An introduction to Public and Private Distributed Ledgers

Jan Felix Hoops
Advisor: Dr. Holger Kinkelin
Seminar Innovative Internet Technologies and Mobile Communications SS2017
Chair of Network Architectures and Services
Departments of Informatics, Technical University of Munich
Email: jan.felix.hoops@in.tum.de

ABSTRACT

Blockchains are fairly novel systems that offer highly reliable, non-mutable and trustworthy storage of records. Applications include distributed payment systems like Bitcoin and smart contract systems like Ethereum. This paper first gives an introduction to blockchain technology using the example of Bitcoin. One of the foci is the so called consensus algorithm that controls which new block can be added to the blockchain. Furthermore we explain why the consensus algorithm of "traditional" blockchains is not suitable for smaller application scenarios such as corporate networks and also is highly wasteful concerning energy consumption. As a next step, the Linux FoundationâĂŹs Hyperledger Project is introduced, which is an umbrella for various open source projects concerning private blockchains. Here the focus is on the blockchain implementation Sawtooth, which features a new consensus algorithm called Proof of Elapsed Time that makes use of a specific hardware security feature of novel Intel CPUs. This algorithm features the best properties of Proof of Work while also eliminating the huge waste of power. However, the dependency on the specific CPU features can also be seen as a major weakness of the concept.

Keywords

blockchain, private blockchain, distributed ledger, consensus, Bitcoin, Hyperledger, Sawtooth, Proof of Work, Proof of Elapsed Time

1. INTRODUCTION

Blockchains are one of the most impactful technologies of the new millennium, even though much of their potential still remains undiscovered. They finally allow for the realization of truly distributed ledgers that do not require a trusted third party. They also raised the bar for non-mutability and reliability, thanks to their high redundancy. These properties make blockchains attractive for numerous public and corporate applications everywhere there are records to keep [12].

The technology emerged in 2008 when Satoshi Nakamoto presented Bitcoin [24], the first decentralized, digital currency. That is why it is the example used to illustrate the explanation of blockchains in section 2. To this day, public crypto-currencies are still the dominant applications using blockchains. More recent ones like Ethereum [5] do not only offer currency, but expand on the idea. Ethereum for example further innovates by featuring a smart contract system.

Even though private blockchains (addressed in section 3) have not seen as much success as public ones yet, there are a lot of big companies actively supporting ongoing research. This difference in development compared to their public counterpart is mainly caused by traditional consensus algorithms not scaling down well. It will be argued in section 3.2, that applying Proof of Work to a small network essentially breaks the system. However, progress is being made and one promising project is presented in section 4. The project in question is Intel's Sawtooth. It is a modular enterprise solution for building private blockchains. Sawtooth features some architectural upgrades compared to a traditional blockchain, but most importantly introduces a new consensus algorithm: Proof of Elapsed Time. This algorithm is very promising for future blockchain development, because it features the strengths of Proof of Work, while also eliminating the large amount of wasted energy.

2. PUBLIC BLOCKCHAINS

A blockchain can be seen as distributed database with high redundancy. It is replicated using a peer-2-peer system and every participating node keeps a full copy. The major benefit of a blockchain is that it provides high trustworthiness without relying on a trusted third party. The absence of a trusted third party involved in consensus also renders blockchains more secure by removing an angle of attack while also cutting costs and complexity.

The following overview will be using the example of Bitcoin. However, the text is kept as generic as possible. Bitcoin is a crypto-currency using a blockchain as a distributed, public ledger and probably the most commonly known application of blockchain technology. The focus of this overview is on the architecture of the blockchain, not the underlying network. Technical details presented in the following subsections were taken from the Bitcoin Developer Reference [2], that can also be recommended for information about the underlying peer-2-peer network.

2.1 Transactions

In contrast to a database, a blockchain stores no tables, but transactions. Generally a transaction transfers an asset from one individual to another. These assets can be of any imaginable kind and the individuals do not need to be human.

Bitcoin transactions are financial transactions. Their structure can be seen in Table 1. As indicated by the fields `tx_in count` and `tx_out count`, a transaction is not limited to only one sender or receiver. These fields contain the number of

Table 1: Bitcoin Transaction Structure [2]

Name	Data Type
version	uint32_t
tx_in count	compactSize uint
tx_in	txIn
tx_out count	compactSize uint
tx_out	txOut
lock_time	uint32_t

Table 2: Bitcoin Block Structure [2]

Name	Data Type
fixed value	0xD9B4BEF9
block size	uint32_t
block header	header
transaction counter	varInt
transactions	transaction list

inputs and outputs (`tx_in` and `tx_out`) of a transaction. Every input references a previous output and contains proof of ownership. Every output specifies the amount of bitcoins to spent and the new owner. Ownership in Bitcoin is handled via asymmetric cryptography. Every individual is identified by a bitcoin address computed from the public key of it's asymmetric key pair.

Blockchain transactions can also be timed. Bitcoin implements this using the `lock_time` field. A transaction may not be processed before the time given there. The field's unsigned integer value is parsed dependent on size: below 500 million it is interpreted as a block height and above as a unix epoch.

In Bitcoin there is one special transaction: the coinbase transaction. It is used to create new bitcoins from nothing. Technically this is realized by using only one input, the coinbase input. It is like a normal input except that it has placeholder values for most fields. The necessity of this transaction is related to mining and will be discussed in section 2.3.

2.2 Blocks

Blocks are used to group up transactions. They consist of a header for meta data and a list of transactions. Every block is connected to its predecessor by containing the hash of the predecessor's header in its own header. The header

Table 3: Bitcoin Block Header Structure [2]

Name	Data Type
version	uint32_t
previous block header hash	char[32]
merkle root hash	char[32]
time	uint32_t
nBits	uint32_t
nonce	uint32_t

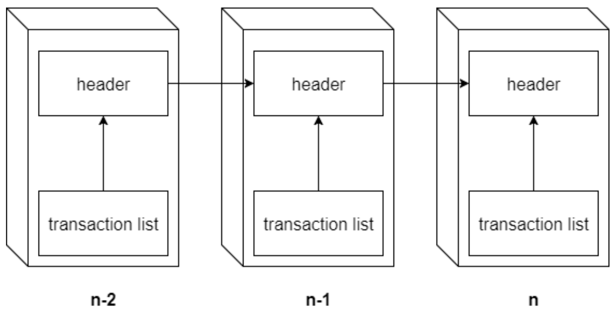

Figure 1: Simplified Blockchain Structure
Every arrow symbolizes a hashing operation. The origin is the value being hashed and the target is where the hash value will be included.

hash is used to uniquely identify a block. By constructing a merkle tree over all transactions of a block and storing the root in the header of said block, it is ensured that the transactions also influence the block header hash. This secures the transactions, because it allows for easy detection of manipulation.

Table 2 shows the structure of a block as used by Bitcoin and the corresponding header can be seen in Table 3.

Bitcoin implements the `previous block header hash` by hashing twice with `SHA256`. The `merkle root hash` is the hash in the root of the merkle tree hashing all transaction ids of this block. If however the block only contains one single transaction (coinbase transaction), its transaction id is used as the merkle root. The `time` field contains a time stamp in the standard unix format, making it possible to enforce the `lock_time` found in transactions. Both `nBits` and `nonce` field hold values important for Bitcoin's consensus mechanism, which will be addressed in section 2.3. In addition to all of these specifications, blocks can also have a data size limit. For Bitcoin, a block may never exceed 1MB in size.

Figure 1 shows the essence of a blockchain's structure. The schematic shows part of a blockchain with emphasis on the chain of hashes. Every arrow symbolizes a hashing operation. It can be seen, that the header of block `n` contains the hash of the header of block `n-1`. These are the horizontal arrows. The vertical ones show the transactions of a block being hashed into its header.

Only the very first block of a blockchain has to deviate from the exact shown structure as it has no predecessor. This block is a kind of dummy block. It is called the genesis block and contains placeholder values. This block is hardcoded in every client for a blockchain.

2.3 Consensus with Proof of Work

The basic problem a blockchain solves is distributed consensus. In a blockchain this means that all participating nodes have to agree on the newest block added to the chain, meaning they implicitly agree on one chain of blocks. In order to achieve consensus, some form of voting is necessary among these nodes. This consensus mechanism is the heart of every distributed ledger and arguably its most defining feature. The ledger's characteristics (e.g. transaction throughput, supported number of nodes) all depend mostly on this mechanism.

Bitcoin uses a leader election protocol (refer to section 3.2) in order to decide on a node being eligible to create the next block. This specific system is the *Proof of Work* (PoW) system. When creating a block, a *crypto-puzzle* has to be solved. To be considered valid, a block's header hash has to be less or equal to the current target threshold. This threshold is adjusted regularly in such a way that a new block is created every 10 minutes on average and is encoded in the `nBits` header field. The header hash is computed by hashing twice with `SHA256`.

A node trying to solve this crypto-puzzle and creating new blocks is referred to as a *miner*. Miners collect transactions requested by other nodes and group them up into a block. In order for the header's hash to meet the required threshold the miners are able to modify the header in the following ways: First and most obvious, they can vary the `nonce` field in the block header. Second, the order of the transactions may be varied (except the coinbase transaction) in order to change the `merkle root hash` in the header. Third and last option is to change one of the fields (`coinbase script`) in the coinbase input.

As long as the used cryptographic hashing function is secure, solving this puzzle is computationally expensive due to the miners simply having to brute force a solution. Because of this, this system can be viewed as a lottery with computational power only affecting the likelihood of winning.

A miner has to invest in hardware and electricity. In order to incentivize nodes to do mining, Bitcoin has a block reward. This is a certain amount of bitcoins that is created in the coinbase transaction as a reward to the miner in addition to the transaction fees. However this reward is halved every 210,000 blocks. That is necessary in order to limit the amount of bitcoins that will ever be created and in doing so, preserving their value. On the downside this means that in the not so distant future the transaction fees will be all the reward a miner receives.

2.4 Blockchain Security Mechanisms

Anyone can create a transaction, but creating a valid one requires proving the ownership of the asset(s) in transfer. Validity in this context means that other nodes of the network will consider the transaction fit to be included in a block on the basis of the current blockchain (e.g. asset was not already spent) and the common specification (i.e. correct transaction format). The transaction is then published onto the network where every node on the way will also check the transaction's validity before distributing it further. If found to be valid, a transaction will be kept by mining nodes to be incorporated into a block. Only after a transaction was published inside a block, it is considered to have been conducted.

Whenever a node solves the proof of work it has finished a new block. The validity of this block depends on the proof of work matching the current threshold, the `merkle tree root` matching the transactions as well as their order and the validity of each transaction. The node then distributes the new block along the network. Every node along the way checks the block's validity before distributing it further and adding it to it's local image of the blockchain.

Since a miner can append his new block anywhere in the chain, the blockchain may fork. Bitcoin's solution to this problem is simple yet effective: In this event the longest chain of blocks is the one agreed upon. This works because creating blocks is expensive. An individual will never have the computational power needed to fork the chain and create a longer chain than all of the other nodes in the meantime. Honest nodes will always switch to mining on the highest valid block, because they will not get any rewards if their mined block is not part of the longest chain. By working on a successor to a block, a node implicitly expresses it's acceptance.

For this blockchain to work there are mainly two assumptions made. The first one is that the majority of nodes are honest. The second one is that no node ever reaches more than 50% of the networks computational power. Both of these make sure that the majority of blocks is always created by honest nodes.

2.5 Attacks on Public Blockchains

This section introduces three attacks on public blockchains and evaluates them. The main purpose of this section is less to show weaknesses of public blockchains, but rather to illustrate why they are so secure.

The most realistic attack on bitcoin is the *double spend* [25]. The goal of this attack is to spend the same transaction output twice. This is possible to a certain extent because a block can later on become invalid if another branch of the chain becomes longer than it's own one. An attacker could buy an item from a seller. As soon as his transaction is encapsulated in a valid block, the seller would give the item to the attacker. Then the attacker would need to fork the blockchain right before his transaction and make it longer than the original branch. This way he could void his transaction while already having obtained the item. In reality this is very difficult to execute as it requires vast amounts of computational power and even then this is still a gamble. As the attacker would most likely have less than 50% of the networks computational power he would have to be lucky in order to overtake the original branch. To conclude, it can be said that this attack is theoretically possible, but not feasible due to the amount of funds and/or luck it requires.

Another take on the double spend attack is referred to as *fast double spend*. Some exchanges with bitcoin require a certain speed. A supermarket, for example, can not wait for about 10 minutes for the next block to encapsulate the transaction. These vendors instead rely on just seeing the customer's transaction on the network. An attacker could possibly create the anticipated transaction while also creating a malicious second transaction that spends the same transaction output as the first one, but this time transfers it to another wallet under the attacker's control. Then both transactions are distributed across the network. The attacker has to make sure the first transaction is broadcasted to the vendor while broadcasting the malicious transaction to most other nodes. This increases the chances of the malicious request being mined first. As both transactions contradict each other, only the first one to be mined will ever be mined. For more detail on this attack refer to [21]. In practice it is still rather difficult and depends on luck.

All of these double spend attacks can be averted by simply waiting. With every block succeeding the one with the transaction in question it gets exponentially more difficult to undo the block. That is because an attacker would have to fork the chain in front of that block, redo all succeeding blocks and overtake the original branch all while the network is still mining on said original branch. After only about 6 succeeding blocks a Bitcoin transaction can be considered rather safe [25]. On the downside, this means having to wait for about an hour.

It should also be mentioned that another angle of attack is strategically withholding and publishing mined blocks such as described in [20]. This kind of attack is referred to as *selfish mining*. Normally an individual should earn a percentage of the total block rewards approximately matching it's percentage of the network's total computational power. The goal of this attack is to increase the percentage of the reward without increasing computational power. Although there are several theoretical strategies for attacking Bitcoin like that, there is no evidence of them ever having been executed.

2.6 Problems in Public Blockchains

A major drawback of the PoW system is the large amounts of computational power it requires. All of this energy is constantly used and is essentially wasted. The Bitcoin network is rapidly expanding and the continuous electricity consumption is projected to be in between the production of a power plant and the consumption of a small country like Denmark by 2020 [19]. But even if this waste is considered to be worth it, there has to be an incentive for the nodes to provide computational power. In a system like Bitcoin, that is only designed to be currency, this incentivization is easy, but there are numerous applications for blockchain that do not involve currency.

Another problem caused by the PoW system is the limited transaction throughput. As the system is tuned to produce only one block (on average) in a certain interval of time, this interval and the block size dictate the transaction throughput [18]. This is a hard maximum that can cause transactions to pile up and essentially clog the blockchain. This maximum can only be increased by changing the specification of the distributed system. That process however is very tedious as can be seen with Bitcoin's block size discussion currently going on [3].

The need to have a full copy of the blockchain for most applications leads to large amounts of data every node has to hold. Currently the complete Bitcoin blockchain already takes up more than 130 Gigabytes of disk space [4]. Even though this can be considered affordable with respect to today's hardware, it also creates an ever rising entry barrier, because every new node joining the network has to download all blocks created so far.

Because a node in a public blockchain may have a full copy of the current blockchain, it has full access to all transactions. For this reason it is impossible to conduct a transaction without everyone else knowing. Even worse: from the transactions the current balance of any individual can be computed. Privacy is not possible in a public blockchain.

3. PRIVATE BLOCKCHAINS

A private blockchain is a blockchain that restricts read and write access using additional security systems. This restriction can be achieved by an access restricted blockchain deployed on a public network, as well as a standard blockchain deployed on a private network. This normally results in a lot fewer nodes being part of the network and typically all of these nodes can be authenticated. This type of blockchain is also referred to as *permissioned* being the contrast of a public, or permissionless blockchain.

3.1 Motivation

The research in the field of private blockchains is motivated by the aforementioned problems found in public blockchains. Private blockchains are mainly envisioned to be used as secure ledgers in and in between companies, but the possibilities are endless.

Blockchains remove an angle of attack on the system by getting rid of an all-powerful admin. Furthermore a blockchain provides better auditability than most current systems. Last, but not least, a blockchain might also save costs, because depending on the implementation of the private blockchain, there is no or just limited involvement of a third party. These aspects are so impactful that blockchain has the potential to forever change a broad range of transaction-based industries [12].

Privacy is required for mainly one reason. It is necessary in order to maintain common business practices like private contracts. Doing so also protects company data and therefore is essential in ensuring competitiveness.

3.2 Consensus

In order to understand why consensus in private distributed ledgers is so challenging, some exposition on consensus mechanisms in general has to be given first. There are mainly two approaches in existence. The first one is using a traditional byzantine fault tolerant algorithm like *Practical Byzantine Fault Tolerance* (PBFT) [16]. PBFT employs a voting system in order to decide on a block and is resilient against a maximum of 33% of the participating nodes being malicious. However, this system needs some central authority to manage network membership and the voting process does not scale well with large network sizes. Nevertheless PBFT is still a decent choice and for example was adopted for Hyperledger Fabric [7]. A new take on PBFT is the *Federated Byzantine Agreement* (FBA) [23]. FBA is fully decentralized and scales better. The central idea is that nodes do not rely on the majority of other nodes on the network for block acceptance any more, but on a majority of a set of nodes they trust. This mechanism is the one used by the financial service Stellar [14].

The other approach is a *leader election protocol* and also known as "Nakamoto consensus", named after the Bitcoin creator [15]. The idea of this protocol is to somehow randomly select a leader that may then create one block. Apart from Proof of Work discussed in section 2.3 the other notable, but far from being as widely used, representative of this approach is *Proof of Stake* (PoS). The general idea of PoS is that someone who owns a large part of a system is expected to naturally act in the interest of this system. Therefore a node's chance of creating the next block should be higher, the more value they control. The crypto-currency PPCoin [22] features an implementation of this mechanism. However, it does not rely on pure PoS and combines PoS with PoW, making mining easier for richer nodes. The feasibility of pure PoS is highly controversial.

All of these mechanisms are flawed in some way, especially when applied to a private blockchain. PBFT (and most other traditional byzantine fault tolerant) algorithms generally are a secure choice for small networks. However, they scale so badly (especially PBFT) that even private networks might quickly grow too large to achieve decent performance. Leader election protocols on the contrary are highly scalable, but PoW is the only one proven and extensively tested one. And while the PoW system still is immensely popular and used by a lot of big blockchain projects, it also has several severe weaknesses. The most important ones were already discussed in section 2.3 and following. When applied to a private blockchain these problems only augment.

The first problem is mining incentivization. This might be addressed by having decent transaction fees. But even if the stakeholders of a private blockchain would agree on a mining reward this would undermine one of the technology's major benefits: The costs of maintaining the blockchain would be tremendous.

The most critical problem relates to the network's size. The small amount of nodes in a private blockchain results in less combined computational power. This makes the system vulnerable to attack such as the ones presented in section 2.5, because one individual can gain over 50% of the network's computational power with relative ease. In terms of security this is catastrophic and renders the blockchain useless. By these reasons, other consensus mechanisms are needed in smaller, private networks.

4. HYPERLEDGER SAWTOOTH

Private blockchains in general are still very early early in development and next to no commercially used private blockchain systems are known at the time of writing (mid 2017). The most promising development in private blockchains so far probably is the *Hyperledger Project* [11]. This initiative was brought to life by the Linux Foundation in 2015 and is intended to form a community developing private blockchains together. The goal is to improve on existing blockchain technology in order to make it applicable for business. Prestigious members like IBM, Intel, SAP and many more, underline the huge amount of interest in this technology.

There are several different projects being developed and all of them are open source. This paper will focus on *Sawtooth* [10] (sometimes also referred to as Sawtooth Lake), because it seems to be the furthest in development and has the best specifications [6]. It is mainly written in Python and offers APIs in C++, Go, Java, Javascript, and Python. Sawtooth is able to support permissioned, as well as permissionless networks with a maximum of about 100 nodes. These networks can be deployed anywhere.

Table 4: Sawtooth Transaction Structure [8]

Name	Data Type
header	bytes
header_signature	string
payload	bytes

Table 5: Sawtooth Transaction Header Structure [8]

Name	Data Type
batcher_pubkey	string
dependencies	repeated string
family_name	string
family_version	string
inputs	repeated string
nonce	string
outputs	repeated string
payload_encoding	string
payload_512	string
signer_pubkey	string

Hyperledger Sawtooth was proposed to be included in the Hyperledger Project by Intel in April of 2016 [9] and is an active Hyperledger project at the time of writing (mid 2017). It is meant to be an enterprise solution for not only running, but also building and deploying (private) blockchains.

Nodes in Hyperledger Sawtooth can fill the roles of clients and/or validators. Clients are individuals interacting with the system by querying the state of the system or issuing transactions. Validators are nodes involved in the consensus algorithm and can be used for clients to request inclusion of transactions into the current blockchain. These validator nodes are comparable to miners in Bitcoin. However, they do use a different consensus mechanism that is presented in section 4.6;

Sawtooth's overall architecture is quite similar to Bitcoin's and will be presented in the following sections. The technical details were taken from the Hyperledger Sawtooth Documentation [8].

4.1 Transactions

Sawtooth features not only one type of transaction, but many different ones. They are grouped up into transaction families. The project comes with three already configured transaction families: `sawtooth_config`, `intkey` and `sawtooth_validator`. Developers are able to add more custom ones in order to tailor the system to every possible use case.

The `sawtooth_config` family allows for storing configuration settings on the chain. This is extremely powerful, as this enables users to configure the chain using the chain itself. For example this includes the transaction limit per block or

Table 6: Sawtooth Batch Structure [8]

Name	Data Type
header	bytes
header_signature	string
transactions	repeated Transaction

Table 7: Sawtooth Batch Header Structure [8]

Name	Data Type
signer_pubkey	string
transaction_ids	repeated string

the desired average time between blocks. This family also is special because it deviates from the standard consensus algorithm. Instead it has two options: Only one authorized node that is able to apply changes on it's own or a voting system for multiple authorized nodes with a configurable acceptance threshold. The second transaction family included is the `intkey` family. These transactions can be used to associate integer values with names and modify these values. This allows for easy creation of bitcoin-like blockchains for testing. The remaining set of pre-installed transactions is the `sawtooth_validator` family, which is responsible for adding new validators to the network.

A transaction in Sawtooth is composed of a header and a payload. Table 4 shows the structure of a transaction and Table 5 lists the transaction header fields. The `dependencies` field is useful for validators and lists the transactions that have to precede this one. The fields `family_name`, `family_version` and `payload_encoding` are providing information about the family (and version of that family) the transaction belongs to, as well as the family specific encoding used for the payload. The `nonce` field is just a random number used to differentiate between two transactions with the same content. The last header field of interest is the `batcher_pubkey`. This field ensures that a transaction can only ever be included in a batch by someone that was allowed to do so by the creator of the transaction. Batches are discussed in section 4.2. Security is provided by the `header_signature` found in the transaction. It can be validated using the signer's public key (`signer_pubkey` in the header) and the payload is bound to the transaction header via it's hash in the `payload_512` field.

The transaction payload consists of a list of key/value-pairs that are encoded according to the `payload_encoding` field. The number and type of these values are defined by the used transaction family.

4.2 Batches

Batches are the first major deviation from the Bitcoin blockchain presented in section 2. A batch groups up one or more transactions of any type. It is to be noted that batches do not replace blocks. Batches are a new component conceptually located in between transactions and blocks. The main benefit of a batch is that it can either be accepted or declined in all of it's entirety, but under no circumstances only part of the transactions in a batch may be accepted. This for example is useful in a scenario where two configuration changes are supposed two be made, where applying only one of these changes might have a catastrophic effect on the system.

Structurally they consist of a header and a list of transactions. This is shown in Table 6 and Table 7. The `signer_pubkey` allows to check whether all transactions were rightfully included into this batch. The transactions of the batch are bound to the batch header via their `transaction_ids`. The batch is secured by the `header_signature` that can be validated using the signer's public key (`signer_pubkey` in the header), just like a transaction is.

4.3 Blocks

One block in Sawtooth groups up transactions to be applied to the ledger just like it does in Bitcoin. The most important property to mention here is that the block does not group up the transactions directly. It only groups up batches. This means that every transaction has to be batched. If a transaction's acceptance is not supposed to depend on any other transaction(s) that still means that a batch has to be created for just this single transaction.

4.4 Global State

On top of simply storing the blocks, every Sawtooth validator also possesses a *radix merkle tree* in order to additionally store the current state of the blockchain. This again is a major difference in comparison to the Bitcoin blockchain. It has the benefit of being able to query the current state without having to compute it from all of the blocks first. This radix merkle tree is separately constructed by every validator and never shared via network.

Every piece of data stored in this tree has a unique address defined by the family and the piece of data. For example, the address that defines the maximum number of transactions per block is `sawtooth.validator.max_transactions_per_block`. The actual address used by the tree is a radix address derived from this address. These radix addresses are 35 bytes long. While the first 3 bytes of the radix address are the first 3 bytes of the `SHA256` hash of the namespace address, the other 32 bytes of the radix address have to be implemented specifically for every transaction family. At each node of the tree there is one byte labeling all outgoing edges. Following the bytes of a radix address leads to the leaf node containing the corresponding piece of data. This enables very efficient lookup operations.

Additionally the tree is also a merkle tree, meaning that all nodes can be hashed up into a single root hash. Each block header contains the root hash of the according tree. This links the blocks with the tree and assures that there also is consensus on the current state.

4.5 Journal

The Journal is the central component of every validator. It is highly modular to allow for different consensus algorithms. It is responsible for generating new blocks, validating candidate blocks received via network and evaluating every valid block in order to determine whether it should be the new chain head. It also stores all blocks of the current chain without any forks.

Whenever the Journal receives a batch it first waits for all dependencies to have been processed. Then it validates the batch and places it in storage where it waits to be included in a block. Periodically the Journal tries to build a new candidate block from the stored batches. If successful, the Journal waits until it is allowed to publish the block by the consensus algorithm (refer to section 4.6) and then proceeds to do so.

When the Journal receives a block, it is processed as soon as all predecessors are present. The following steps are taken: At first the root of the fork is determined. If the block simply extends the last block, the current chain head is the fork root. Then from this fork root on, all successors up until the new block are validated. In the end, it has to be decided whether the new block should become the new chain head. This decision is entirely dependent on the used consensus algorithm. If the new block is accepted as the new chain head, the block store is updated and all batches included in this block are removed from the pending batches.

The block validation process begins with a check for formal completeness. This includes existence of a valid predecessor as well as the presence of all batches on the block and the correct batch order. Next, all batches are validated. There may neither be duplicate batches, nor transactions and all of the transactions need to have valid dependencies. Then the block is validated according to consensus rules. Finally, the state root hash present in the block header is checked against the locally computed one.

Processing blocks inside the Journal is asynchronous. This improves performance while under great load.

4.6 Consensus using Proof of Elapsed Time

Because of all the problems PoW has, Intel decided to come up with their own leader election protocol: *Proof of Elapsed Time* (PoET) [27]. Although Sawtooth can theoretically support other consensus mechanisms, PoET is the most prominent (and currently only) one. The basic idea of this system is that every validator generates a random wait time for every new block it wants to create. The first validator who's timer expires is allowed to create the block. This way every CPU has the same chances of winning this lottery. Proof of Elapsed Time is a huge leap forward in blockchain technology, because it features all the properties of Proof of Work without the huge waste of power.

In order to guarantee the security of this system, it is necessary to be able to verify the correct execution of the timer process. This is realized by using Intel's *Software Guard Extension* (SGX) [13] [17]. Since the Skylake generation of Intel processors, this feature is becoming available on more and more Intel processors. SGX runs programs in protected areas of memory called enclaves. In this enclave the correct execution of a program can be cryptographically attestated and neither the user, nor the operating system is able to read or manipulate the enclave. Even though the initialization of an enclave is performed by untrusted software, it is possible to verify that the enclave was initialized properly. A *measurement hash* generated during initialization can prove that the enclave was set up correctly.

The backbone of this system's security is that a remote party can verify the integrity of an enclave. Every chip manufactured by Intel receives two secrets into the chip's one-time programmable registers: The *Provisioning Secret* and the *Seal Secret*. While the first one is known to Intel's *Provisioning Service*, the latter one is generated on the chip and not known by anyone. This is beneficial should Intel ever be the target of a successful hack, because the Seal Secret is used to derive a key that is used for securely storing other used keys in the system. With a *Provisioning Key*, generated from the Provisioning Secret, the enclave's certificate-based identity (created by the enclave's author) and a security version number, the *Provisioning Enclave* can request an *Attestation Key* from the Provisioning Service. After provisioning the local SGX, the *Quoting Enclave* can be used for software attestation. It receives a local attestation report from the enclave running PoET and signs it with the Attestation Key. The signed *attestation* (also called quote) is the proof for a newly created block. This whole system is completed by the *Intel Attestation Service* (IAS) [1], a web service that verifies the attestation issued by a node's enclave.

However the PoET system is still experimental and has not yet been fully implemented. Documentation is also lacking to non-existent, making it very hard to go into detail. At the time of writing this paper (mid 2017) only a test implementation exists that is not safe for use in a production environment, because it only uses a simulated enclave. All details mentioned here should be taken with a grain of salt.

4.7 Problems with Proof of Elapsed Time

PoET in the currently envisioned version requires several Intel services. This can be seen as a setback to the completely decentralized nature of a blockchain, as Intel is acting as a central authority and every node has to trust them.

Another disadvantage introduced by the IAS is an increase in difficulty of verifying blocks in comparison to PoW. While the latter one only requires computing a hash and comparing it, this new one needs to query the IAS. This adds a full round trip time to every verification.

The SGX is theoretically breakable with a realistic amount of effort, resources and knowledge. The result is catastrophic because with PoET a single broken node is able to perform majority attacks on the blockchain, such as the ones discussed in section 2.5. Intel is working on detecting compromised nodes through statistical analysis, but the attainable effectiveness remains to be seen.

Further problems and vulnerabilities may become apparent once PoET has been fully implemented and thoroughly tested. Before that it will not be possible to accurately assess the suitability of PoET as a distributed consensus mechanism.

5. RELATED WORK

Apart from Hyperledger Sawtooth, that was presented in section 4 of this paper, there are some other development efforts in the field of private blockchains worth mentioning.

Hyperledger Fabric [7] also is a part of the Hyperledger Project. Fabric was proposed for Hyperledger in early 2016 by IBM and Digital Asset. It provides a foundation for developing permissioned blockchains using PBFT consensus.

Ripple and *Stellar* are two big financial service providers relying on their own permissioned blockchains. They connect financial institutions like banks with their network, allowing for fast and secure transactions in between them. These transactions use their respective crypto-currency to make simple currency exchange possible. The big difference in between these two is, that Stellar is a nonprofit organization, while Ripple is not. In terms of technology, Ripple uses a custom voting algorithm called *Ripple Protocol Consensus Algorithm* (RPCA) [26] and Stellar employs FBA [23].

6. CONCLUSION

This paper gave an introduction to blockchains using the example of Bitcoin. It was shown that PoW, being the most used public consensus algorithm, is effective at securing a distributed ledger if the network is large enough. Yet it is far from efficient due to the enormous power consumption introduced by mining. It was further argued that PoW simply is not applicable for small networks, such as private blockchain systems, because they extremely facilitate 50%-attacks.

Hyperledger Sawtooth by Intel was found to be a promising project to become a breakthrough in the field of private blockchains. The most important innovation is the new leader election protocol in development called Proof of Elapsed Time. It theoretically provides the same benefits as PoW does, but is applicable to smaller networks and does not nearly consume as much power. All of this comes at a price though: Because of it's reliance on Intel's trusted execution environment SGX, the algorithm requires nodes to place at least some trust in Intel. Also the block verification is slightly more complex, because it requires a query to the IAS.

Even though PoET is showing a lot of potential, it is far too soon for a final verdict. The actual viability remains to be seen once Sawtooth is production ready.

7. REFERENCES

[1] Attestation Service for Intel Software Guard Extensions (Intel SGX): API Documentation. `https://www.hyperledger.org/projects/sawtooth`, retrieved June 2017.

[2] Bitcoin Developer Reference. `https://bitcoin.org/en/developer-reference`, retrieved June 2017.

[3] Bitcoin Wiki: Block size limit controversy. `https://en.bitcoin.it/wiki/Block_size_limit_controversy`, retrieved June 2017.

[4] Coin Dance: Bitcoin Statistics. `https://coin.dance/stats#blockchain`, retrieved July 2017.

[5] Ethereum Website. `https://www.ethereum.org/`, retrieved June 2017.

[6] Hyperledger Comparison Guide. `https://github.com/hyperledger/hyperledger/blob/master/hyperledger_Overview_Comparison_Guide.md`, retrieved June 2017.

[7] Hyperledger Fabric Website. `https://www.hyperledger.org/projects/fabric`, retrieved June 2017.

[8] Hyperledger Sawtooth Documentation. `http://intelledger.github.io/`, retrieved June 2017.

[9] Hyperledger Sawtooth Proposal. `https://docs.google.com/document/d/1j7YcGLJH6LkzvWdOYFIt2kpkVlLEmILErXL6t-Ky2zU/edit`, retrieved June 2017.

[10] Hyperledger Sawtooth Website. `https://www.hyperledger.org/projects/sawtooth`, retrieved June 2017.

[11] Hyperledger Website. `https://www.hyperledger.org/`, retrieved June 2017.

[12] Hyperledger Website: About. `https://www.hyperledger.org/about`, retrieved June 2017.

[13] Intel SGX Developer Zone. `https://software.intel.com/en-us/sgx`, retrieved June 2017.

[14] Stellar Website. `https://www.stellar.org/`, retrieved June 2017.

[15] M. Ali and J. Nelson. Blockstack: A Global Naming and Storage System Secured by Blockchains, 2016.

[16] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance, 1999. `https://www.usenix.org/legacy/events/osdi99/full_papers/castro/castro_html/castro.html`, retrieved June 2017.

[17] V. Costan and S. Devadas. Intel SGX Explained, 2016.

[18] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juelsand, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On Scaling Decentralized Blockchains, 2016.

[19] S. Deetman. Bitcoin could consume as much electricity as Denmark by 2020, March 2016. `https://web.archive.org/web/20160828092858/http://motherboard.vice.com/read/bitcoin-could-consume-as-much-electricity-as-denmark-by-2020`, retrieved June 2017.

[20] I. Eyal and E. G. Sirer. Majority is not Enough: Bitcoin Mining is Vulnerable, 2013.

[21] G. O. Karame, E. Androulaki, and S. Capkun. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin, 2012.

[22] S. King and S. Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake, 2012.

[23] D. MaziÃÍres. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, 2015.

[24] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[25] M. Rosenfeld. Analysis of hashrate-based double-spending, 2012.

[26] D. Schwartz, N. Youngs, and A. Britto. The Ripple Protocol Consensus Algorithm, 2014.

[27] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse. REM: Resource-Efficient Mining for Blockchains, 2017.