection}(M_u), \mathit{txSelection}(M_v)) = 0$$

Higher mempool similarity increases the probability of selecting similar transactions. Note that even if two nodes have different mempools, they may still choose the same transactions if the fee-to-size ratio of a particular transaction exceeds the fee-to-size ratios of other transactions. Miners are assumed to behave rationally, meaning they try to maximize the reward associated with the selection of transactions. In the simulation, we assume an exponential distribution of the transaction fees and an independent exponential distribution of the size. □

Remark 3. *Selecting different transactions for B and abandoned(B) leads to higher TPS .*

This follows directly from the definition of the conflict inclusion rate. If abandoned(B) contains the same transactions as B, the conflict inclusion rate is 100%. The DAG-NAKAMOTO *protocol cannot include any additional transactions compared to the* NAKAMOTO *protocol.*

Theorem 2. *Lower mempool similarity leads to higher TPS in* DAG-NAKAMOTO *with the greedy selection strategy*

Proof. This follows directly from Lemma 4 and Remark 3. Lower mempool similarity will lead to a difference in transaction selection for blocks B and *abandoned*(B) and thus a higher reference inclusion rate in the DAG-NAKAMOTO protocol. □

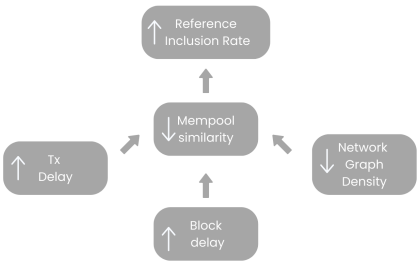

Figure 3.4: Parameter influences in DAG-NAKAMOTO protocol between parameters with the greedy selection strategy. The white (thin) arrows denote an increase in the measurement or parameter. The grey (thick) arrow shows how the direction of influence of the parameter / measurement.

Figure 3.4 shows the influences parameters have, on the mempool similarity. Increasing transmission delays[4] on blocks and transactions will lead to lower mempool similarity throughout the execution of the protocol. If there was no transmission delay, all nodes would share the same mempool in a fully connected network. Since every node would have access to the same transactions and blocks at the same time. Conversely, increasing transmission delays induces heterogeneity in the mempools. Lower mempool similarity leads to a higher reference inclusion rate and thus also to higher throughput[5].

The network layer in the simulation is modeled as a full mesh network, causing all the nodes to receive the same transactions and blocks eventually. Lowering the network graph density will decrease the mempool similarity. Lowering the network graph density can be thought of as increasing the transaction delay since messages need to traverse more nodes and thus a higher delay on messages is incurred.

The paper Inclusive Block Chain Protocols [8] examines an inclusive protocol, similar to the throughput closure based on NAKAMOTO consensus. The paper leads a game-theoretic argument in which rational nodes aim to improve their rewards by adapting a different transaction selection strategy different than the greedy one γ. The paper defines a *myopic strategy* which focuses on selecting the transactions for a single block, regardless of how that selection would influence the selection of future blocks. Furthermore, the

[4]Delay parameter to the random exponential distribution.

[5]Increasing block delay leads to a higher reference inclusion rate but not necessarily to higher throughput.

paper compares it to the optimal selection strategy which would be reached by cooperation of nodes, in which every node selects unique transactions for every block (thus actively avoiding collisions by being greedy). In conclusion, they show that the myopic strategy achieves higher throughput than the greedy selection in the non-inclusive protocol and relatively close to the optimal throughput.

4 Evaluation

To evaluate the implemented protocol a combination of parameters is selected from the parameter space and used to run a simulation that generates 100 blocks. The simulation is run with several constants and derived parameters. The number of nodes is constant[1] and set to $|\mathcal{N}| = 4$. The share of mining power per node is uniformly distributed among nodes. The parameter T_n controls the number of transactions per second (generated by participants in the protocol), it is set, such that nodes have no empty mempools. The maximum block size parameter *BMAXSIZE*, is set to 1 *MB* and the average transaction size to $546B$, which yields on average ≈ 1800 transactions per block. Detailed descriptions of all parameters are listed in Table 3.1.

4.1 Parameter space

The parameter space consists of three parameters:

Parameter	Description	Values
TX_DELAY	Transaction propagation delay distribution	$\{0.5, 5, 10, 15, 30, 60\}$
BLOCK_DELAY	Block propagation delay distribution	$\{0.3, 4, 10, 20, 30\}$
BLOCK_CREATION_INTERVAL	Block creation interval distribution[2]	$\{0.5, 1, 2, 50, 600\}$

Table 4.1: The parameter space with a description for each parameter. In total there are 150 combinations with the given values.

For every parameter from the parameter space as shown in Table 4.1, a set of discrete values has been selected. In total this yields *TX_DELAY* $\cdot$ *BLOCK_DELAY* $\cdot$ *BLOCK_CREATION_INTERVAL* $= 5 \cdot 5 \cdot 6 = 150$ combinations. Every parameter p from the parameter space, is used in the simulation as input $\lambda_p = \frac{1}{p}$ to generate an exponentially distributed r.v. $X_p \sim Exp(\lambda_p)$.

[1] The number of nodes was set sufficiently large to tolerate a potential adversary and not significantly increase the simulation's running time.

It is computationally infeasible to run the simulation with a high number of nodes $|\mathcal{N}|$. Take for instance, the real Bitcoin network, it spans ≈ 18000 nodes[3]. With one simulation task taking ≈ 10 minutes this would lead to an execution time in the order of 125 days.

4.1.1 Choice of parameters

The simulation script allows to modify a plethora of variables such as the number of nodes, transaction and block delay, transaction fees, sizes, and more. In this case, we are interested primarily in variables affecting the TPS, reference or conflict inclusion rate and the inclusion times outlined in Section 3.4.1.

The parameter T_n was set, such that the mempools are never empty and every block can hold an amount of transaction sufficiently close to the maximum. We are interested improving throughput, which does not make sense if the protocol does not need to process a large number of transactions.

[3] `https://bitnodes.io/`

4.2 Simulation pipeline

Figure 4.1: Simulation pipeline. (1) The execution script is written in Python, it can be run with a task ID. The task ID is used to select one of the 150 parameter permutations. Several nodes and the management of the BlockDAGs are simulated. (2) To speed up the execution, 150 parallel tasks run on the University HPC. (3) The results are written to disk on the HPC and are then transferred to (4) Google Colab, which spins up an ephemeral VM with persistent code that can be run. Google Colab is used to plot the results.

The simulation pipeline shown in Figure 4.1 consists of roughly four components (1) execution script (2) parallel execution on UBELIX, the University of Bern HPC [4] (3) collecting the results in a suitable format and (4) plotting of the results in Google Colaboratory (Colab)[5]. Google Colab allows running Jupyter Notebooks [7] (Python) on ephemeral VMs.

```
1  #!/bin/bash
2  #SBATCH --job-name="BlockDAG Simulation"
3  #SBATCH --time=01:00:00
4  #SBATCH --mem-per-cpu=2G
5  #SBATCH --cpus-per-task=4
6  #SBATCH --array=1-150
7
8  module load Python/3.9.5-GCCcore-10.3.0
9  module load Workspace_Home
10
11 cd /storage/homefs/rr15t008/thesis/BlockSim/src/
12 pip install -r requirements.txt
13
14 cd /storage/homefs/rr15t008/thesis/BlockSim/src/
15
16 srun python3 -m Main ${SLURM_ARRAY_TASK_ID}
```

Listing 4.1: Slurm Job Submission Script

The execution script is a bash script that launches 150 tasks in parallel on UBELIX. Listing 4.1 shows the submission script. For each simulation task, 2 GB RAM and 4 CPUs are requested from the cluster. The entire cluster consists of around $\approx$ 300 Nodes with $\approx$ 13000 Cores and $\approx$ 35 TB of RAM.
The Python module is loaded and the folder "workspace-home", which is part of the dispatch server is mapped into to the computation nodes (using General Parallel File System, or GPFS by IBM). The necessary dependencies are installed (*Line 12*) and the actual job is submitted with one of the 150 combinations to an execution node (using the *SLUM_ARRAY_TASK_ID*). One simulation task takes around 10 minutes

[4] http://www.id.unibe.ch/hpc
[5] https://colab.research.google.com/

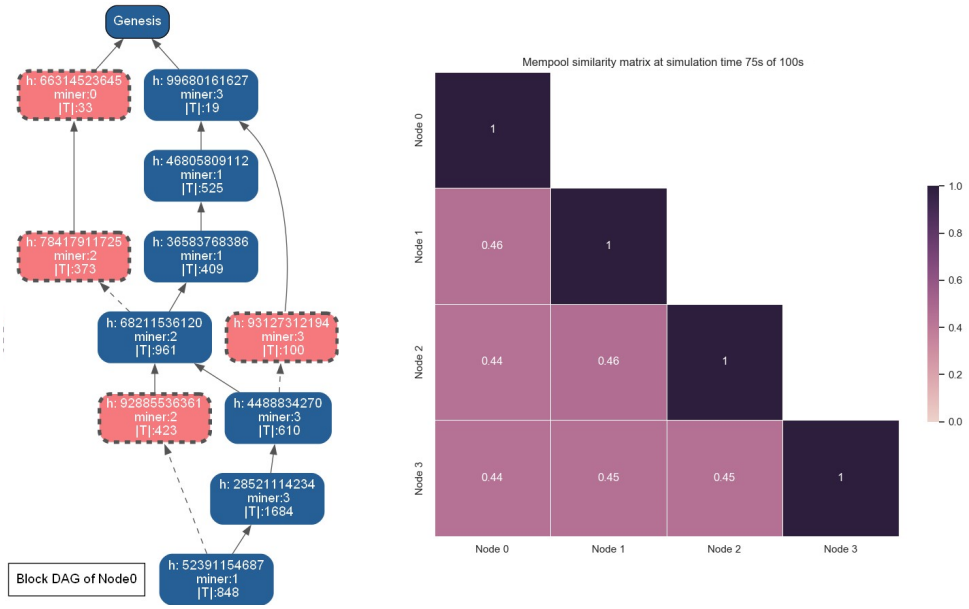

Figure 4.2: Outputs of a simulation during the execution of the simulation. The left side shows the BlockDAG of node 0 at $t = 100s$ in the simulation. On the right side, the mempool similarity matrix graph at $t = 75s$. The graph displays a similarity matrix between all the nodes' mempools. It indicates that approximately $\approx 45\%$ of all transactions in the mempools are shared among all pair of nodes.

to run, the duration varies strongly with the number of nodes simulated. Similarly, simulating a low block creation interval leads to a longer duration of simulation tasks.

The execution nodes run the tasks and the simulation writes the result to disk, thanks to GPFS we can easily collect the output files from the "workspace-home"-directory. The output file is a JSON formatted file containing the input parameters and the measured results. To generate the evaluation plots the files are uploaded to an ephemeral VM on Google Colab and the plots are generated using several libraries (Pandas, Plotly and Numpy). Google Colab allows you to run Jupyter Notebooks without any additional setup.

4.3 Simulation outputs

The simulation generates varying outputs, depending on the **extended parameters** outlined in Table 3.1. The simulation will output following artifacts: mempool similarity matrix plot, a BlockDAG plot of every node and the measured results in JSON format. Figure 4.2 displays the graph for the BlockDAG of a node and the mempool similarity matrix plot at an arbitrary point during the simulation. The JSON file with the results is displayed in Listing 4.2.


```
{
    "params":{
        "const_node_count":4,
        "const_bsize":1.0,
        "const_sim_time":100,
        "block_creation_interval":10,
        "block_delay":10,
        "tx_delay":15.1,
        "tx_per_second":60,
        "model":4
    },
    "results":{
        [...]
        "tps_sim":51.85,
        "optimal_tps_sim":59.52,
        "bitcoin_tps_sim":50.56,
        [...]
        "conflict_inclusion_rate_avg":0.85,
        "reference_inclusion_rate_avg":0.15,
        "conflict_time_to_inclusion_avg":0.15,
        "reference_time_to_inclusion_avg":2.047,
        "fork_rate":0.36
    }
}
```


Listing 4.2: JSON-formatted output for the simulation execution includes measured results and the parameters used for the simulation.It provides information such as the simulated TPS ("tps_sim"), the optimal TPS ("optimal_tps_sim"), and the TPS Bitcoin would achieve ("bitcoin_tps_sim"). The conflict inclusion rate ("conflict_inclusion_rate") measures how many transactions in forked blocks were already included in the main chain. Reference time to inclusion measures the latency for transactions in forked blocks included by reference. The fork rate indicates how many blocks have been forked.

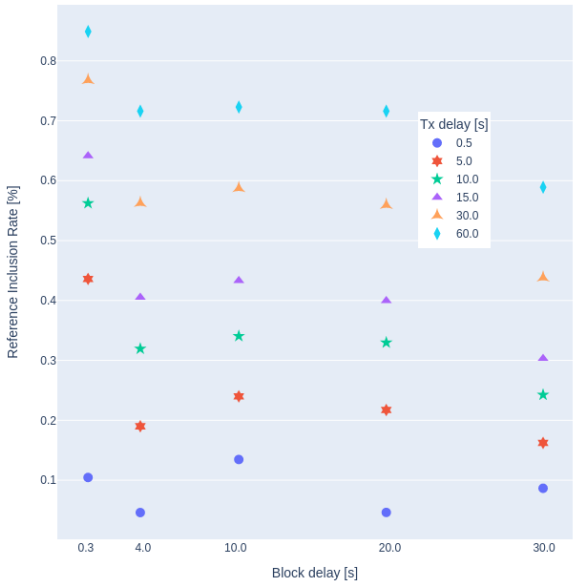

Figure 4.3: The graph represents the results of 30 simulations, each involving a different combination of block delays and transaction delays. Every point in the graph corresponds to the measured reference inclusion rate with different block delays and transaction delays. The measured reference inclusion rate is averaged over all forked blocks. Higher transaction delay leads to a higher reference inclusion rate. Around 5% - 90% of transactions in forked blocks can be included through a reference. Transactions that cannot be included by reference, have been included in the main chain by another block processed earlier.

4.4 Results

Reference inclusion rate Figure 4.3 shows several executions with different *BLOCK_DELAY* and *TX_DELAY*, the plot shows executions with *BLOCK_CREATION_INTERVAL* $= 0.5$. We can see that higher transaction delays lead to higher reference inclusion rate. A higher block delay seems to slightly lower the reference inclusion rate. The conflict inclusion rate is the inverse of the reference inclusion rate because we do not consider other forms of transaction invalidity as outlined in Section 2.5. Thus $\mathit{conflictInclusionRate} = 1 - \mathit{referenceInclusionRate}$.

Throughput Figure 4.4 shows the TPS (of DAG-NAKAMOTO protocol) with a block creation interval of NAKAMOTO protocol. We can see that the achieved TPS roughly are consistent with the real measured TPS of NAKAMOTO protocol at around 2.5 to 3[tx/s][6]. We attribute the variance in simulated TPS to the randomness of the simulation, there seems to be no direct correlation between TPS and the higher block delay parameter.

Figure 4.5 shows the TPS of DAG-NAKAMOTO with a *BLOCK_CREATION_INTERVAL* $= 2$. The graph shows that with constant block delay and increasing transaction delay the TPS increases. Higher block delay leads to a decrease in TPS, note that we have shown that higher block delay leads to a higher reference inclusion rate. Since blocks are created in a small interval, the lower TPS can be explained by interrupted proof-of-works as mentioned in **Remark 2**.

[6] `https://www.blockchain.com/de/explorer/charts/transactions-per-second`

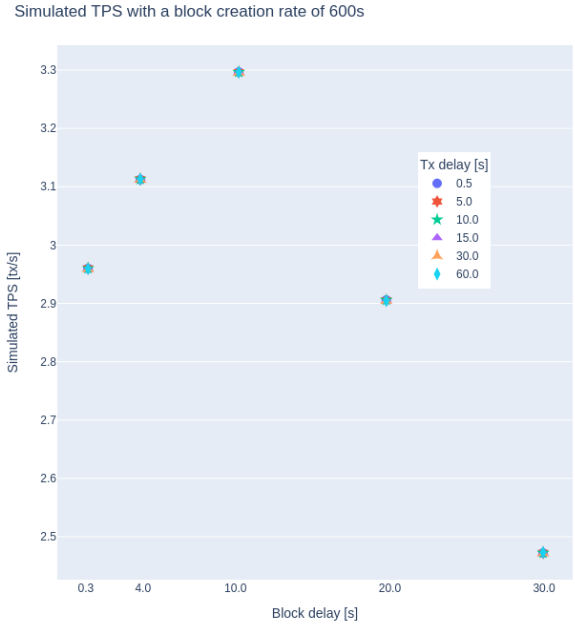

Figure 4.4: The graph represents the result of 30 simulations, each involving a different combination of block and transaction delays. The parameter *BLOCK_CREATION_INTERVAL* $= 600$ is constant across data points. The transaction delay does not influence the TPS because the fork rate is close to 0. We can see that the simulated TPS are in the range of 2.5 to 3 [tx/s]. This roughly corresponds to the TPS measured in Bitcoin.

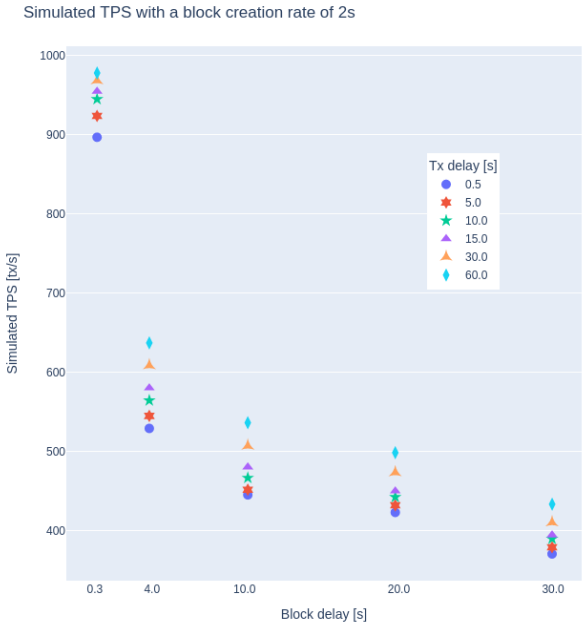

Figure 4.5: The graph represents the simulated TPS across different block and transaction delays. The parameter *BLOCK_CREATION_INTERVAL* $= 2$ is constant across all simulation results shown in the graph. We can see that with higher block delay we achieve lower TPS. On the other hand with constant block delay and higher transaction delay, we achieve higher TPS.

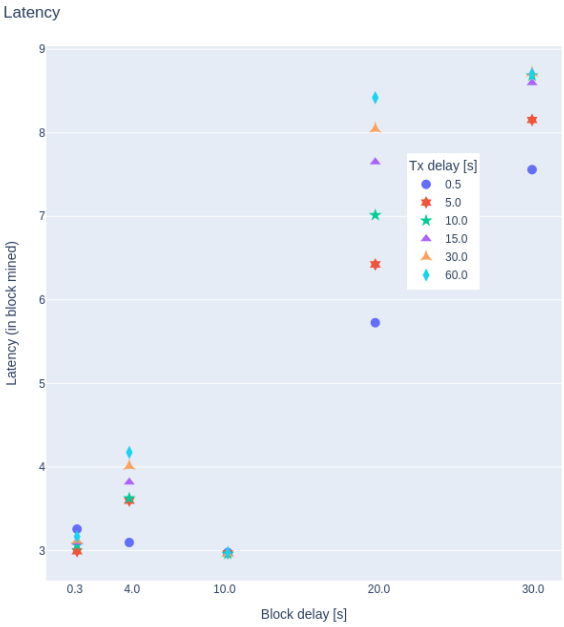

Figure 4.6: Average latency of transactions from forked blocks in simulations with different block and transaction delay parameters. The *BLOCK_CREATION_INTERVAL* $= 2$ is fixed. The latency is measured in blocks mined in the main chain since the initial transaction propagation. Note that confirmation depth k is subtracted from the latency in this graph. Transactions from the main chain would thus have an expected latency of around 0.

Latency The paper [2] shows that the latency of the throughput closure is smaller than or equal to the latency of the original protocol. Figure 4.6 shows the average latency of transactions in **forked blocks** that have been included through a reference in the main chain. Note that we do not account for the confirmation depth k. Thus, any transaction in a non-forked block would have an expected latency of 0. We can see with very low block and transaction delay, the latency is ≈ 3 blocks[7]. Note that the confirmation depth k has to be adjusted, when more blocks are created per second (higher α) to guarantee security.

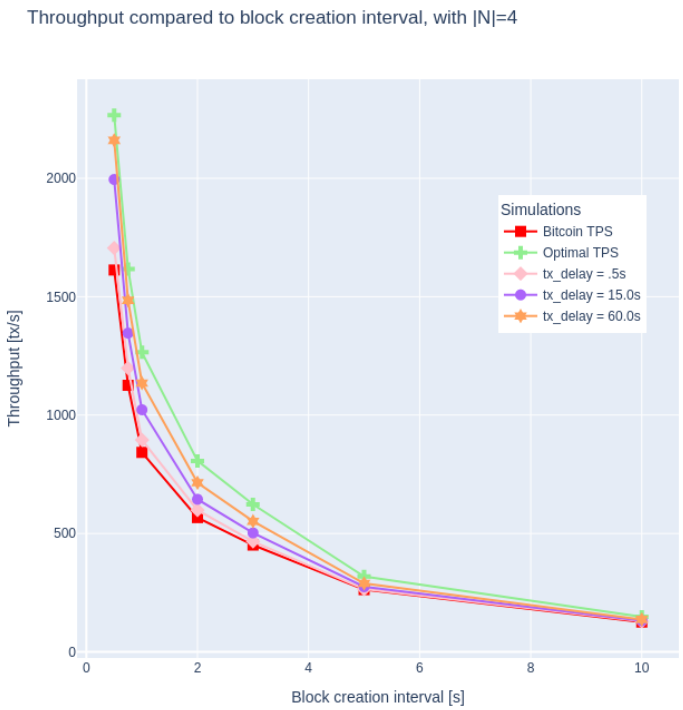

Figure 4.7: Comparison of throughput for different block creation intervals and transaction delays. The red (square) points show the lower bound on TPS, this is the number of transactions that can be delivered per second if no transaction in a forked block is included. The green (plus) points show the upper bound on TPS, this is the amount of transactions that can be delivered if every transaction in a fork can be included. To ensure a consistent fork rate, the random values generated from an exponential distribution for block creation are initialized with an identical seed throughout all simulations. We can see that by varying *TX_DELAY* we can achieve higher throughput and approach the upper bound on TPS. With very small transaction delay and a high block rate, the nodes tend to include the same transactions and thus the gain in TPS by including referenced blocks is small.

[7]The delay of 3 blocks can be explained, by implementation details: When two blocks A, B are created concurrently and both extend the same block G. A node receiving A (the case where B is received earlier is analogous), immediately starts a new PoW for block C referencing block A. Thus the D block after C is the earliest block that can reference B. As such, 3 blocks A, B, C have been added to the main chain before B is delivered. A higher block delay parameter leads to more blocks mined between a fork and a block referencing one of the forked blocks.

Upper- and lower-bound on TPS Figure 4.7 shows different executions of DAG-NAKAMOTO protocol implementation compared to the upper- and lower bounds on TPS as seen in **Remark 1**. By adjusting the transaction delay, we can decrease the mempool similarity, thus increasing the reference inclusion rate and we can achieve higher throughput.

Parameter considerations The *BLOCK_CREATION_INTERVAL* should be chosen to prevent *unfair mining*. We define *unfair mining* as a single node adding disproportionately many blocks to the main chain compared to its mining power. Thus, it must be ensured that:

$$\frac{\mathit{BLOCK\_CREATION\_INTERVAL}}{|\mathcal{N}|} < \mathit{BLOCK\_DELAY}$$

Assuming a uniform distribution of mining power across all nodes.

5
Conclusion and future work

This master thesis focused on assessing the inefficiencies of existing protocols due to the longest-chain rule and implemented the conversion of a blockchain protocol to a BlockDAG protocol. We have shown that the longest-chain rule introduces a constraint on the maximal throughput by forcing all nodes to extend a single block to ensure safety. Thus introducing a trade-off in security versus throughput.

We have shown that the conversion from NAKAMOTO to DAG-NAKAMOTO protocol functions and with the right parameters can approach the optimal upper bound on TPS.

In summary, this work provides an overview of the essential background and notation to convert a blockchain protocol into a BlockDAG protocol, based on the construction of the throughput closure [2]. Several attempts were made at evaluating the efficiency of the throughput closure of existing protocols. Various approaches were documented, to record transactions and blocks of running protocols. Finally, the BlockDAG protocol was implemented using a simulation framework called BlockSim [1]. The BlockSim blockchain model was transformed into a BlockDAG protocol. The implementation is written in Python and can easily be parameterized and produce various plots and statistics.

Additionally, we showed some impacts of implementing the throughput closure on several aspects of a protocol: network layer, reward layer, transaction validity and block validity. Various metrics were defined to assess the performance of the throughput closure compared to the original protocol. We showed how parameters influence each other. We evaluated the implementation's efficiency with the help of the defined metrics. The evaluation of the implementation involved a series of simulations across a range of parameters using a simulation pipeline.

In conclusion, BlockDAG protocols have the potential to achieve higher TPS than their blockchain counterparts. BlockDAG protocols constructed with the throughput closure have the same safety as the original protocols. However, it is crucial to approach adjustments to the block rate with careful consideration, ensuring alignment with the confirmation depth k, to guarantee safety properties of the protocol.

5.1 Future work

As we conclude the current work, we want to underline some compelling avenues for future research and development. The potential enhancements to DAG protocols can be approached from various angles. We briefly introduce three intriguing directions for future works.

In-depth analysis of the message complexity and contribution to TPS As we have seen, implementing the throughput closure will affect the message complexity of the protocol because invalid blocks and transactions will also be propagated. Exploring messages regarding their contribution to the BlockDAG, and consequently to the TPS, presents an interesting area of research.

Optimizing the transaction selection strategy A simple protocol change to approach the upper bound on TPS, would involve optimizing the transaction selection strategy γ. This thesis analyses how delays and other parameters affect the TPS.

Optimizing γ could be achieved by implementing additional messaging to signalize between nodes, which transactions are being included in a block by a POW. For instance, a bloom filter could be used by a node, to signalize to other nodes, the ongoing inclusion of a transaction. False positives would lead to transactions falsely not being selected and thus to a slightly higher latency (this rate can also be reduced by using a filter with more bits). If a transaction is not in the bloom filter, we can safely start validating it. This would ensure that all blocks contain different transactions.

Analyze the security-performance trade-off The simulation does not study more closely guarantees against colluding Byzantine nodes. In the realm of research, it would be interesting to analyze the safety of the throughput closure with a high probability of finding blocks (high α) and how the confirmation depth k has to be set to achieve safety with high TPS.

Optimized implementation The simulation leverages Python, enabling a rapid implementation. However, for a more precise evaluation of the protocol's performance, it is important to implement the protocol in a lower-level language and leveraging more sophisticated data structures. This would more accurately mirror real-world conditions.

Bibliography

[1] Maher Alharby and Aad van Moorsel. Blocksim: A simulation framework for blockchain systems. *SIGMETRICS Perform. Evaluation Rev.*, 46(3):135–138, 2018.

[2] Ignacio Amores-Sesar and Christian Cachin. We will DAG you. *CoRR*, abs/2311.03092, 2023.

[3] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 585–602. ACM, 2019.

[4] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*, pages 1–10. IEEE, 2013.

[5] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *IACR Cryptol. ePrint Arch.*, page 1119, 2018.

[6] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[7] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing, Göttingen, Germany, June 7-9, 2016*, pages 87–90. IOS Press, 2016.

[8] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 528–547. Springer, 2015.

[9] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. [Online; accessed 11. October 2023].

[10] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. PHANTOM GHOSTDAG: a scalable generalization of nakamoto consensus: September 2, 2021. In Foteini Baldimtsi and Tim Roughgarden,

editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 57–70. ACM, 2021.

[11] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2015.

[12] Wikipedia. Directed acyclic graph — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Directed_acyclic_graph`, 2023. [Online; accessed 11. October 2023].

[13] Wikipedia. Jaccard index — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Jaccard_index`, 2023. [Online; accessed 11. October 2023].

[14] Wikipedia. Topological sorting — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Topological_sorting`, 2023. [Online; accessed 11. October 2023].

Declaration of consent

on the basis of Article 30 of the RSL Phil.-nat. 18

Name/First Name: Rao, Renato Claudio

Registration Number: 15-723-059

Study program: Computer Science

Bachelor ☐ Master ☑ Dissertation ☐

Title of the thesis: Blockchain and BlockDAG protocols

Supervisor: Prof. Dr. Christian Cachin

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of 5 September, 1996 is authorized to revoke the title awarded on the basis of this thesis.

For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with future theses submitted by others.

Luterbach, December 8, 2023

Place/Date

Signature