Selective Blockchain Transaction Pruning and State Derivability

Emanuel Palm, Olov Schelén, Ulf Bodin
Luleå University of Technology,
Luleå, Sweden
E-mail: $\{firstname.lastname\}$@ltu.se

***Abstract*—Distributed ledger technologies, such as blockchain systems, have in recent years emerged as promising platforms for machine-to-machine commerce and other forms of multi-stakeholder applications. However, despite the potential demonstrated by projects such as Bitcoin, Ethereum, and Hyperledger Fabric, the disk space typically required to host a copy of a ledger may be prohibitively large for many categories of devices. In this paper, we introduce an approach for reducing ledger size in blockchain systems, based on arbitrary pruning predicate functions, allowing each network participant to independently select and remove any already applied transactions. We also show that if only pruning certain ledger transactions, the ability to derive an unmodified state data structure from the remaining transactions is maintained. The approach is validated through a supply chain use case utilizing a modified version of Hyperledger Fabric, in which ledger size is reduced by about 84.49% via selective transaction pruning.**

I. Introduction

The release of Bitcoin at the end of 2008 [1] marked the beginning of a major technological trend, which has grown to encompass a plethora of derivatives and alternatives [2] [3]. These systems, commonly referred to as *Distributed Ledger Technologies* (DLTs), remove the need for using trusted middle-men by instead relying on networks of voting computers, sharing immutable histories of transactions that each participant can verify to be correct. Bitcoin famously hosts a distributed electronic cash platform where there is no central authority [1], and many other use cases have since been proposed. These include political election systems [4], self-driving cars buying their own fuel [2], and *smart factory* [5] machines autonomously buying and selling material and goods within and between factory plants. Use cases typically have in common that they require a shared notion of identity, ownership, and transfer of ownership within a network of machines controlled by different stakeholders, each with its own incentive to maintain the system.

Although distributed ledgers could be a key enabler for emergent technologies such as autonomous machine-to-machine commerce, the storage space typically needed to host a ledger might be prohibitively large for many categories of devices. At the time of writing, the Bitcoin blockchain ledger required about 134 gigabytes of memory, growing steadily at a pace of about 52 gigabytes per year [6], while the ledger of Ethereum [7], another popular blockchain system, required almost 395 gigabytes of memory [8]. Both of these systems are *permissionless* (i.e. open to public participation) meaning that any accepted activity by any participant will likely lead to an increase of ledger size. On the other hand, *permissioned* systems, such as R3 Corda [9], Quorum [10], or Hyperledger Fabric [11], assume that ledgers only be replicated within closed groups of known participants. This may lead to ledgers being smaller, but could also mean that many ledgers are maintained at the same time. In [12], a system is presented where each network stakeholder can maintain private blockchains with any subsets of other network participants in order to protect sensitive interactions.

The problem of ever-growing distributed ledgers is already well-known. Bitcoin and Ethereum allow the use of light clients [1] [7] that only carry limited ledger and state subsets, but are unable to participate in the consensus process. At the cost of more disk usage, Bitcoin Core [13] and related efforts [14] [15] [16] retain the ability to validate transactions by first constructing a complete state data structure, and then pruning all but the n last ledger blocks. These approaches are, however, lacking in that they (1) do not facilitate fine-grained control of what transactions to keep, and (2) do not allow useful data structures to be derived from partially pruned blocks.

In this paper, a pruning strategy applied to Hyperledger Fabric [17] version 0.6 is presented. The strategy is similar to the disk space reclaiming procedure proposed in the original Bitcoin paper [1], with the significant difference that transactions of interest can be left unpruned, and the ability to rebuild a valid state data structure can be maintained if certain conditions apply. Pruning is regulated through the use of arbitrary predicate functions, which can be specified such that only transactions cancelling each other out, or are fully superseded by later trancations, are deleted. For example, certain money transfers may sum up to zero, or, an ownership statement may fully replace one or more previous such. We show that if only removing certain transactions with the mentioned properties, the state data structures derivable from any blocks of interest can be made to remain unaltered.

The contributions of significance presented in this paper are: (1) A strategy for pruning blockchains relying on arbitrary predicate functions for transaction selection. (2) The identifying and outlining of significant implications of pruning, including how it affects expected memory growth and its impact on being able to build unmodified state data structures from pruned blocks. And finally, (3) a supply chain use case facilitated by a modified version of Hyperledger Fabric [17] version 0.6 that serves to validate the presented approach and to exemplify its impact on ledger storage requirements.

978-1-5386-7204-4/18/$31.00 ©2018 IEEE
DOI 10.1109/CVCBT.2018.00009

The paper is organized as follows: Section II describes related efforts in more detail. Section III outlines a general strategy for selective blockchain transaction pruning, as well as methods for predicting blockchain growth. Section IV presents a modification to Hyperledger Fabric making it able to prune its ledger, together with a use case and benchmark results. Section V discusses related topics and suggestions for future research. Finally, Section VI concludes the paper.

II. Related Work

In the original Bitcoin white paper [1], two approaches for reducing ledger size were presented, namely *Reclaiming Disk Space* (RDS) and *Simplified Payment Verification* (SPV). Even though other approaches have been developed, such as for Bitcoin Core [13] or Cryptonite [14], all such known to the authors are generally similar to those originally suggested. For this reason, the names of the original approaches are used here to represent the current state-of-the-art of ledger size reduction.

A. Reclaiming Disk Space

The key idea of the RDS approach is to remove some transactions, from the oldest and onward, after applying them to a state data structure [1]. To avoid losing the ability to verify that given blocks or transactions are part of the pruned past, block hashes and Merkle Tree roots [18] may be saved.

Systems relying on probabilistic consensus [19] are required to allow already accepted blocks of transactions to be replaced if a chain of such with sufficient *weight* is presented [1]. It is, therefore, needful to refrain from pruning the n most recent blocks. n is chosen to represent a *weight* (e.g. proof-of-work [1]) that is large enough to avoid the practical possibility for an $m > n$ block reorganization to occur, where m is the number of known main blocks that are superseded by a reorganization. An $m > n$ reorganization leads to a given network participant no longer being able to assume its state to be relevant, meaning the state must be acquired again from its network.

While this approach leaves room for keeping transactions of interest, no strategy for choosing such is described in any work known to the authors. Our solution, on the other hand, gives fine-grained control over what transactions to keep, which we also show can be used to maintain particular system properties, such as being able to rebuild unmodified state data structures from partially pruned blocks.

B. Simplified Payment Verification

Using SPV means that a network participant gives up the ability to verify the validity of new transactions by itself, and instead delegates that responsibility to other nodes of its network [1]. Giving this ability up means that only the portions of a blockchain that are of interest to the owner of a given SPV node need to be saved.

In Bitcoin Core [13], SPV nodes download all block headers, but only transactions of interest. As block headers contain Merkle Tree [18] root hashes, any retrieved transaction can be proved to belong to any of the known block headers. Transactions and blocks are requested by providing a number of regular nodes with filters [20], which are used to decide what transactions to relay. Given that a majority of the connected regular nodes convey all desired transactions, a valid partial state can be built from them. Transactions of interest can be selected up to the granularity of instances of addresses, keys or script hashes within individual transactions.

Requesting other nodes to send certain transactions is, however, not the same as leaving out transactions while pruning. Also, the procedure facilitated by Bitcoin Core does not allow filters to consider any particular state of the data structure built by applying all transactions, only the transactions themselves. Our solution allows arbitrary information to be considered while determining if an individual transaction is to be pruned, such as derived states, blocks, significant timestamps, or any other data of interest. Even though it could require more storage space, our solution does not require giving up the ability to participate in the consensus process.

C. Other Approaches

Not all distributed ledger systems store their transactions in blocks, or even in a sequential history. Systems such as Swirlds [21], IOTA [22], or R3 Corda [9] use graph-like ledgers rather than chains of blocks, which could yield significantly different pruning implications. Nodes in R3 Corda, for example, can allegedly prune transactions of liquid asset transfers (e.g. money transfers) by requesting an asset issuer to re-issue a given asset, effectively aggregating relevant previous transactions into a single new transaction. Whether or not the pruning strategy presented in this paper could be applied to these graph-like ledgers is not considered, but is left as an open topic for future research.

III. Selective Transaction Pruning

Selective transaction pruning is to be understood as the practice of blockchain network participants independently removing transactions that neither contribute with important system properties or are of particular interest. An important system property could be protection against not having any main blocks after a reorganization, while interest depends on the information needs of a given network participant. In a supply chain scenario, transactions related to lost, damaged or late goods could be particularly interesting to shipping companies, terminal stations, etc, as they might be needed when negotiating with partners or insurance agencies. Computers on delivery trucks or container ships may only find it relevant to carry transactions related to transported goods, while statisticians may want to refrain from pruning anything from their servers to allow for future data analysis.

In order to clarify what general properties are maintained by a blockchain, and how these are changed as transactions are pruned, this section begins with a general anatomy of the blockchain data structure. It is followed by descriptions of how significant blockchain properties may be maintained, the definition of a selective transaction pruning algorithm, and a description of how the presented algorithm can be used to only remove transactions that cancel each other out or have been superseded. Finally, a method for predicting blockchain memory requirements is presented.

TABLE I: Significant parts of a general pruned blockchain.

Name	Formal Name	Invariant	Description
Indicators			
Block Height	h	$1 \leq h$	Authoritative blocks part of a blockchain. $h = 1$ implies the existence of only a genesis block.
Block Position	i	$1 \leq i \leq h$	The index of the ith block. The genesis block is represented by $i = 1$.
Transaction Position	j	$1 \leq j \leq u_i$	The jth transaction of the ith transactions set, containing u_i transactions.
Reorganization Height	m	$1 \leq m \leq h$	Number of main blocks superseded by a reorganization.
Guard Height	n	$0 \leq n \leq h$	Number of consequtive blocks, beginning with the most recent, used to protect against reorganizations.
Primitives			
Block	b_i		The ith block.
Header	H_i		Header of ith block, containing block predecessor hash, checksum of block transactions, etc.
Transaction	$t_{i,j}$		The jth event description out of u_i belonging to the ith block.
Transaction Set	$t_{i,*}$		All transactions belonging to the ith block.
State	s_i		A data structure built by applying every $t_{i,j} \in t_{i,*} \in b_i \in \{b_1, \ldots, b_i\}$.
Significant Blocks			
Genesis Block	b_1		The initial block.
Current Block	b_h		The most recent block of a currently authoritative chain of blocks.
Main Blocks	b_*		All blocks, pruned or not, part of a currently authoritative chain of blocks.
Guard Blocks	$b_{*,guard}$		n consequtive blocks, beginning with the most recent, used to protect against reorganizations.
Free Blocks	$b_{*,free}$		$h - n$ blocks thay may or may not contain pruned transactions.
Significant States			
Current State	s_h		The hth state, representing the application of all blockchain transactions.
Derivable State	s'_i		Underived s_i that could be constructed from only local data.
Retrievable State	s''_i		Underived s_i that might be constructible from network peer data.

A. Blockchain Anatomy

Primitives: A blockchain is an ordered sequence of *transactions* grouped into *blocks*. Each transaction describes a change to a *state*, which could be any kind of data structure, such as a table of monetary accounts or a key/value store. Each block includes a *header*, which contains enough information to identify any preceding blocks and to determine if these blocks are unmodified. Concretely, the header could include the hash of the preceding block and a hash of the transactions included in the block itself (e.g., [1] [7] [14]).

Significant Blocks: Every blockchain contain two generally significant blocks, the *genesis block* and the *current block*. The genesis block is the first block in its chain, and does sometimes contain special information that dictates how a blockchain can be used. For example, Ethereum uses the genesis block to set an initial mining difficulty and can use it to preallocate funds to given accounts [7]. The current block is the most recent block in its chain, and is typically special since it must have a derived state, or the given node owning the block will not be able to participate in the process of validating new transactions. If a blockchain is part of a system relying on probabilistic consensus [19], such as Bitcoin [1] or Ethereum [7], room must be given for block reorganizations. The currently authoritative chain of blocks is typically referred to as *main blocks*. If a system with probabilistic consensus supports transaction pruning, the n most recent blocks may serve as *guard blocks*, meaning they act as protecting against $m > n$ reorganizations by not containing pruned transactions, where m is the number of main blocks superseded by the reorganization. Any block not being a guard is a *free block*, and could be pruned if desired by its owner. Figure 1 depicts a blockchain with n guard blocks and $h - n$ free blocks. In blockchain systems using non-probabilistic consensus algorithms (e.g. PBFT [23]), such as Quorum [10] or Hyperledger Fabric [11], it becomes unnecessary to make these distinctions. Technically, all blocks are both authoritative and free, as they cannot possibly be superseded by reorganizations.

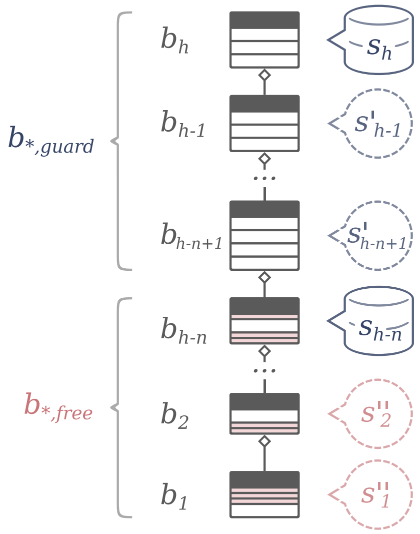

Fig. 1: A blockchain of h blocks. The latest n blocks serve as guard against reorganizations. See Table I for definitions.

Significant States: A state data structure, or state, is constructed by applying all transactions in a given block and all of its preceding blocks. Each block could be considered having an associated state, even if that state does not currently exist. Currently existing states are here referred to as being *derived*, while non-existing states are considered being either *derivable* or *retrievable*. The difference in being derivable or retrievable lies in whether or not enough information is had by a network participant to construct a given state, or if that information has to be acquired from other participants. It becomes relevant to talk about retrievable states when a given blockchain system supports pruning, as removing transactions could imply that some states seize to be derivable. Lastly, the state derived from the current block is referred to as the *current state*.

TABLE II: Logical descriptions of significant blockchain properties. See Table I for complementary definitions.

Expression	Description
Block Predicates	
$Unmodified(b_i) \impliedby Successor(b_{i+1}, b_i) \bigvee Unmodified(t_{i,*})$	The block b_i is known to be unmodified if either b_{i+1} can be proved to be the successor of b_i, or if the order and state of the transactions in b_i can be proved to be unmodified.
$Successor(b_i, b_{i-1}) \impliedby hash(b_{i-1}) = H_{i,predecessor}$	The block b_i is provably the successor of b_{i-1} if the hash of the predecessor b_{i-1} is equal to the *predecessor* field of the header of b_i.
$Valid(b_i) \impliedby apply(s_{i-1}; t_{i,*}) = s_i$	The block b_i is provably valid if applying all of its transactions to the derived state of the preceding block yields a valid state s_i.
Transaction Predicates	
$Unmodified(t_{i,j}) \impliedby Unmodified(t_{i,*}) \impliedby hash(t_{i,*} \cdot H_{i,\complement checksum}) = H_{i,checksum}$	The transaction $t_{i,j}$ is provably the jth transaction of the block b_i if the hash of all block transactions and relevant block header fields (here assumed to be all fields but the checksum) equals the checksum field of the header.
$Valid(t_{i,j}) \impliedby apply(s_{i-1}; t_{i,j}) \neq \varnothing$	The transaction $t_{i,j}$ is provably valid if applying it to the derived state of its preceding block yields any valid state.
State Predicates	
$Derivable(s'_i) \impliedby apply(s_{i-1}; t_{i,*}) = s_i \bigvee unwind(s_{i+1}; t_{i+1,*}) = s_i$	s_i can be constructed if all transactions in b_i can be applied to the state derived from the block preceding it, or if the transactions in the successor block b_{i+1} can be used to undo their changes to the successor state s_{i+1}.
$Retrievable(s''_i)$	The state s_i can be retrieved if either blocks, state and blocks, or just the state in question can be retrieved as needed from other network participants, and whatever data is acquired can be trusted to be correct.
Auxiliary Functions	
$apply(s_{i-1}; t_{i,*}) = \begin{cases} s_i, & \text{if } Valid(s_i) \\ \varnothing, & \text{otherwise} \end{cases}$	Represents the application of the given set of transactions $t_{i,*}$, where each transaction is ensured not to modify the given state s_{i-1} such that it describes a non-permitted outcome.
$hash(x) = y$	A function that takes whatever arguments given, combines them and turns them into a checksum. $\cdot$ denotes an argument combination function.
$unwind(s_i; t_{i,*}) = s_{i-1}$	Represents the undoing of the modifications made to the given state s_i by the given set of transactions $t_{i,*}$.

B. *Pruning and Property Changes*

Removing data from a blockchain may free up computer memory, but it could also lead to other property changes. Table II lists properties that may be maintained by a blockchain as logical predicates. It should be noted when reviewing the table that pruned blocks could lead to the loss of those properties.

A Property Preserving Hashing Procedure: To be able to prove that a block is unmodified, or that one particular block is the successor of another, requires the use of a hashing function $hash(x) = y$, as shown in Table II. Since the hashing function is expected to change its output y with the slightest alteration of x, pruning a block will normally lead to its hash changing, consequently making those properties unprovable. This could, however, be mitigated by using a hashing procedure that accounts for missing transactions. One such procedure combines stored hashes of pruned transactions with calculated hashes of the remaining such. Given that $\cdot$ is an arbitrary combination operator, and the definitions in Table II, the procedure could be defined formally as:

$$\begin{aligned} h_i &= hash(H_{i,\complement checksum} \cdot f(t_{i,*})) \\ f(t_{i,*}) &= g(t_{i,1}) \cdot g(t_{i,2}) \cdot \ldots \cdot g(t_{i,u}) \\ g(t_{i,j}) &= \begin{cases} t_{i,j,saved\ hash}, & \text{if } Pruned(t_{i,j}) \\ hash(t_{i,j}), & \text{otherwise} \end{cases} \end{aligned} \quad (1)$$

The same procedure would be used both to prove a given block is unmodified, and that it is the predecessor of its successor. An alternative procedure using Merkle trees [18] [1] could likely be used to reduce the number of stored hashes, but its definition is left as a topic for future research.

C. *Selective Pruning Algorithm*

The proposed selective pruning algorithm, illustrated in Figure 2, operates in three phases, namely *preparation*, *marking* and, lastly, *sweeping*. The reader should note that the algorithm is described in terms of these phases not because they are strictly required to occur in sequence, but rather because it makes it straightforward to describe the algorithm.

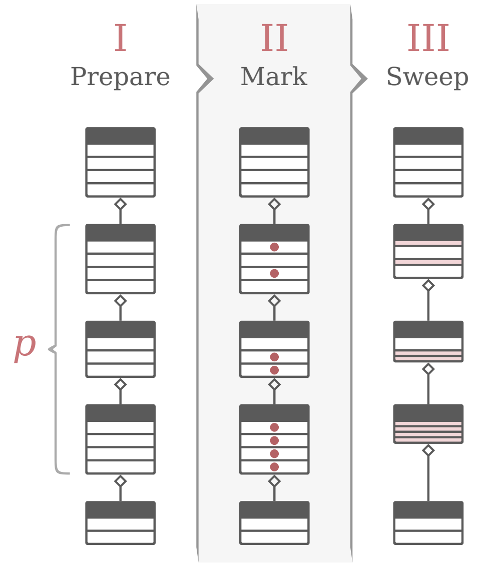

Fig. 2: The three phases of the selective pruning algorithm.

I. Preparation: The objective of the preparation phase is to identify the set of blocks that will be pruned p, and to identify and assemble any data d that will be required when considering individual transactions for pruning. That data could include one or more state data structures, which may have to be derived to become available. It could also include actual blocks, or external data not available through the pruned blockchain.

II. Marking: After p and d have been identified and assembled, every transaction $t_{i,j} \in b_i \in p$ is tested using a predicate function T, provided by the initiator of the pruning algorithm. T must satisfy:

$$T : \langle t_{i,j}; d \rangle \mapsto \{true, false\} \tag{2}$$

The position (i, j) of each transaction where T yielded $true$ is stored in a set R, which keeps track of all transactions pending removal. An example of a function satisfying T is given in Section IV-C. The formulation and analysis of other T-functions is left as a topic for future research.

III. Sweeping: Lastly, each transaction identified by R is removed. Additional measures may be required to ensure that memory is practically freed when the phase is over, such as compacting blocks to make room for additional such.

D. Selective Pruning and Maintaining Derivability

The blockchain properties presented in Section III-B and Table II imply that removing any transaction from a block leads to the state associated with that block to no longer be derivable, at least in its original form. There are, however, special cases when this is not true. Transactions may be categorized as being *generally significant*, *universally insignificant* or *retroactively insignificant* in relation to state derivability, where transactions in the latter two categorized may be pruned subject to certain conditions. The categories are defined below.

Significant Transactions: If the removal of a particular transaction results in the state associated with its block, or any succeeding block, no longer being derivable in its original form, the transaction in question is to be regarded as generally significant to state derivability. Given the definitions in Tables I and II, it may be expressed formally as:

$$\begin{aligned} Significant(t_{i,j}) \impliedby (Removed(t_{i,j}) \implies \\ \exists s'_x \; (x \geq i \; \wedge \; \neg Derivable(s'_x))) \end{aligned} \tag{3}$$

For example, consider transaction $t_{1,1}$ in Figure 3. If it is removed, then $[a : 1]$ would no longer be part of s'_1, and as a consequence $[a : 11]$ would no longer be associated with s'_2. As neither s'_1 or s'_2 would be derivable in their original forms, the transaction cannot be removed without impacting the derivability of any state.

Universally Insignificant Transactions: If removing one or more associated transactions in the same block does not lead to the state associated with the block becoming different, then those transactions are together considerable as universally insignificant to state derivability. As each state builds upon its predecessor state, if a set of transaction removed from the same block does not effect the state derived from that block, no subsequent state is affected either. Given the definitions in Tables I and II, and that $t_{i,J} \subset t_{i,*} \in b_i$, it may be described formally as:

$$\begin{aligned} Insignificant_u(t_{i,J}) \impliedby (Removed(t_{i,J}) \implies \\ Derivable(s'_i)) \end{aligned} \tag{4}$$

An example can be taken from Figure 3, where $t_{2,3}$ has no impact on c, as $3 \; (mod \; 5) = 3$. If $t_{2,3}$ is removed, then s'_2 remains unchanged, making the transaction universally insignificant to state derivability. To give an example where $t_{i,J}$ contains more than one transaction, consider b_3 in Figure 3. The block contains $add(a, 5)$, $add(a, 3)$ and $sub(a, 8)$, which if all applied leads to no change of a. As these transactions effectively cancel each other out, they become insignificant only if considered together.

Retroactively Insignificant Transactions: Given the existence of a set of states that no longer need to be derivable R', if removing one or more associated transactions from any blocks directly related to any member of R' does not lead to any state not in R' becoming different, then those transactions are together considerable as being retroactively insignificant to state derivability. Considering the definitions in Tables I and II, and that $t_{i,J} \subset t_{i,*} \in b_i$, it may be defined as:

$$\begin{aligned} Insignificant_r(t_{i,J}, R') \impliedby (Removed(t_{i,J}) \implies \\ \forall s'_x \; (x < i \; \vee \; s'_x \in R' \; \vee \; Derivable(s'_x))) \end{aligned} \tag{5}$$

Consider transaction $t_{1,2}$ in Figure 3. If removed, then s'_1 ceases to be derivable in its original form, but s'_2 remains unaltered. The reason for this is that $t_{1,2}$ is completely superseded by $t_{2,2}$, which overwrites the same b first set by $t_{1,2}$. Therefore, if $s'_1 \in R'$, then $t_{1,2}$ can be removed.

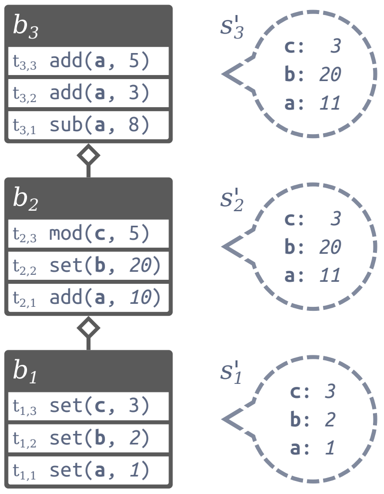

Fig. 3: A blockchain of 3 blocks, where each block has a derivable state. Transactions set or modify arbitrary integers associated with alphabetic keys. See Table I for definitions.

E. *Predicting Blockchain Growth*

Given a blockchain of h blocks, where the mean size of a block is μ_b, the mean size of a block header is μ_H, the mean size of a transaction is μ_t, the mean number of transactions per block is μ_u and ϵ represents any sources of error, then the size S of a blockchain may be defined as:

$$\begin{aligned} S &= h\mu_b + \epsilon \\ \mu_b &= \mu_H + \mu_t\mu_u + \epsilon \end{aligned} \tag{6}$$

If the mean block size is not known and cannot be estimated, Equation 6 may also be formulated with μ_b substituted as:

$$S = h(\mu_H + \mu_t\mu_u) + \epsilon \tag{7}$$

It should be noted that any of the variables in Equation 6 could be expressed as functions over one or more variables, such as time. Typical sources of error ϵ could include platform or storage media restrictions, such as page or block sizes, whether or not blocks are stored in multiple file system files, etc.

Example 1: Some blockchain currently consist of 40000 blocks, and is expected to grow with a rate of 120 blocks per day, where the mean size of a block is known to be constant at 0.30 MB. Given that d is the number of days elapsed since the current time, $h = (40\ 000 + 120d)$, and $\epsilon = 0$, then its size in MB is:

$$\begin{aligned} S &= h\mu_b + \epsilon \\ &= (40\ 000 + 120d)0.30 + 0 \\ &= 12\ 000 + 36d \end{aligned}$$

Accounting for Pruned Blocks: Given that n is the number of unpruned blocks in some blockchain, that $\mu_{p(b)}$, $\mu_{p(H)}$, $\mu_{p(t)}$ and $\mu_{p(u)}$ are the mean sizes of pruned blocks, pruned headers, pruned transactions and average number of pruned transactions per block, respectively, then pruned blockchain size $\tilde{S}$ could be calculated using:

$$\begin{aligned} \tilde{S} &= n\mu_b + (h-n)\mu_{p(b)} + \epsilon \mid h \geq n \\ \mu_b &= \mu_H + \mu_t\mu_u + \epsilon \\ \mu_{p(b)} &= \mu_{p(H)} + \mu_t(\mu_u - \mu_{p(u)}) + \mu_{p(t)}\mu_{p(u)} + \epsilon \end{aligned} \tag{8}$$

Example 2: A blockchain is expected to grow with a pace of 1 block every two minutes, or about 262800 block per year. The mean sizes of both unpruned and pruned blocks are known to be constant at 2.30 MB and 0.50 MB, respectively. Equation 8 is used to calculate the expected size of the blockchain after 19 years. Given that the last $n = 1\ 000$ blocks will need to be kept unpruned, $h = 262\ 800 \cdot 19$, and $\epsilon = 0$, then the blockchain size in MB is expected to be:

$$\begin{aligned} \tilde{S} &= n\mu_b + (h-n)\mu_{p(b)} + \epsilon \\ &= 1\ 000 \cdot 2.30 + (262\ 800 \cdot 19 - 1000)0.5 + 0 \\ &= 2\ 498\ 400 \end{aligned}$$

If, on the other hand, the maintainers of the blockchain would have refrained from pruning any blocks, its size in MB would have been:

$$\begin{aligned} S &= h\mu_b + \epsilon \\ &= (262\ 800 \cdot 19)2.30 + 0 \\ &= 11\ 484\ 360 \end{aligned}$$

The size reduction gained from pruning is in this scenario $1 - (2\ 498\ 400 \div 11\ 484\ 360) \approx 78.25\%$.

Blockchain Growth Linearity: Pruning a blockchain may yield significant savings in storage space requirements, but can only be used to limit the size of a blockchain to a desired threshold if the mean size of a pruned block $\mu_{p(b)}$ can be or approaches zero. Figure 4 depicts five different growth trajectories, calculated using Equation 8, for blockchains where pruning leads to different changes in mean block size. Only one trajectory, where $\mu_{p(b)} = 0$, represents a fixed storage requirement relative to the number of unpruned blocks n. Hence, blockchain growth may be assumed to always be linear over time, pruning or not, unless the size of a pruned block is effectively zero. Whether or not it is reasonable to remove entire blocks, including their headers, would depend on the expectation that any of the information contained in a block will be useful in the future.

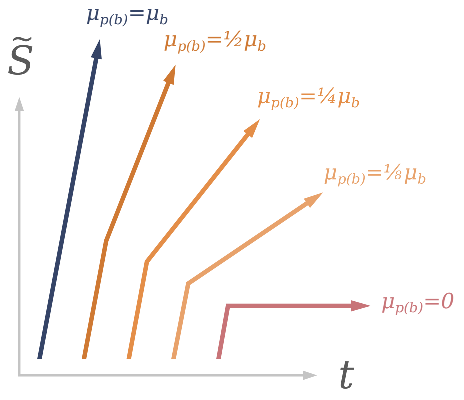

Fig. 4: The impact of pruning on blockchain size over time. μ_b and $\mu_{p(b)}$ are assumed to be constants. Each line represents a growing blockchain where n blocks are kept unpruned.

IV. Selective Transaction Pruning in Hyperledger Fabric

This section presents the application of the theory presented in Section III on Hyperledger Fabric. It begins with a brief overview of the system for readers not already familiar with it and continues with a presentation of the modifications made to it for it to support selective transaction pruning. Finally, it ends with a description of an asset-delivery use case followed by benchmark results, which are presented to verify that selective transaction pruning is possible and yields predictable results.

A. Overview of Hyperledger Fabric

Hyperleger Fabric [11] is a blockchain system developed and maintained by Hyperledger [24], a project hosted by the Linux Foundation aimed at creating and maintaining open source blockchain systems for enterprise applications. The system is significantly permissioned and relies on non-probabilistic consensus. The transactions of its blockchain describe invocations of chaincode functions, where a chaincode is a form of containerized application serving as a smart contract [25]. Each valid chaincode invocation may result in the key/value store associated with the system running the chaincode being updated.

B. The Pruning Extension

Hyperledger Fabric was significantly extended to support *selective transaction pruning* and *blockchain size analysis*. To allow the abilities in Table II associated with hashing to be preserved to some extent, the block *hashing procedure* was also modified. The remainder of this subsection is dedicated to outlining how these features were implemented. The reader should, however, first note that when the presented features were developed, the more stable version of Fabric was deemed to be version 0.6 [17]. This means that details mentioned below may not be true for more recent versions of Fabric.

Selective Transaction Pruning: The pruning algorithm described in Section III-C requires a provided predicate function for the purpose of determining which transactions to prune. Hyperledger Fabric allows its blockchain to be modified only via chaincodes, which are deployed independently by the maintainers of the nodes participating in the transaction validation process. Chaincodes do only have to be functionally equivalent for the system to operate, meaning that each maintainer could use its own implementation. To allow each participant to also provide its own pruning predicate function, the chaincode messaging protocol and procedures were extended for the purpose. If provided, the predicate function is called exclusively with transactions generated or validated using the same chaincode, meaning there is no way for one pruning predicate function to decide whether transactions generated by other chaincodes are to be removed. When invoked via an added REST [26] endpoint, the implemented pruning procedure proceeds as follows:

I. PREPARE: An effective read-only copy of the most recently assembled state data structure is created and used as d. All blocks, from the most recent to the oldest, used to derive the copied state are used as p.

II. MARK: All non-pruned transactions in p are provided together with d, one at a time, to the pruning predicate function associated with the chaincodes first used to generate them, if such exists. Each transaction provided to a pruning predicate function returning *true* is marked for pruning.

III. SWEEP: Each block is deserialized. Marked transactions are hashed, have their contents removed, their types changed to `PRUNED`, and their `metadata` fields set to their hashes. Finally, the block in question is serialized and saved over its previous version. When no more blocks to prune remain, d is removed.

Two things should be noted by the reader about the presented selective pruning procedure. Firstly, Hyperledger Fabric uses a non-probabilistic consensus algorithm. Therefore, no transactions already used to construct a state data structure are technically required to participate in the validation of new blocks (see Section III-A). Secondly, as pruning predicate functions have arbitrary implementations, guarantees about being able to derive useful state data structures from pruned blocks depend on those implementations. An example of such an implementation is given in Section IV-C.

Blockchain Size Analysis: Measuring the impact of pruning requires a way for block size metrics to be collected. As blocks are stored in serialized form in a database, Hyperledger Fabric was extended to gather block byte sizes by iterating through blocks and checking the number of bytes used to represent them. In order to also measure the sizes of headers and transactions, blocks are deserialized and their transactions serialized and measured, one by one. To calculate the size of each block header, the size sum of the transactions belonging to the same block is subtracted from the size of the entire block. Given that $S_{i,b}$ is the byte size of the ith block, $S_{i,t,*}$ is the byte size of the transactions of the same block, u_i is the number of transactions it contains, $S_{i,H}$ is the block header byte size, and, finally, $\psi(x)$ is a function that serializes and determines the byte size of x, the measurements may be expressed formally as:

$$\begin{aligned} S_{i,b} &= \psi(b_i) \\ S_{i,t,*} &= \sum_{j=1}^{u_i} \psi(t_{i,j}) \\ S_{i,H} &= S_{i,b} - S_{i,t,*} \end{aligned} \tag{9}$$

As Google Protocol Buffers [27] is used as serialization format, there is some byte overhead associated with designating consecutive transactions as members of a collection. This collection overhead varies with the number of transactions in the collection and affects calculated sizes of block headers. To make the collected metrics accessible from outside Hyperledger Fabric, existing REST [26] endpoints were modified such that blocks and transactions are served together with information about their sizes. It should be noted that the reported sizes do not account for any other storage space than that of the blocks themselves. Database indexes and other overhead is not accounted for.

Property Preserving Hashing Procedure: The Block hashing procedure used by Hyperledger Fabric only entails feeding a serialized block to a hash function. If a block is pruned, its serialized form becomes different, which means that it yields a new output if provided again to the hashing procedure. Pruned blocks cannot be proved to be unmodified or to be the predecessors of their successors unless the hashing procedure is modified as described in Section III-B. As it was assumed to be meaningful to retain these properties even if transactions are pruned, the procedure described by Equation 1 was implemented. Pruned transactions are replaced with only their hashes and a `PRUNED` type indicator. As Hyperledger Fabric blocks hold a `nonHashData` header field, the implementation ignores this field.

C. *Use Case: Asset Delivery Network*

One proposed use case for systems such as Hyperledger Fabric is supply chain management [12]. Supply chains may cross political, geographical or cultural boundaries, and could require the cooperation of stakeholders with conflicting interests. Blockchain technology could be a way to manage fairness where trusted middle-men or other kinds of arbiters are hard to agree on. To verify that selective transaction pruning could be used to reduce ledger size and preserve significant transactions, a naive supply chain scenario was formulated with the intent of reflecting the interactions of different stakeholders in an asset delivery network. No attempt is made to account for subjects of contention, such as false claims about delivery times or asset locations. The scenario, depicted in Figure 5, includes 18 *sites* connected via unidirectional *routes*, through which *assets* are delivered via different kinds of media, such as shipping or trucking. All significant delivery events are registered with a member of a Hyperledger Fabric cluster through a chaincode written for the purpose.

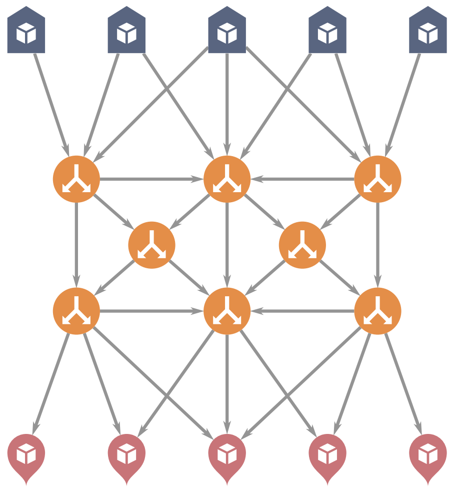
Fig. 5: A delivery network of 5 warehouses, 8 distribution centers and 5 delivery points connected via 33 unidirectional routes, allowing assets to be transferred via 387 different paths.

Chaincode: The chaincode written to support the use case allows assets to be added to warehouses, forwarded from sites to routes, and received from routes to distribution centers or delivery points. When assets are received the opportunity is given to the receiving party to claim the asset has been lost. Assets successfully received at a delivery point may be removed. Chaincode functions also exist for adding sites and routes, as well as for querying the state of the delivery network. Finally, a pruning predicate function was defined as part of the chaincode. The function is implemented such that the state directly associated with the last main block remains fully derivable, as defined in Section III-D, which is facilitated by the implementation outlined in Algorithm 1.

Simulation: An external application was written to simulate the behavior of the stakeholders of the delivery network. The application is started with an asset cardinality and a seed for its pseudo-random number generator. When initiated, the application operates in delivery rounds. Each round, assets are generated, forwarded, lost or removed. A bounded random number of assets are generated and assigned to a random path through the delivery network. Assets currently in transit are forwarded, from site to route or from route to site, via their previously assigned paths. Each time an asset is received at a site it has a risk of being lost, determined by a probability property configured for each route. The first delivery round after an asset has been successfully received at a delivery point, it is removed. When no more assets remain to be generated or moved, the application is terminated.

Algorithm 1 Pruning predicate function associated with use case chaincode. The function only returns $true$ if $transaction$ affects key/value pairs not present in $state$. This will be the case only if $state$ represents a point in time after which the asset concerned by $transaction$ was successfully delivered and removed.

```
function IsPrunable($transaction$, $state$)
    $keys \leftarrow$ ResolveAffectedKeys($transaction$)
    $values \leftarrow$ GetStateValuesByKeys($state$, $keys$)
    return IsEmpty($values$)
end function
```

Benchmark: The simulation application was executed with the instruction to move 1000 assets through the scenario delivery network, out of which 120 were randomly selected to be lost at random routes of their delivery paths. When the simulation completed, relevant size metrics were collected, and the one modified Hyperledger Fabric peer used to maintain the blockchain was instructed to prune it. After pruning, size metrics were collected again and then compiled into the statistics available in Table III. Some of the statistics are also illustrated in Figure 6. Significantly, the size of the entire blockchain was reduced from about 9.51 MB to 1.47 MB, which is a size reduction of circa 84.49%. The reader may note that the average block header size decreased from 277.36 B to 253.22 B, despite that no header values of any block were altered in any way. This reduction is related to the way the header size is calculated, described in Section IV-B.

Projection: Assume that blocks are generated at a rate of 1 block per unit of time t, that the blockchain is expected to keep growing with constant block mean size, and that the blockchain is pruned every time a new block is added. Given the definitions in Tables I and III, Equation 8, $n = 0$, $h = t$, and $\epsilon = 0$, then could the byte size of the blockchain $\tilde{S}$ be:

$$\begin{aligned}\tilde{S} &= n\mu_b + (h-n)\mu_{p(b)} + \epsilon\\ &= 0 \cdot 22\ 376.50 + (t-0)3\ 470.28 + 0\\ &= 3\ 470.28t\end{aligned}$$

At $t = 425$ would $\tilde{S} = 3\ 470.28 \cdot 425 = 1\ 474\ 869$, which is approximately the same as $\tilde{S}$ in Table III. If $n = 100$ blocks were kept unpruned at $t = 425$, then would the size have been:

$$\begin{aligned}\tilde{S} &= n\mu_b + (h-n)\mu_{p(b)} + \epsilon\\ &= 100 \cdot 22\ 376.50 + (425-100)3\ 470.28 + 0\\ &= 3\ 365\ 491\end{aligned}$$

TABLE III: Asset delivery simulation statistics. Fractional numbers are rounded to two decimal places.

	Pruned		Original	
Block Size				
Mean	$\mu_{p(b)} = 3\ 470.28$ B	$\sigma_{p(b)} = 1\ 705.41$ B	$\mu_b = 22\ 376.50$ B	$\sigma_b = 4\ 862.87$ B
Extremes	$\max_{p(b)} = 27\ 407$ B	†$\min_{p(b)} = 219$ B	$\max_b = 28\ 357$ B	†$\min_b = 918$ B
Total	$\tilde{S} = 1\ 474\ 868$ B		$S = 9\ 510\ 013$ B	
Block Header Size				
Mean	$\mu_{p(H)} = 253.22$ B	$\sigma_{p(H)} = 24.71$ B	$\mu_H = 277.36$ B	$\sigma_H = 29.56$ B
Extremes	$\max_{p(H)} = 313$ B	†$\min_{p(H)} = 151$ B	$\max_H = 313$ B	†$\min_H = 152$ B
Total	$\tilde{S}_H = 107\ 619$ B		$S_H = 117\ 877$ B	
Transaction Size				
Mean	$\mu_{p(t)} = 123.80$ B	$\sigma_{p(t)} = 201.63$ B	$\mu_t = 850.43$ B	$\sigma_t = 12.59$ B
Extremes	$\max_{p(t)} = 892$ B	$\min_{p(b)} = 68$ B	$\max_t = 895$ B	$\min_t = 815$ B
Total	$\tilde{S}_t = 1\ 367\ 249$ B		$S_t = 9\ 392\ 136$ B	
Transactions per Block, Mean	$\mu_{p(u)} = 25.99$	$\sigma_{p(u)} = 5.48$	$\mu_u = 25.99$	$\sigma_u = 5.48$
Cardinalities				
Blocks	$h = 425$		$h = 425$	
Transactions	unpruned $= 768$	pruned $= 10\ 258$	unpruned $= 11\ 044$	pruned $= 0$
Assets	delivered $= 880$	lost $= 120$	delivered $= 880$	lost $= 120$

† The always empty genesis block, which in this case is 16 B, is not considered.

V. Discussion

Limiting Blockchain Growth: It was shown in Section III-E that pruning can only reduce, not effectually limit, the rate of blockchain growth unless pruned blocks can be removed completely. If this is possible in a given scenario depends on whether older blocks contain useful data, in the context of consensus or otherwise, as explained in Section III-A.

The Ratio of Prunable Transactions: The blockchain produced by the scenario presented in Section IV-C could be reduced in size by about 84.49% largely because circa 92.88% of transactions could be pruned. This ratio is high because (1) most of the transactions were subjectively decided to not contain interesting information, and (2) most transactions dealt with information having limited lifespans. Assets were introduced, moved through the sites and routes of the delivery network, and finally removed if nothing exceptional happened. This implies that the gains of pruning could vary greatly with different kinds of use cases.

Identifying Prunable Transactions: In Section III-D conditions for pruning transactions are presented, and in Section IV-C an example of a working pruning predicate function implementation is given. The example implementation is only able to identify transactions that deal with data that has later been removed, but any set of eligible transactions that effectively cancel each other out, or are fully superseded by later transactions, could be prunable. What would function implementations look like for pruning other—potentially highly complicated—sets of transactions? Could such an implementation be identified that is guaranteed to find all prunable transactions modifying, for example, a key/value store? Further work on these and related questions could lead to selective transaction pruning becoming more generally applicable.

Sharing Pruned Blocks: What would happen if a blockchain network participant, perhaps in the process of rejoining its network, was provided with a pruned chain of blocks? Through the use of a hashing procedure similar to that presented in Section IV-B, the received blocks could be decided to contain only valid remaining transactions. There would, however, be no way for the receiver to know whether transactions it deems significant have been pruned, meaning that it cannot be decided whether the state data structures derivable from the received blocks are useful or not. This could perhaps be mitigated by having network participants collectively agree on which states are significant, via state hashes or otherwise. After applying pruned blocks, network participants could determine if the significant states are unmodified. The feasibility of such an approach could be a topic for future research.

Irrevocably Lost Transactions: Even though a blockchain network may continue to function if some transactions are removed by all participants, it could be undesirable that information can be irrevocably lost without any chance for it to be detected. A solution could include assigning different portions of the past to special historian nodes. Another could be to make pruned transactions have a random chance of becoming forever protected from pruning, which could be tuned to make the event of permanent transaction loss unlikely.

Pruning Performance: In this paper, the only performance metric of concern has been that of disk space. It could be expected, however, that the pruning algorithm presented in Section III-C may use significant computer resources while being executed, such as primary memory or processor time. As the algorithm requires the consideration of every transaction in every considered block, it could be assumed to have something reminiscent of a linear relationship between the number of considered transactions and processing time. The implementation presented in Section IV-B is able to execute while also participating in the process of accepting new blocks and updating its current state. If the algorithm cannot be modified to yield better than linear performance characteristics, then there might be additional ways to avoid degrading the performance of more critical systems tasks, such as the consensus process.

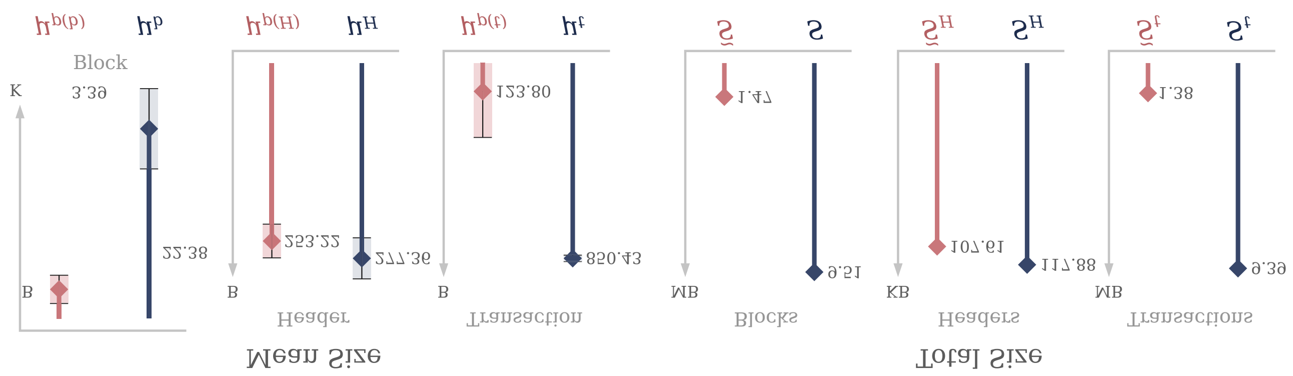

Fig. 6: Plotted asset delivery simulation statistics. All numbers are rounded to two decimal places.

VI. Conclusions

It is shown in Sections III and IV that selective transaction pruning is possible, theoretically and practically. The anatomy of a general blockchain is presented, as well as descriptions of how such maintains significant properties, a selective pruning algorithm, conditions for selective transaction pruning to not affect significant state derivability, and methods for predicting blockchain growth. A modified version of Hyperledger Fabric [17] is used to demonstrate that a blockchain with transactions from an artificial supply chain scenario could be reduced in size with 84.49% by pruning 92.88% of its transactions. It is our conclusion that selective transaction pruning is a generally viable approach to limiting blockchain growth while keeping transactions of interest, and that it could fruitfully be applied in any context where the benefit of freeing up memory outweighs the gains of having a complete blockchain.

Acknowledgements

Dr. Johan Kristiansson, formerly at Ericsson Research in Luleå, provided indispensable guidance. Also, this work was funded via the *Productive 4.0* project (EU ARTEMIS JU grant agreement no. 737459).

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[3] T. Bocek, B. B. Rodrigues, T. Strasser, and B. Stiller, "Blockchains everywhere – a use-case of blockchains in the pharma supply-chain," in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE, 2017, pp. 772–777.

[4] M. Swan, *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc., 2015.

[5] E. Hofmann and M. Rüsch, "Industry 4.0 and the current status as well as future prospects on logistics," *Computers in Industry*, vol. 89, pp. 23–34, 2017.

[6] Saint Bitts LLC. (2017) Blockchain size. Accessed 2017-11-22. [Online]. Available: https://charts.bitcoin.com/chart/blockchain-size

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[8] EtherScan. (2017) Ethereum ChainData size growth (FULL Sync). Accessed 2017-11-22. [Online]. Available: https://etherscan.io/chart/chaindatasizefull

[9] M. Hearn. (2016) Corda: A distributed ledger. Accessed 2018-02-15. [Online]. Available: https://docs.corda.net/_static/corda-technical-whitepaper.pdf

[10] JP Morgan Chase. (2016) Quorum white paper. Accessed 2018-02-15. [Online]. Available: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum Whitepaper v0.1.pdf

[11] E. Androulaki, A. Barger *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, pp. 30:1–30:15. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190538

[12] H. Wu, Z. Li, B. King, Z. Ben Miled, J. Wassick, and J. Tazelaar, "A distributed ledger for supply chain physical distribution visibility," *Information*, vol. 8, no. 4, 2017, accessed 2018-01-17. [Online]. Available: http://dx.doi.org/10.3390/info8040137

[13] Bitcoin Project. (2017) Bitcoin core. Accessed 2017-11-23. [Online]. Available: https://bitcoin.org/en/bitcoin-core

[14] J. D. Bruce. (2014) The mini-blockchain scheme. Accessed 2017-11-23. [Online]. Available: http://cryptonite.info/files/mbc-scheme-rev3.pdf

[15] R. Dennis, G. Owenson, and B. Aziz, "A temporal blockchain: a formal analysis," in *Collaboration Technologies and Systems (CTS), 2016 International Conference on*. IEEE, 2016, pp. 430–437.

[16] A. Chepurnoy, M. Larangeira, and A. Ojiganov, "Rollerchain, a blockchain with safely pruneable full blocks," *arXiv*, 2016, accessed 2018-04-20. [Online]. Available: https://arxiv.org/abs/1603.07926v3

[17] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[18] R. C. Merkle, "A certified digital signature," in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 218–238.

[19] K. Saito and H. Yamada, "What's so different about blockchain? – blockchain is a probabilistic state machine," in *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 168–175.

[20] M. Hearn and M. Corallo, "Connection bloom filtering," Bitcoin Improvement Proposal, Bitcoin Project, BIP 0037, 2012, accessed 2017-12-01. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki

[21] L. Baird. (2016) The swirldls hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Accessed 2017-11-27. [Online]. Available: http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

[22] S. Popov. (2017, October) The tangle. Accessed 2017-11-27. [Online]. Available: https://iota.org/IOTA_Whitepaper.pdf

[23] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[24] Linux Foundation. (2017) Hyperledger. Accessed 2018-01-26. [Online]. Available: http://hyperledger.org

[25] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[26] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000, vol. 7.

[27] Google. (2018) Protocol buffers. Accessed 2018-01-30. [Online]. Available: https://developers.google.com/protocol-buffers