Resource efficient peer-to-peer time-stamping service

Gísli Hjálmtýsson
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
Email: gisli@ru.edu

Steinar Sigurðsson
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
Email: steinars17@ru.is

***Abstract*—Ushered in by the introduction of Bitcoin, blockchain systems are attracting interest from business and academia. The dominant blockchains, Bitcoin and Ethereum, introduce token coins, whose introduction simultaneously is the goal of each system, requires token-coin functionality, and is integral to the probabilistic correctness of each system. Both systems use proof-of-work as a consensus mechanism with unrestricted membership of writers, resulting in increasing resource use that is becoming unsustainable.**

We see the primary value of a blockchain system as being the creation of a totally ordered sequence of events in a distributed system, that is immutable and third party verifiable. We propose that all other functions, are not seminal to the function of the blockchain system and should be implemented as applications enabled from this basic functionality. On the contrary, that including functionality beyond what is essential to achieve this primary goal, introduces complexity beyond value, threatens scalability and is ultimately detrimental to the overall system performance and usefulness.

In this paper we propose a lightweight peer-to-peer permitted blockchain system, having only the minimum functionality to ensure immutability, and third-party verifiability. We achieve this with minimal resources without proof-of-work or token coins. We show how this system provides high transaction rate while supporting a number of interesting applications.

1. Introduction

Ushered in by the introduction of Bitcoin, blockchain systems are attracting interest from business and academia. However, the dominant blockchains, Bitcoin, Ethereum, and their derivatives [1], [2], suffer from complexity and scalability problems. Designed for the purpose of doing peer-to-peer monetary transactions, an integral part of the Bitcoin blockchain is the manipulation of electronic wallets and electronic "cash" management. Ethereum elaborates on the token coin functionality by supporting smart contracts further adding to the complexity. The introduction of a token coin implies a monetary issuance policy, in each case implemented via a reward scheme to the writers of the blockchain. The respective token coins are moreover integral to the probabilistic correctness of each system. None of this functionality or associated complexity seems necessary to the underlying blockchain.

Moreover, consensus based on proof-of-work with an unrestricted writer set has resulted in rapidly increasing resource use that is becoming unsustainable. Originally proposed as a distributed mechanism to combat DOS by throttling service request rate [3], proof-of-work requires fixed state at each writer (miner), independent of the size of the writer set, and no explicit coordination beyond the announcement of the next block. The trade-off cost is substantial resource use, slow consensus and low transaction rate, limiting the scalability of the system. Combined with unlimited writer set, the increasing resource use is unsustainable.

Unlike in typical networks, where each node adds to the overall value of the system, doubling the number of miners does *not* increase the inherent value of the overall proof-of-work based token coin blockchain system, a clear manifestation of the tragedy of the commons. Acknowledging this the Ethereum community is proposing to move to a proof-of-stake based consensus, which includes a proposal to actively manage the size of the writer group (called the validator pool) [4] and in fact active monitoring and policing of writer behavior and compliance.

We propose that the essential value of any blockchain system is the creation of a ledger recording the chronological order of events in a distributed system, that is immutable and third party verifiable. Furthermore, that all other functions of interest are in essence applications enabled by this basic functionality and not essential to the function of the blockchain system. On the contrary, we contend that including functionality beyond what is essential to provide these two seminal properties, introduces complexity beyond value, threatens scalability and is ultimately detrimental to the overall system performance and usefulness.

We consider permissioned blockchains only, where a certain set of writers have the exclusive right to add to the chain. Additional members of the system can have read access to the system. Whereas, similar to [4], we don't put any hard upper limit on the size of the set of writers, the efficiency of our solution is negatively affected

as this writer set grows without bounds. It would seem that an unrestricted writer set will ultimately suffer from the tragedy of the commons.

Regardless of size of writer set, we observe that even if only elementary measures are taken to manage the writers permitted, failures of writers occur at timescales several orders of magnitude that of the timescale of transactions. Moreover, failures in a given round are not randomly distributed independent of previous rounds. The implication is that in all but pathological cases the cost of discovering the set of correctly working nodes is comparatively small. Persistently malfunctioning nodes (or even underperforming) may be removed by a slow-timescale writer group management protocol - potentially a human administrator. Of course the protocol must cope with failures occurring, but can be optimized for the common case.

Similar to other blockchains, we incorporate a mechanism for nodes to be rewarded for contributing to the construction of the chain, and consequently malfunctioning nodes to be penalized. However, rather than obfuscating the protocol with a new cryptocurrency, the chain records the identity of the writer of each new block, thus becoming a record of contribution suitable for any type of rewarding mechanism.

In this paper we propose a new light-weight permitted blockchain system, having only the minimum functionality to ensure immutability, and third-party verifiability. The principal contribution of this paper is scalable resource efficient peer-to-peer time-stamping service based on this blockchain system. We further introduce a novel lightweight consensus protocol supporting very high transaction rate. Lastly, our analysis verifies the properties of the consensus protocol, complemented with simulation results to demonstrating its robustness and performance. Lastly, we discuss a number of applications of interest.

The rest of this paper is organized as follows. In Section 2 we discuss related work. In Section 3, we formulate the discussion, before describing our novel blockchain system in Section 4. In Section 5, discusses failures and cancelled rounds. Analysis and modeling of properties and performance are detailed in Section 6. In Section 7 we describes interesting applications using the blockchain system. We then conclude in Section 8.

2. Related work

In their 1999 paper, H. Massias et al [5] describe a "secure time-stamping service with minimal trust requirements", where a trusted third party issues timestamps upon client requests. Our basic immutable third-party verifiable event log is based on this work, albeit such that no single writer is trusted and is replaced by a peer-to-peer network of writers.

Whereas [1], [5] describe mechanisms to pack multiple client requests into a single block, any such mechanisms are orthogonal to our blockchain.

The canonical problem of distributed consensus is the Byzantine generals problem. Since this problem was introduced almost 40 years ago [6] a slew of papers have focused on various aspects and solutions [7], [8]. The Byzantine generals problem has been shown to be tolerant to up to a third of the nodes being faulty or malicious.

The Byzantine generals problem is formulated to reach a consensus *one time* and hence the timescale of the single consensus is the same as the timescale of failures. The correctness criteria of a blockchain is weaker. The chain is a long log of events where blocks are only added to the chain if a consensus is reached. In our case a failure of even one writer prevents consensus of a given round. However, this does not impact correctness as the round is cancelled (see Section 5), and merely reduces the overall performance in terms of transaction throughput. We show that under practical assumptions this impact is small.

Clearly, the most influential publication on blockchain is the "Satoshi" paper introducing Bitcoin [1]. Similar to [1] we base our immutable event log on [5]. However, rather than introducing PoW to select the winner of each round, we take a protocol approach, avoiding the resource consumption of hashing altogether. Whereas the effort and cost of hashing may potentially be justified for Bitcoin as the Bitcoin and its use is *the* proposed value of the Bitcoin chain, the growing resource consumption seems unsustainable and to threaten the viability of the Bitcoin blockchain. In contrast, our resource-light protocol focuses on the value embedded in the body of each block, the immutability, and the total order logged by the chain.

The Bitcoin blockchain is created for one application only, namely digital cash[1]. Ethereum [2] adds smart-contract functionality, whereby each transaction implements an arbitrary functionality, executed by the blockchain system. This introduces substantial complexity into Ethereum system and unpredictability in the execution of transactions. Adding to the chain is the *critical section* of the underlying distributed system. It seems contrary to the learning of the operating systems and distributed systems community to introduce substantial complexity and unpredictability into the critical section.

From the perspective of this research, smart contracts and their execution, while interesting, should be carried out separately from the underlying blockchain system.

This view resonates with the approach taken by the IBM Hyperledger [9]. IBM promotes the Hyperledger as a distributed operating system based on blockchain, with a very generic model for smart-contracts, supporting in principle smart-contracts based on any programming language. However, although the execution of smart contracts is the responsibility of the Hyperledger system, their execution is part of preparing a proposed transaction, before such transaction is submitted for addition to the blockchain, and therefore executed outside of the critical-section where the block is added.

Our approach takes one step further, pushing the management and execution of smart-contracts further out,

1. We acknowledge that users are using it for other applications including smart contracts.

and completely out of the blockchain system. Whereas we see substantial potential in the ongoing research on smart contract, we propose that a smart-contract system be built on top of our lightweight blockchain, as an application.

Following the success of Bitcoin, a number of cryptocurrencies have been launched, the majority of which use the proof-of-work consensus algorithm [10]. To combat the resource consumption of proof-of-work, a proof-of-stake (PoS) system have been introduced by some blockchains [4], [11], [12].

The notion of implementing blockchains in the Internet of Things (IoT) has been divisive among developers. The combination of the two can be powerful as blockchains bring a lot of advantages to the table, especially their decentralization and immutability.

Christidis et al [13] examined the use of blockchain in IoT systems, finding that blockchains to provide low transaction processing throughput and high latency, as compared to regular DBs. We see this as a promising potential use for our lightweight blockchain.

3. The blockchain system

We define a blockchain system as a set of nodes with the common objective of writing and verifying a single totally ordered chain of transaction blocks.

A subset of the nodes are *writers* that are allowed to write new blocks to the chain. Writers participate in a (distributed) consensus algorithm to determine a winner, who gets to write the next block. The set of writers W may change dynamically, although the changes to the membership are assumed occur at timescales several orders of magnitude larger than that of transactions. Nodes that are not writers are either clients, generating transaction requests, or observers, all of which can read the entire chain, and verify that the chain is validly construed. Writers communicate directly over the Internet, essentially forming a peer-to-peer network. Some messages may be lost in this peer-to-peer network.

Nodes may be rewarded for writing to the block[2] or otherwise desire to write to the block but may misbehave to gain an advantage, or attempt to rewrite part of the chain. Clearly, nodes can fail and may maliciously attempt to game the system, or simply prevent it from functioning.

In what has become referred to as public chains, including the Bitcoin block-chain, all nodes are potential writers and equal with respect to their privileges and participate equally in the consensus protocol.

3.1. Generalized permitted chains

We define a permitted chain to a be a block-chain system where the set of writers W consists of nodes that are explicitly *permitted* into the set of writers. The changes in the membership are thus governed by some additional protocol and policy. Whereas this introduces some complexity in terms of managing the membership of writers, we observe, that the time scale of such management is orders of magnitude that of the processing transactions. For those permitted chains that we know of, this management is off-line. No assumptions are made about the number of writers or their distribution. A permitted chain only limits the privileges of writers, but is publicly open to any client or observer.

2. Not necessarily with direct monetary reward, but could be any type of recognition

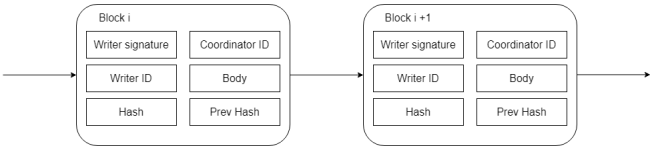

Figure 1. An immutable chain of events

4. The lightweight Blockchain system

In this section we describe the three main mechanisms of our blockchain system. First, an elementary event log built on a peer-to-peer time-stamping service. Second, we define and describe the lightweight decentralized consensus protocol. Third, the management and monitoring of the group of writers.

4.1. A simple light-weight event log

Our event log is essentially a timestamp server, similar to that described in [5] and adopted for Bitcoin. Each entry in the log consists of, i) the id of the coordinator, ii) the id of the winner, iii) the body, iv) the hash of the block, v) the hash of the previous block, and vi) the signature of the writer of the block (i.e. the winner). The critical element is that the hash of the block includes the hash of the previous entry in its hash. By induction the hash value of the new block is a function of all previous blocks, providing the immutability of the chain with respect to the set of writers. The body may be composed of multiple transactions that may be further organized e.g. into Merkle trees [14]. Such optimizations are not the topic of this paper.

The chain is immutable as no previous block cannot be changed without the participation of *ALL* members of the writer group. Each block is signed cryptographically by the winner who writes it. Moreover, any rewrite could be cancelled by any writer in the group. Most importantly, such rewriting would be clear and obviously detected by any of the writers. Thus, **our time-stamping service is peer-to-peer without proof of work**.

Milestoning and immutability. To further make the event log immutable beyond the set of active writers, periodically a milestone event is added to the chain, and publicized widely. The first milestone event is the genesis block. For publication any persistent publicly accessible forum will do, for example a Usenet or a Facebook post. For our

implementation, we add a milestone event for every 100K events, by writing the milestone event into the Bitcoin blockchain.

Publicizing the event seals the immutability of the chain. Even if the chain is private and only accessible to a limited set of readers outside the writer group, for example external auditors, the public milestones serve as external anchors to the chain. Even if the chain exists for a limited time, and the set of writers is retired, the last milestone serves as the end of chain, which again cannot be altered thereafter without it being detectable.

4.2. A lightweight high-performance consensus protocol

For a high level view of the underlying idea of the lightweight decentralized consensus protocol, we select a leader based on the following. Each writer has a private cryptographically secure one-time pad (OTP) [15]. In each round, each writer produces the next number from its OTP. From all the numbers we calculate an aggregate result, i.e. a generalized "sum". The writer whose number is closest to the aggregate is declared the leader.

The consensus protocol. To formalize the above idea into a protocol, we define a set W set of writers, and start by considering a single round. First we enumerate the set of writers, from 0 to $||W|| - 1$. For each round a *coordinator* is selected (described below), to coordinate the round. The Coordinator does not participate in the round as a contender to win. All arithmetic is performed modulus some large N. The round is in four steps:

- Step 1: The Coordinator broadcasts to all other writers in W a request for the next number.
- Step 2: Each writer, generates the next number from its local OTP, and responds by transmitting it to the Coordinator.
- Step 3: The Coordinator, aggregates the numbers into a result r, and determines the winner, w, as the identity of the writer whose number is "closest" to r. The Coordinator then broadcasts to all writers in W, the result r, the winner w, and all numbers received in Step 2.
- Step 4: Each writer, verifies the round by confirming i) that its number is correctly received and reported, ii) the calculation of r, and thus iii) the winner w. The winner broadcasts its winning block, cryptographically signed. All others, if all is correct, reply with a confirm to the coordinator. Otherwise, a dissenting writer *broadcasts* a reject.

For the initial round the writer with the lowest number acts as the round coordinator. For subsequent rounds we have used a simple round robin.

To aggregate the number from all the writers, we could in principle use a simple summation modulus N. For its simplicity when working with large numbers (e.g. multiple words) and for its cryptographic properties, we have chosen to use XOR as the aggregation function.

The distance metric is more tricky. We can think off, and have experimented with a number of distance metrics, including i) the arithmetic difference, ii) the Hamming distance, iii) the value of the result XOR-ed to each value, i.e. $r \oplus s_i$, where s_i is the number generated by writer w_i.

To break ties we have used the enumerated identity of w. Whereas, more elaborate tie breaking exist, with N large enough, the likelihood of ties is small.

Complexity. The message complexity is $||W|| - 1$ messages of fixed size for each step one and two. For step three $||W||$ messages each of size $||W||$. For step four, if no writer dissents, $||W|| - 2$ messages of fixed size are sent to the coordinator, plus $||W|| - 1$ messages from the winner to transmit the new block to all. Hence for successful rounds the number of messages becomes $o(||W||)$.

If the round fails, each dissenting writer broadcast a message, in which case the complexity becomes $c * ||W||$, where c is the number of cancelling nodes. To avoid this last broadcast, the protocol can be augmented by adding one extra round of messages, where the coordinator broadcasts a cancel if one is received. However, in the situations that we are targeting, we foresee that c to be too small for this to affect the overall performance of the protocol.

Another potential issue is the the size of the message in Step 3 is $||W||$. For the numbers we envision for the set of writers, e.g. no more than few hundreds, it is not an issue. For larger sets, without affecting the correctness of the protocol, the coordinator may omit sending the numbers received from all and simply announce the result by broadcasting the result and the identity of the winner.

The space complexity at each writer is linear in the size of the writer set.

To prevent any bias and reduce the dependency of the coordinator, the protocol can be elaborated by using two or more coordinators, which all play the role of the coordinator and do not participate in the protocol to be selected as potential winners. With multiple coordinators, any mismatch between coordinators would render such round cancelled.

Alternatively, a monitor may be nominated, either as a separate node, or as a rolling function among the writer set, allowing the monitoring of biases, collusion and other misbehavior, ruling out misbehaving nodes. For a number of reasons, including debugging, in our work we include such a monitor as a matter of course.

Correctness. The purpose of the consensus algorithm is in each round to select the writer w that becomes the leader, i.e. gets to write to the chain. Each process draws from its OTP an initial value. The goal of the protocol is that all valid writers decide on a value v. The problem can be formally defined in terms of three properties:

- Validity: If all correct writers propose the same value w, then any correct process that decides, decides w.

- Agreement: No two correct writers decide differently.
- Termination: Every correct process eventually decides.

The first two properties ensure correctness, whereas the last ensures termination [16].

The value w is broadcast in Step 3, and verified by each writer in Step 4. If the coordinator is faulty it may select a writer $\vec{v}$. In any case, if a writer does not concur with the result, regardless of the reason, then such a writer will trigger a cancel. Hence, if no cancel occurs, all processes in the writer set (including the coordinator) will decide on the same value w, thus establishing correctness. Clearly, every correct process will eventually respond, and thus decide.

Performance enhancements. It is clear that the latency (from beginning to end) of the protocol is two round-trips. However to enhance the throughput, it is worth observing that the next round does not need to await the completion of the previous round. Rather, the next coordinator in line, having generated its response and completed Step 2, of round i, immediately initiates round $i+1$, and so on. The result is that the latency from beginning of one round to the beginning of the next is the *one way* latency in the network of writers.

To increase the transaction rate further, one can adopt a similar approach to for example the Bitcoin blockchain, where multiple transactions are written in each block, effectively scaling the transaction rate by a constant. Alternatively, in each round, rather than generating a single number, each writer can generate k next numbers from the OTP, resulting in k next winners being selected in each round.

4.3. Managing the set of writers

We consider the membership and management of the groups of writers.

To create a new lightweight blockchain, a creator prepares a configuration containing a list of permitted participants and distributes it to the participants as a way of invitation. One easy way to do this is to simply publish the list at a forum known and accessible to the participants, e.g. on the web. Each participant retrieves the list of invitees, verifies their identities and public keys, and joins the peer-to-peer network of writers by initiating a session with one or more of the participants. Participants may be designated as writers, readers, clients or monitors. The configuration may include additional properties, such as number of coordinators per round and more. The blockchain may be open to readers and/or clients. In that case anyone will be admitted for read access, and/or as a new client issuing requests.

Each writer maintains a state for every writer in the writing set. This state includes, the writers id, and public key, but also the protocol state of the writer, either active or penalty. In addition writers may keep track of statistics of other writers and more.

The basic protocol deals with occasional message losses as message losses will either result in timeout and subsequent retransmission, or will simply trigger a cancel. For smaller writer sets, we have implemented the peer-to-peer networks as a fully connected mesh, with reliable delivery (TCP).

Connectivity of the peer-to-peer network connecting the group of writers is maintained separately from the consensus protocol and blockchain management. Using a relatively simple echo-reply style protocol, disconnected nodes can be identified. At the protocol level, such nodes are treated as being in the penalty box.

5. Failures and cancelled rounds

A round fails if any of the writers fails to confirm the consensus (in step 4). This occurs if for some reason the verification step fails, if for any reason a writer fails to respond (time-out), or otherwise fails. Our failure model and operating assumption make writer failures unlikely, however when a writer fails, it will be in a failed state for a number of rounds. To reduce the impact of failures on the overall throughput of the system, a failed writer is put into a penalty box (i.e. a suspend list) where it remains for some amount of time until it is allowed back as an active writer.

When a player faults (whatever the reason), it enters the penalty box initially for four rounds. If the writer is still faulty, when the penalty expires, the penalty time is doubled, resulting in exponentially increasing penalty time, thereby limiting the number of rounds affected by a failure of a particular writer. For practical reasons, we implement a maximum penalty time.

If a coordinator is faulty, it will be observed by multiple writers, each of which broadcasts a cancel message. Thus the round is cancelled, and the protocol moves on.

There is a possibility that a coordinator may collude with one of the writers, by collecting numbers from all other writers before selecting a winning number for its partner. A rewarding scheme that rewards failure free participation above winning, essentially eliminates the incentive to collude this way. Monitoring the set of writers and penalizing misbehaving nodes e.g. similar to that discussed in [4], is another alternative.

6. Analysis and Modeling

In this section we analyze some of the properties of the protocol and model the throughput performance. We start with formalizing the following definition of the sequence of winners.

6.1. A distributed one-time-pad

Consider Z_n the set of non-negative integers less than n. Let $\vec{\sigma_i}$ be a sequence, $\sigma_{i,0}, \sigma_{i,1}, \sigma_{i,2}, ...$ such that $\sigma_{i,t}$ is

in Z_n for all $t \geq 0$, and where $\vec{\sigma_i}$ is the sequence generated by writer i. We say that $\sigma_{i,t}$ is the number generated by node i at round t.

Let $s_t = \sum s_{j,t}$ in $Z_n, \forall j$, and let $\vec{s}$ be the sequence of s_t for $t \geq 0$. Whether Σ denotes the arithmetic sum or summation by XOR it follows by induction that $\vec{s}$ satisfies the following properties:

- $\vec{s}$ is a uniquely defined.
- if for any i the sequence $\vec{\sigma_i}$ is *cryptographically safe*, then so is the sequence s_t

It is interesting to contemplate the ability of node i to predict $\vec{s}$ and use it to its advantage. Starting with the case when $||W|| = 2$ it is clear that by making a uniform random guess, node 0 always has at least $1/n$ probability of guessing node 1's number. Conversely, if node 1 selects the next number randomly and uniformly in Z_n, then clearly node 0 has at best $1/n$ probability of guessing that number. Hence neither node can do better.

It is elementary to show, that for uniform random variables $X, Y \in U(0, n)$ over Z_n their sum $Z = X + Y \mod n$ is a uniformly distributed random number, i.e. $Z \in U(0, n)$. By induction this applies for sum of any number of such variables.

It follows that node i's ability to predict the sum is minimized if all other nodes issue number sequences that are random and uniformly distributed over Z_n. Hence any writer wishing to maximize its likelihood of becoming a winner will adopt the game strategy of selecting its number at random uniformly in $[0..Z_n]$.

6.2. Modelling of failed writers

To model the failures of writers, we use a two-phased *Modulated Markov Process* (MMP). In this model each writer is considered to be in an active phase until it fails (for whatever reason), and moves to a failed state for some time, until it recovers and returns to being active. With $state - 0$ denoting active, and $state - 1$ denote failed, the transition matrix of this two-phase on/off model is

$$P = \begin{pmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & p_{1,1} \end{pmatrix}, \sum_{j=0}^{1} p_{i,j} = 1, i = 0, 1.$$

Given the two independent parameters $p_{0,0}$ and $p_{1,1}$, the probability that the system is in $state - i$, is

$$\pi_i = \begin{cases} \frac{p_{1,0}}{p_{1,0}+p_{0,1}} & \text{if} \quad i = 0 \\ \frac{p_{0,1}}{p_{1,0}+p_{0,1}} & \text{if} \quad i = 1 \end{cases}$$

and thus the probability that a given writer is active is $\mu = \pi_0$. The expected sojourn time as in $state - i$ is $E_i\{L\} = 1/(1 - p_{i,i})$, with variance $Var_i\{L\} = p_{i,i}/(1 - p(i, i))^2$.

We use this model to model the failures of individual writers, failing independently.

6.3. Simulating transaction throughput

We evaluate the performance of the lightweight blockchain system by studying the transaction throughput. As a failure model we use the two-phased model above. Ours is a permitted chain, and hence the writer set is selected, and thus reasonably assumed to be composed of computers receiving some minimum monitoring and maintenance. A permitted chain is by definition managed, however minimally that may be.

Using the on-off model, there are three controlling parameters for our simulations, i) the size of the writer set, ii) the expected up-time of each writer, iii) the duration of failures. We simulate for a range of values for the size of the writer set, but are primarily interested in small to medium sized writer sets. For our simulations we assume all writers have the same fail parameters for both expected up-time and duration of failures. The duration of failures captures whether failures are frequent but minor, or they are scheduled downtime or failures that require human intervention. The latter will occur on timescales of minutes, hours or days. Hence, there really is a fourth parameter, namely the rate of transaction rounds per second. We effectively embed this parameter, by setting the duration of failures in terms of number of rounds.

In the following section we show simulation results, for effective throughput (number of completed transaction vs failed), as the controlling parameters are varied. We moreover show how the occupancy of the penalty box is impacted by the same.

6.4. Simulation Results

Table 1 shows the number of failed rounds as the size of the writer set varies. The average up-time is set to 99.5%. Each column shows the results for a different value for the expected duration of failures. Faulty writers enter the penalty box as described above. We run each simulation for 100 million rounds. At two transaction rounds per second, 100 million rounds correspond to just over 19 months; at 5 transaction rounds per second, 100 million rounds correspond to just over 7,5 months or 32 weeks. The three columns show failed state duration of 1.800, 7.200, and 28.800 rounds, corresponding to 15 minutes, one hour and four hours until repair respectively, at the rate of two transaction rounds per second. Results shown are mean and standard deviation for sixteen runs each.

At a fixed up-time, the longer the duration of failures, the fewer new failures occur. A writer failure causes a round to be canceled, and again each time the penalty expires while still in failed state. However, due to the exponential backup the impact is limited to the first few rounds. As a consequence the number of failed rounds decreases with longer duration of failures. This is clearly visible in the numbers. However, equally visible is the dramatic impact of the exponential backup as the fraction of canceled rounds is minuscule even for short failed times. As would be expected the number of rounds cancelled

$\|\|W\|\|$	1800		7200		28800	
2	5040	± 211	1587	± 136.1	431	± 86
5	12534	± 200	3758	± 218	1107	± 99
10	24948	± 330	7694	± 365	2324	± 146
20	50167	± 507	15361	± 402	4547	± 139
40	100570	± 260	30444	± 571	8959	± 383
100	251062	± 995	76541	± 905	22736	± 560
200	500955	± 1892	151983	± 1244	45158	± 624
400	996612	± 3925	305795	± 1892	90470	± 691

TABLE 1. CANCELED ROUNDS AS THE SIZE OF THE WRITER SET VARIES, FOR THREE VALUES FOR DOWNTIME DURATION.

Number	$\|\|W\|\| = 40$	$\|\|W\|\| = 100$
0	80.615.517	60.532.563
1	16.527.002	25.926.021
2	2.561.942	10.075.722
3	265.803	2.779.696
4	27.984	575.466
5	1.749	96.829
6		13.484
7		216

TABLE 2. FREQUENCY TABLE OF NUMBER OF WRITERS IN THE PENALTY BOX

increases with a larger writer set. Still even with a writer set of size 400, the success rate of the system with a 4 hour burst of failures is >99.9%. We conclude, that our simple light-weight blockchain exhibits superior throughput performance which is mostly unaffected by such failures.

Table 2 shows the frequency table for number of writers in the penalty box averaged over 16 runs of 100 million events, for two writer sets, size 40 and 100 respectively. The simulation is for four hour repair time for failures, with average up-time of 99.5%. As expected the penalty box is slightly more populated with the larger writer set. However, in over 95% of the time, two or less writers are being penalized. We conclude as before that the impact of failures on the overall performance is negligible.

7. Applications of the blockchain system

We have experimented with a number of applications enabled by the basic functionality of the light-weight blockchain system. Three of them are briefly discussed below.

7.1. Digital Certification Service

A Digital Certification Service in its purest form is essentially a time-stamping service, merely evidencing that the digital object written into the chain existed at a certain time relative to other events on the chain. Adding a (local) timestamp to the body, with the writers participating in the Network Time Protocol [17], provides absolute time of events correct within tens or at most hundreds of milliseconds. For usability, additional information may be collected, written to the body of block, and/or stored in an affiliated database system.

We have implemented a certification service for academic transcripts and diplomas, as a web service. A client submits a document (using the web services API, or manually using a web front end). The service computes a hash of the document and writes it into the blockchain. A reference "ticket" in the form of a QR code is returned to the client. Any third party, having a read access to the chain, can read the chain (using the ticket to speed up its search), compute the document hash and verify when and where it was written in the chain.

7.2. Traceability service

We define a traceability service, as the ability to track events related to a named object. We employ a hierarchical naming system, allowing a named object to be broken into parts. In addition some service specific attributes are written to the body. The motivation is to be able to trace objects (and/or their electronic twins) from the source, with applications in food, news, data, and even decisions.

Apart from the naming scheme, the traceability service builds on and uses the digital certification service. An observer verifies the source by selecting all the events matching the name, identifying the first such event in the log.

7.3. e-Wallets

We define an e-wallet as a named object having *(attribute,value)* pairs, whose current value is stored in the value and address is stored in the attribute. The application is built upon the traceability service where the e-wallet traces all transactions to and from the wallet's address in the blockchain. A transaction is defined as the wallets owner transferring funds from its e-wallet to another address and cryptographically signing the transaction to prevent spoofing.

This application can be implemented on the client side or as a web service, storing the data in the cloud. When users register a wallet, they are given a public private key pair, where the public key is the wallets address and the private key is used for signing the transactions. The e-wallet is safe as long as the users private key is secure.

8. Conclusion

In this paper we propose a lightweight peer-to-peer permitted blockchain system, having only the minimum functionality to ensure the seminal properties of immutability, and third-party verifiability. We have shown how this is achieved without proof-of-work and in general at minimum amount of resources. We have further introduced a novel lightweight consensus protocol supporting very high transaction rate, and demonstrated its correctness and viability. Correctness is conservative - any disconsent or failure results in a given round being cancelled.

Our analysis and simulation show, that in spite of this, with reasonable failure assumptions, the impact of

failure on overall transaction performance is negligible, with consensus being reached in over 99% of the rounds for at least 97% of the players.

Lastly we have discussed key applications of this new blockchain that we have prototyped.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Mar. 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf (visited on 11/10/2019).

[2] V. Buterin, *Ethereum: A next-generation smart contract and decentralized application platform*, Accessed: 2020-05-26, 2014. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper (visited on 05/07/2020).

[3] A. Back, "Hashcash - a denial of service countermeasure," Sep. 2002.

[4] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *ArXiv*, 2017. [Online]. Available: https://arxiv.org/pdf/1710.09437.pdf (visited on 06/15/2020).

[5] H. Massias, X. S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirement," in *The 20th Symposium on Information Theory in the Benelux*, Citeseer, 1999.

[6] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Language Systems*, vol. 4, pp. 382–401, Jul. 1982.

[7] G. Bracha, "An asynchronous [(n - 1)/3]-resilient consensus protocol," in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, Association for Computing Machinery, 1984, pp. 154–162.

[8] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[9] E. Androulaki, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, A. Barger, S. Cocco, J. Yellick, V. Bortnikov, C. Cachin, K. Christidis, A. Caro, D. Enyeart, and G. Laventman, "Hyperledger fabric: A distributed operating system for permissioned blockchains," Apr. 2018, pp. 1–15. [Online]. Available: https://arxiv.org/pdf/1801.10228.pdf (visited on 11/10/2019).

[10] M. Sadek Ferdous, M. Jabed Morshed Chowdhury, M. A. Hoque, and A. Colman, "Blockchain consensus algorithms: A survey," *ArXiv*, 2020. [Online]. Available: https://arxiv.org/pdf/2001.07091.pdf (visited on 06/01/2020).

[11] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *Self-published paper, August*, vol. 19, 2012.

[12] J. Kwon, "Tendermint: Consensus without mining," *Draft v. 0.6, fall*, vol. 1, no. 11, 2014.

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

[14] R. Merkle, "Protocols for public key cryptosystems," in *In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society*, Apr. 1980, pp. 122–133.

[15] G. S. Vernam, "Cipher printing telegraph systems: For secret wire and radio telegraphic communications," *Journal of the AIEE*, vol. 45, no. 2, pp. 109–115, Feb. 1926. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6534724 (visited on 05/28/2020).

[16] B. Alpern and F. Schneider, "Defining liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, Oct. 1985.

[17] *Network time protocol version 4: Protocol and algorithms specification*, https://tools.ietf.org/html/rfc3912, Jun. 2010. (visited on 05/08/2020).