Centrally Governed Blockchains: Optimizing Security, Cost, and Availability

Leif-Nissen Lundbæk, Andrea Callia D'Iddio, and Michael Huth(✉)

Department of Computing, Imperial College London, London SW7 2AZ, UK
{leif.lundbaek,a.callia-diddio14,m.huth}@imperial.ac.uk

Abstract. We propose the formal study of blockchains that are owned and controlled by organizations and that neither create cryptocurrencies nor provide incentives to solvers of cryptographic puzzles. We view such approaches as frameworks in which system parts, such as the cryptographic puzzle, may be instantiated with different technology. Owners of such a blockchain procure puzzle solvers as resources they control, and use a mathematical model to compute optimal parameters for the cryptographic puzzle mechanism or other parts of the blockchain. We illustrate this approach with a use case in which blockchains record hashes of financial process transactions to increase their trustworthiness and that of their audits. For Proof of Work as cryptographic puzzle, we develop a detailed mathematical model to derive MINLP optimization problems for computing optimal Proof of Work configuration parameters that trade off potentially conflicting aspects such as availability, resiliency, security, and cost in this governed setting. We demonstrate the utility of such a *mining calculus* by applying it on some instances of this problem. We hope that our work may facilitate the creation of *domain-specific* blockchains for a wide range of applications such as trustworthy information in Internet of Things systems and bespoke improvements of legacy financial services.

1 Introduction

There is little doubt that modern accounting systems have benefitted, ever since the advent of commercial computing machines, from the digitization of the processing and recording of financial transactions. The automated processing of payroll information in the 1950ies was perhaps one of the earliest examples of such benefits: IBM introduced its *702 Data Processing System* for businesses in 1953. And the use of RFID technology or smart phones for contactless payment of small items such as coffees is a more recent example thereof.

It is then striking that the mechanisms used for managing the integrity of accounts are, in essence, those developed at least a thousand years ago. What we call the modern *double-entry bookkeeping* was already used by Florentine merchants in the 13th century, for example. Without going into great detail, the key idea is in simplified terms that each account has an associated *dual* account and that each credit in one account is recorded as a debit in that dual account. This allows for the formulation and verification of an important *financial invariant*:

L. Aceto et al. (Eds.): Larsen Festschrift, LNCS 10460, pp. 578–599, 2017.
DOI: 10.1007/978-3-319-63121-9_29

no matter how complex financial transactions may be, or how many transactions may occur, it must always be the case that over the totality of accounts

"Assets equal liabilities plus capital."

Modern realizations of this method may enrich account entries with time stamps and other contextual data so that the flow of assets can be better understood, for example to support an audit. The above invariant may be quite simple to verify, and its verification may give us reassurance that every debit has an associated credit. But it does not prevent the recording of transactions that may be unauthorized, fraudulent, or that may be incorrect due to human error. For example, transaction records within accounting books may be manipulated to commit fraud whilst these manipulations still satisfy the above invariant.

One may say that processing of transactions is governed by a form of *legal code* that is informed by policy on fraud prevention and detection, regulation, compliance, risk, and so forth. But the enforcement of such legal code within the *technical code* that operationalizes modern financial processes has been difficult at best, and too costly or impossible at worst.

Digitized financial processes can utilize cryptographic primitives to help with narrowing this gap between legal and technical code: digital signatures can be associated to transactions (for example embedded within transaction objects), and commitment schemes can be used to realize consistent distributed storage whose consistency is resilient to adversarial manipulation; see for example the discussion of Byzantine Agreement Protocols in [19]. But the advent of decentralized, *eventual consistency* storage protocols, as pioneered in the cryptocurrency Bitcoin [13], opened up a new way of thinking about the processing of financial transactions, even of creating and managing a currency as a unit of account.

In the Bitcoin network, a randomized race of solving a cryptographic puzzle, *Proof of Work*, is used to elect a leader whose block will be added to the blockchain. The leader election is done for each new block, and a consensus protocol ensures that the majority of network nodes have a consistent copy of the blockchain. There is little doubt that cryptocurrencies are one of the most important innovations [1,14], along with the invention and introduction of central banks, in financial services since the advent of the double-entry bookkeeping.

In this paper, we investigate how governed, closed blockchains can be designed so that they can support the resilient, distributed, and *trustworthy* storage of authentication of transactions within conventional financial processes. Such governed systems restrict access, notably to the definition and solving of cryptographic puzzles. Therefore, they give us better control on balancing the use of energy for puzzle solving with the security of the Proof of Work algorithm when compared with open systems that rely on Proof of Work, such as Bitcoin. Specifically, we propose that transactions (in the sense of Bitcoin) within blocks are hashes of transactions (in the sense of conventional financial processes). We then define mathematical models that describe the design space of such a blockchain in terms of the cryptographic puzzle used – in this paper Proof of Work, in terms of expected availability, resiliency, security, and cost, and in terms that reflect that the system is centrally governed. We stress that our approach is also consistent with transactions within blockchains that encode transaction history, which we don't consider in the use case of this paper.

Outline of Paper. In Sect. 2 we present our use case. Our mathematical model for governed Proof of Work is subject of Sect. 3. The derivation of optimization problems for these is done in Sect. 4 and shown to support robust design security in Sect. 5. An algorithm for solving such optimization problems and experimental results are reported in Sect. 6. The wider context of our work and related work are discussed in Sect. 7, and the paper concludes in Sect. 8.

2 Use Case

Our use case is one of a financial process that creates financial transactions. We would like to enhance the trustworthiness of this process through a blockchain that records hash-based authentications of transactions, see Fig. 1, where the interaction between the legacy process and the blockchain is conceptually simple – and consistent with the use of double-entry bookkeeping if desired.

Our assumption is that the event streams of such transactions are not linearizable and so we cannot rely on techniques such as hash chains [7] to obtain immutability of transactions. A hash chain could also be recomputed by an attacker with partial control of the system. Alternatively a blockchain created through consensus protocols such as BFT [10,19] would also be subject to manipulation after consensus protocols have been executed and blocks added to the chain. A blockchain based on cryptographic puzzles is thus much more resilient to such manipulation attacks, since it takes considerable effort to solve the number of cryptographic puzzles needed for rewriting parts or all of a blockchain.

Our data model represents a transaction as a string *input*, authenticated with a hash $hash(input)$. String *input* may be a serialization of a transaction object that contains relevant information such as a time stamp of the transaction, a digital signature of the core transaction data and so forth. The trustworthiness of transaction *input* is represented outside of the blockchain by the triple

$$(input, hash(input), location) \tag{1}$$

where *location* is either the block height (≥ 0) of a block b in the blockchain such that $hash(input)$ occurs in block b or *location* is NULL, indicating that the transaction is not yet confirmed on the blockchain.

The hashes $hash(input)$ of transactions that still need to be confirmed are propagated on the blockchain network, where they are picked up by miners and integrated into blocks for Proof of Work. We assume a suitable mechanism by which nodes that manage legacy accounts learn the blockheights of their transactions that have been successfully added to the blockchain. Such nodes may have a full copy of the blockchain and update *location* values in accounts if the hash of the corresponding transaction occurs in a block that was just added.

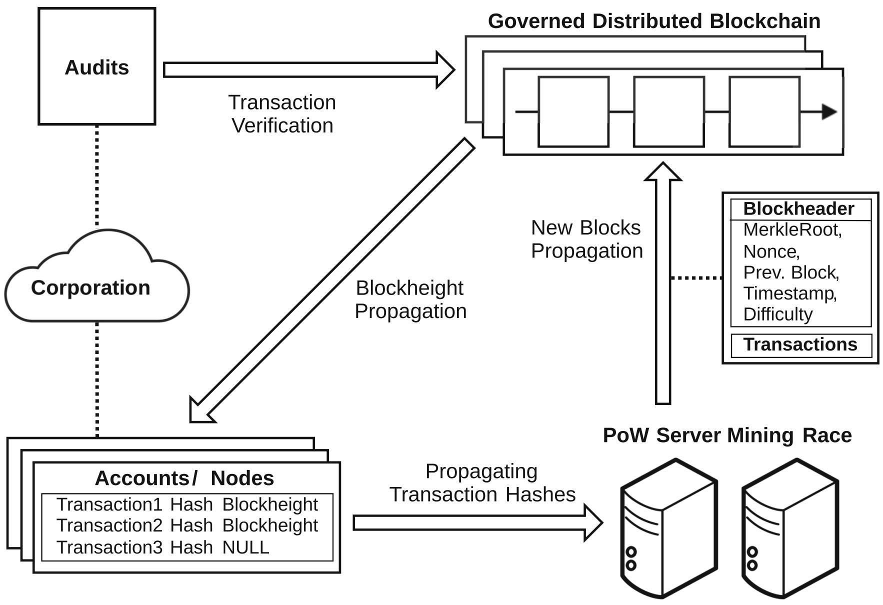
Audits
Transaction Verification
Governed Distributed Blockchain
Corporation
Blockheight Propagation
New Blocks Propagation
Blockheader
MerkleRoot, Nonce, Prev. Block, Timestamp, Difficulty
Transactions
PoW Server Mining Race
Accounts/ Nodes
Transaction1 Hash Blockheight
Transaction2 Hash Blockheight
Transaction3 Hash NULL
Propagating Transaction Hashes

Fig. 1. Governed blockchain for financial process authentications: note that **Transactions** recorded in blocks on the left are mere hashes of transactions conducted in the legacy system. The latter are listed in **Accounts/Nodes** on the left

A transaction is unverified if its *location* value is NULL or if its hash does not equal the one stored externally as in (1); it is trustworthy if $0 \leq location$ and $location + k \leq currentBlockHeight$ where $k \geq 0$ is a suitable constant and *currentBlockHeight* denotes the number of blocks added to the blockchain so far. The value of k may be a function of how fast blocks are added to the chain on average, to ensure sufficient resiliency of trustworthiness. An auditor could then inspect any transaction by examining its triple stored as in (1). If *location* equals NULL or if $location + k > currentBlockHeight$, the transaction is considered neither valid nor trustworthy by the auditor. Otherwise, we have $0 \leq location$ and $location + k \leq currentBlockHeight$ and the auditor uses the Merkle tree hash in block *location* to verify that $hash(input)$ is in the block of height b. If that is the case, the auditor considers the transaction to be verified; otherwise, the auditor considers the transaction not to be trustworthy.

Note that this use case does not require transaction scripts to be stored, nor any run-time system for verifying such transactions. But the mathematical modelling approach we present in this paper is consistent with use cases that have such script generation and verification support.

System Architecture. A system architecture that could support such a use case is shown in Fig. 1. Unverified transactions have their hashes propagated on the network. Miners pick up those hashes and integrate them into blocks for Proof of Work. We abstract away how miners manage their pools of hashes and how Proof of Work blocks are propagated and added to the blockchain; this gives us flexibility in the use of blockchain technology. Once blocks are added to the blockchain, blockheights are propagated to the legacy account. As mentioned above, these accounts could have full copies of the blockchain and thus implement their own update mechanisms for value *location* in triples stored as in (1).

Auditors would interface with both accounts and the blockchain to verify, in a trustworthy manner, the authenticity of transactions. Any transaction that is not verified as discussed above would be flagged up in this audit. Any pre-existing audit process – which may focus on compliance, regulations and other aspects – is consistent with such *trustworthiness checking*; and the trustworthiness of the pre-existing audit process would be increased as it would refuse to certify any financial transaction histories that involved a transaction that is not authenticated on the blockchain.

Analysis. The approach we advocate in this section seems consistent with consensus mechanisms as used in Bitcoin but it may also support 2-phase commitment schemes as proposed in [5]. Our system architecture allows for full nodes to be associated with accounts, sets of accounts or corporate boundaries. Our blockchain does not create any currency, and so there is no inherent incentive to mine. But there is an incentive for the owners of this blockchain to allocate mining resources in a manner that establishes trustworthiness of transactions as recorded in this blockchain. We deem elimination of incentives for miners and their game-theoretic implications to be a benefit, as well as the simple ways of propagating trust through hashes of transactions. Such a blockchain may also be consulted by legacy systems to inform the authorization of further financial transactions.

Our blockchain does not spend any funds and so has no problem of *double spending*, and double spending in the legacy system would be detectable with existing mechanisms such as audits. Our approach does allow for *double authentication* though: a transaction hash may occur more than once in a blockchain, be it in the same block or in different blocks. We deem this to be unproblematic as audits would only need to establish *some*, sufficiently old, authentication of the transaction in the blockchain to establish its trustworthiness – noting that hash-based authentication is deterministic.

One expectation is that blocks would only be added to the blockchain if signed by one of the miners that is resourced for this Proof of Work service. This requires that the public keys of such miners are securely stored and available within the system. Over time, some of these miners may be removed from such a list (e.g. decommissioned) and new ones may be added (e.g. system upgrade).

3 Mathematics for Centrally Governed Proof of Work

Our model assumes a cryptographic hash function

$$h\colon \{0,1\}^p \to \{0,1\}^n$$

where $p \geq n > 0$ such that h has *puzzle friendliness* [14]. The *level of difficulty* d is an integer satisfying $0 < d < n$: Proof of Work has to produce some x where $h(x)$ has at least d many leftmost 0 bits. We write $T > 0$ for the time to compute a sole hash $h(x)$ and to decide whether it has at least d leftmost zeros. Since the range of d will be relatively small, we make T a device-dependent constant.

Our probabilistic modeling will treat h in the *Random Oracle Model* (ROM): function h is chosen uniformly at random from all functions of type $\{0,1\}^p \rightarrow \{0,1\}^n$; that is to say, h is a deterministic function such that any x for which h has not yet been queried will have the property that $h(x)$ is governed by a truly random probability distribution over $\{0,1\}^n$.

We may assume that x consists of a block header which contains some random data field – a nonce *nonce* of bitlength r, that this nonce is initialized, and that the nonce is then increased by 1 each time the hash of x does not obtain Proof of Work. In particular, this yields that $\{0,1\}^p \cong \{0,1\}^{p-r} \times \{0,1\}^r$ where $0 < r < p$: the input to h will be of form $x = \mathit{data} \,||\, \mathit{nonce}$ where *data* and *nonce* have $p - r$ and r bits, respectively. Our use of ROM will rely on the following assumption:

Assumption 1 (Invariant). *The mining of a block with one or more miners will use an input to h at most once, be it within or across miners' input spaces.*

This assumption and ROM give us that hash values are always uniformly distributed in the output space during a mining race. Now consider having $s > 1$ many miners that run in parallel to find Proof of Work, engaging thus in a *mining race*. We assume these miners run with the same configurations and hardware. As already discussed, in our approach miners do not get rewarded:

Assumption 2 (Miners). *Miners are a resource controlled by the governing organization or consortium, and have identical hardware. In particular, miners are not rewarded nor have the need for incentive structures.*

But miners may be corrupted and misbehave, for example they may refuse to mine. To simplify our analysis, we assume miners begin the computation of hashes in approximate synchrony:

Assumption 3 (Approximate Synchrony). *Miners start a mining race at approximately the same time.*

For many application domains, this is a realistic assumption as communication delays to miners would have a known upper bound that our models could additionally reflect if needed.

Next, we want to model the *race* of getting a Proof of Work where each miner j has some data data_j. To realize Assumption 1, it suffices that each miner j has a nonce nonce_j in a value space of size

$$\lambda = \lfloor 2^r / s \rfloor$$

such that these nonce spaces are mutually disjoint across miners.

Our probability space has $(data_j)_{1\leq j\leq s}$ and d as implicit parameters. For each miner j, the set of basic events E^j is

$$E^j = \{\otimes^k \cdot \checkmark \mid 0 \leq k \leq \lambda\} \cup \{\mathsf{failure}\} \tag{2}$$

Basic event $\mathsf{failure}$ denotes the event that all λ nonce values $nonce_j$ from $\{(j-1)\cdot\lambda, \dots, j\cdot\lambda - 1\}$ for miner j failed to obtain Proof of Work for $data_j$ at level of difficulty d. Basic event $\otimes^k \cdot \checkmark$ models the event in which the first k such nonce values failed to obtain Proof of Work for $data_j$ at level d but the $k+1$th value of $nonce_j$ did render such Proof of Work for $data_j$.

To model this mining race between s miners for $(data_1, data_2, \dots, data_s)$ and d as implicit parameters, we take the product $\prod_{j=1}^{s} E^j$ of s copies E^j and quotient it via an equivalence relation $\equiv$ on that product $\prod_{j=1}^{s} E^j$, which we now define formally.

Definition 1

1. *The s-tuple* $(\mathsf{failure}, \dots, \mathsf{failure})$ *models failure of this mining race, and it is $\equiv$ equivalent only to itself.*
2. *All s-tuples $a = (a_j)_{1\leq j\leq s}$ other than tuple* $(\mathsf{failure}, \dots, \mathsf{failure})$ *model that the mining race succeeded for at least one miner. For such an s-tuple a, the set of natural numbers k such that $\otimes^k \cdot \checkmark$ is a coordinate in a is non-empty and therefore has a minimum* $\min(a)$. *Given two s-tuples $a = (a_j)_{1\leq j\leq s}$ and $b = (b_j)_{1\leq j\leq s}$ both different from* $(\mathsf{failure}, \dots, \mathsf{failure})$, *we can then define*

$$a \equiv b \textit{ iff } \min(a) = \min(b)$$

So two non-failing tuples are equivalent if they determine a first (and so final) Proof of Work at the same round of the race. This defines an equivalence relation $\equiv$ and adequately models a synchronized mining race between s miners.

The interpretation of events $\otimes^k \cdot \checkmark$ in the mining race is then the equivalence class of all those tuples a for which $\min(a)$ is well defined and equals k: all mining races that succeed first at round k. The meaning of $\mathsf{failure}$ is still overall failure of the mining race, the equivalence class containing only tuple $(\mathsf{failure}, \dots, \mathsf{failure})$. The set of events for the Proof of Work race of s miners is therefore

$$E^s = \{\otimes^k \cdot \checkmark \mid 0 \leq k \leq \lambda\} \cup \{\mathsf{failure}\} \tag{3}$$

In (3), expression $\otimes^k \cdot \checkmark$ denotes an element of the quotient

$$(\prod_{j=1}^{s} E^j)/\equiv$$

namely the equivalence class of tuple $(\otimes^k \cdot \checkmark, \mathsf{failure}, \mathsf{failure}, \dots, \mathsf{failure})$. Next, we define a probability distribution $prob^s$ over E^s. To derive the probability $prob^s(\otimes^k \cdot \checkmark)$, recall

$$\tilde{p}(\otimes^k) = (1 - 2^{-d})^k$$

as the probability that a given miner does not obtain Proof of Work at level d in the first k rounds. By Assumption 1, these miners work independently and over disjoint input spaces. By ROM, the expression

$$\left[(1 - 2^{-d})^k\right]^s = (1 - 2^{-d})^{k\cdot s}$$

therefore models the probability that none of the s miners obtains Proof of Work in the first k rounds. Appealing again to ROM and Assumption 1, the behavior at round $k + 1$ is independent of that of the first k rounds. Therefore, we need to multiply the above probability with the one for which at least one of the s miners will obtain a Proof of Work in a single round. The latter probability is the complementary one of the probability that none of the s miners will get a Proof of Work in a sole round, which is $(1-2^{-d})^s$ due to the ROM independence. Therefore, we get

$$prob^s(\otimes^k \cdot \checkmark) = (1 - 2^{-d})^{k\cdot s} \cdot [1 - (1 - 2^{-d})^s] \tag{4}$$

This defines a probability distribution with a non-zero probability of $\mathsf{failure}$. Firstly,

$$\sum_{k=0}^{\lambda}(1 - 2^{-d})^{k\cdot s} \cdot [1 - (1 - 2^{-d})^s]$$

is in $(0, 1)$: to see this, note that this sum equals

$$[1 - (1 - 2^{-d})^s] \cdot \frac{1 - [(1 - 2^{-d})^s]^{\lambda+1}}{1 - (1 - 2^{-d})^s} = 1 - (1 - 2^{-d})^{s\cdot(\lambda+1)}$$

Since $0 < d, s$, the real $1 - 2^{-d}$ is in $(0, 1)$, and the same is true of any integral power thereof. Secondly, $prob^s$ becomes a probability distribution with the non-zero probability $prob^s(\mathsf{failure})$ being $1 - prob^e(E^s\backslash\{\mathsf{failure}\})$, that is

$$prob^s(\mathsf{failure}) = (1 - 2^{-d})^{s\cdot(\lambda+1)} \tag{5}$$

That this failure probability is almost identical to that for $s = 1$ is an artefact of our modeling: if each miner has 64 bits of nonce space, e.g., then our model would have $r = 64 \cdot s$, so failure probabilities do decrease as s increases.

4 Mathematical Optimization in Mining Design Space

Generality of Approach. We want to optimize the use of $s > 1$ miners using a level of difficulty d, and a bit size r of the global nonce space with respect to an objective function. The latter may be a cost function, if containing cost is the paramount objective or if a first cost estimate is sought that can then be transformed into a constraint to optimize for a security objective – for example to maximize the level of difficulty d, as seen further below. Higher values of d add more security: it takes more effort to mine a block and so more effort to manipulate the mining process and used consensus mechanism. But lower values of d may be needed, for example, in high-frequency trading where performance can become a real issue. We want to understand such trade-offs. Moreover, we want to explore how corruption of some miners or inherent uncertainty in the number of deployed miners or in the level of difficulty across the lifetime of a system may influence the above tradeoffs.

Optimizing Cost and Security. The flexibility of our approach includes the choice of an objective function for optimization. Let us first consider an objective function

$$\mathrm{Cost}(s, r, d) = \mathrm{TVC} \cdot E^s(noR) \cdot s + \mathrm{TFC} \cdot s \tag{6}$$

that models cost as a function of the number of miners s, the bit size of the nonce r – implicit in random variable $E^s(noR)$, and the level of difficulty d; where we want to *minimize* cost.

The real variable TVC models the *variable* cost of computing *one* hash for *one* miner, reflecting the device-dependent speed of hashes and the price of energy. The real variable TFC models the *fixed* costs of *having one miner*; this can be seen as modeling procurement and depreciations. Variables s, r, and d are integral, making this a *mixed integer* optimization problem [8]. The expression $E^s(noR)$ denotes the *expected number of rounds* (of approximately synchronous hash attempts) needed to mine a block in a mining race that uses s miners, level of difficulty d, and nonce bitsize r. The derivation of this expression below shows that it is non-linear, making this a MINLP optimization problem [8,16].

We may of course use other objective functions. One of these is simply the expression d, which we would seek to *maximize*, the intuition being that higher values of d give us more trust into the veracity of a mined block and the blockchains generated in the system. Figure 2 shows an example of a set of constraints and optimizations of security and cost for this.

$$\begin{array}{llll} 0 < s_l \leq s \leq s_u & 0 < d_l \leq d \leq d_u & 0 < r_l \leq r \leq r_u & \epsilon \geq prob^s(\mathsf{failure}) \\ \tau_u \geq T \cdot E^s(noR) \geq \tau_l & & \delta_2 \geq prob^s(disputes\ within\ \mu) & \\ \delta \geq prob^s(PoWTime > th) & & \delta_1 \geq prob^s(PoWTime < th') & \end{array}$$

Fig. 2. Constraint set $\mathcal{C}$ for two optimization problems: (a) *minimize* $\mathrm{Cost}(s, r, d)$ as in (6) subject to constraints in $\mathcal{C}$; and (b) *maximize* d subject to $\mathcal{C} \cup \{\mathrm{Cost}(s, r, d) \leq budget\}$ for cost bound *budget*. This is parameterized by constants $0 \leq \delta, \delta_1, \delta_2, \epsilon, th, th', \tau_l, \mathrm{TVC}, \mathrm{TFC}$ and $0 < T, s_l, r_l, d_l$. Variables or constants $s_l, s_u, s, d_l, d_u, d, r_l, r_u, r$ are integral

Integer constants s_l and s_u provide bounds for variable s, and similar integer bounds are used to constrain integer variables r and d. The constraint for ϵ uses it as upper bound for the probability of a mining race failing to mine a block. The next two inequalities stipulate that the expected time for mining a block is within a given time interval, specified by real constants τ_l and τ_u.

The real constant δ_2 is an upper bound for

$$prob^s(disputes\ within\ \mu)$$

the probability that more than one miner finds PoW within μ seconds in the same, approximately synchronous, mining race. The constraint for real constant δ says that the probability

$$prob^s(PoWTime > th)$$

of the *actual* time for mining a block being above a real constant th is bounded above by δ. This constraint is of independent interest: knowing that the expected time to mine a block is within specified bounds may not suffice in systems that need to assure that blocks are *almost always* (with probability at least $1 - \delta$) mined within a specified time limit. Some systems may also need assurance that blocks are almost always mined in time *exceeding* a specified time limit th'. We write

$$prob^s(PoWTime < th')$$

to denote that probability, and add a dual constraint, that the actual time for mining a block has a sufficiently small probability $\leq \delta_1$ of being faster than threshold th'.

Constraints as Analytical Expressions. We derive analytical expressions for random variables occurring in Fig. 2. Beginning with $E^s(noR)$, we have

$$E^s(noR) = \sum_{0 \leq k \leq \lambda} prob^s(\otimes^k \cdot \checkmark) \cdot (k+1) \tag{7}$$

which we know to be equal to

$$\sum_{0 \leq k \leq \lambda} (1 - 2^{-d})^{k \cdot s} \cdot [1 - (1 - 2^{-d})^s] \cdot (k+1)$$

We may rewrite the latter expression so that summations are eliminated and reduced to exponentiations: concretely, we rewrite $\sum_{0 \leq k \leq \lambda} prob(\otimes^k \cdot \checkmark) \cdot (k+1)$, the righthand side of (7), to $\lambda + 1$ summations, each one starting at a value between 0 and λ, where we exploit the familiar formula

$$\sum_{k=a}^{b} x^k = \frac{x^a - x^{b+1}}{1 - x}$$

This renders

$$E^s(noR) = \frac{1 - y^{\lambda+1} - (\lambda+1) \cdot (1 - y) \cdot y^{\lambda+1}}{1 - y} \tag{8}$$

where we use the abbreviation

$$y = (1 - 2^{-d})^s \tag{9}$$

The expected time needed to get a proof of work for input *data* is then given by

$$E^s(poW) = T \cdot E^s(noR) \tag{10}$$

We derive an analytical expression for the probability $prob^s(PoWTime > th)$ next. Note that $(th/T) - 1 < k$ models that the actual time taken for $k+1$ hash rounds is larger than th. Therefore, we capture $prob^s(PoWTime > th)$ as

$$\sum_{\lceil (th/T)-1 \rceil < k \leq \lambda} prob^s(\otimes^k \cdot \checkmark) = y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1} \tag{11}$$

assuming that $\lceil (th/T) - 1 \rceil < \lambda$, the latter therefore becoming a constraint that we need to add to our optimization problem. One may be tempted to choose the value of δ based on the Markov inequality, which gives us

$$prob^s(PoWTime \geq th) \leq T \cdot E^s(noR)/th$$

But we should keep in mind that upper bound $T \cdot E^s(noR)/th$ depends on the parameters s, r, and d; for example, the analytical expression for $E^s(noR)$ in (8) is dependent on λ and so dependent on r as well. The representation in (11) also maintains that expression $y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1}$ is in $[0, 1]$, i.e. a proper probability. Since $y = (1 - 2^{-d})^s$ is in $(0, 1)$, this is already guaranteed if $\lceil (th/T) - 1 \rceil + 1 \leq \lambda + 1$, i.e. if $\lceil (th/T) - 1 \rceil \leq \lambda$. But we already added that constraint to our model. Similarly to our analysis of $prob^s(PoWTime > th)$, we get

$$prob^s(PowTime < th') = 1 - (1 - 2^{-d})^{s \cdot (\lfloor (th'/T)-1 \rfloor + 1)} = 1 - y^{\lfloor (th'/T)-1 \rfloor + 1} \tag{12}$$

which needs $0 < \lfloor (th'/T) - 1 \rfloor$ as additional constraint.

To derive an analytical expression for $prob^s(disputes\ within\ \mu)$, each miner can perform $\lfloor \mu/T \rfloor$ hashes within μ seconds. Let us set

$$w = (1 - 2^{-d})^{\lfloor \mu/T \rfloor + 1} \tag{13}$$

The probability that a given miner finds PoW within μ seconds is

$$\sum_{k=0}^{\lfloor \mu/T \rfloor} (1 - 2^{-d})^k \cdot 2^{-d} = 2^{-d} \cdot \frac{1 - (1 - 2^{-d})^{\lfloor \mu/T \rfloor + 1}}{1 - (1 - 2^{-d})} = 1 - w \tag{14}$$

Therefore, the probability that no miner finds PoW within μ seconds is

$$prob^s(0\ PoW\ within\ \mu) = (1 - (1 - w))^s = w^s \tag{15}$$

The probability that exactly one miner finds PoW within μ seconds is

$$prob^s(1\ PoW\ within\ \mu) = s \cdot w^{s-1} \cdot (1 - w) \tag{16}$$

Thus, the probability that more than one miner finds PoW within μ seconds is

$$\begin{aligned} prob^s(disputes\ within\ \mu) &= 1 - prob^s(0\ PoW\ within\ \mu) - prob^s(1\ PoW\ within\ \mu) \\ &= 1 - w^s - s \cdot w^{s-1} \cdot (1-w) \\ &= 1 - w^s - s \cdot w^{s-1} + s \cdot w^{s-1} \cdot w \\ &= 1 + (s-1) \cdot w^s - s \cdot w^{s-1} \end{aligned} \tag{17}$$

Figure 3 shows the set of constraints $\mathcal{C}$ from Fig. 2 with analytical expressions and their additional constraints, we add constraint $0 \leq \lfloor \mu/T \rfloor$ to get consistency for the analytical representation of $prob^s(disputes\ within\ \mu)$.

$$\begin{aligned} & s_l \leq s \leq s_u \qquad d_l \leq d \leq d_u \qquad r_l \leq r \leq r_u \qquad \lambda = \lfloor 2^r/s \rfloor \\ & y = (1-2^{-d})^s \qquad w = (1-2^{-d})^{\lfloor \mu/T \rfloor + 1} \qquad 0 \leq \lfloor \mu/T \rfloor \\ & \epsilon \geq y^{\lambda+1} \qquad \lceil (th/T) - 1 \rceil < \lambda \qquad 0 < \lfloor (th'/T) - 1 \rfloor \\ E^s(noR) = {} & \frac{1 - y^{\lambda+1} - (\lambda+1) \cdot (1-y) \cdot y^{\lambda+1}}{1-y} \\ & \tau_u \geq T \cdot E^s(noR) \geq \tau_l \qquad \delta_1 \geq 1 - y^{\lfloor (th'/T) - 1 \rfloor + 1} \\ & \delta \geq y^{\lceil (th/T) - 1 \rceil + 1} - y^{\lambda+1} \\ & \delta_2 \geq 1 + (s-1) \cdot w^s - s \cdot w^{s-1} \end{aligned} \tag{18}$$

Fig. 3. Arithmetic version of set of constraints $\mathcal{C}$ from Fig. 2, with additional soundness constraints for this representation. Feasibility of (s, r, d) and $r_u \geq r' > r$ won't generally imply feasibility of (s, r', d) due to the constraint in (18)

5 Robust Design Security

Our model above captures design requirements or design decisions as a set of constraints, to optimize or trade off measures of interest subject to such constraints. We can extend this model to also *manage uncertainty* via robust optimization [2]. Such uncertainty may arise during the lifetime of a system through the possibility of having corrupted miners, needed flexibility in adjusting the level of difficulty, and so forth. For example, corrupted miners may refuse to mine, deny their service by returning invalid block headers, pool their mining power to get more mining influence or they may simply break down. Robust optimization treats such uncertainty as non-deterministic choice and refers to it as *strict* or *Knightian* uncertainty.

Consider $1 \leq l < s$ corrupted miners. We can model their *pool power* by appeal to ROM and the fact that the mining race is approximately synchronized: the probability that these l miners win $c > 0$ many subsequent mining races is then seen to be $(l/s)^c$. We can therefore bound this with a constant δ_3 as in Fig. 3.

We model uncertainty in the number of miners available by an integer constant u_s as follows: if s miners are deployed, then we assume that at least $s - u_s$

and at most s many miners participate reliably in the mining of legitimate blocks: they will not mine blocks that won't verify and only submit mined blocks that do verify to the network. Constant u_s can model aspects such as denial of service attacks or a combination of such attacks with faults: $u_s = 3$, e.g., subsumes the scenario in which one miner fails and two miners mine invalid blocks.

Integer constant u_d models the uncertainty in the deployed level of difficulty d: intuitively, our analysis should give us results that are robust in that they hedge against the fact that any of the values d' satisfying

$$| d - d' | \leq u_d$$

may be the actually running level of difficulty. This enables us to understand a design if we are unsure about which level of difficulty will be deployed or if we want some flexibility in dynamically adjusting the value of d in the running system.

The corresponding robust optimization problem for cost minimization is seen in Fig. 4. It adds to the constraints we already consider further requirements on constants l, c, and δ_3 as well as the constraint

$$l^c \leq \delta_3 \cdot s^c$$

The robustness of analysis is achieved by a change of the objective function from $\mathrm{Cost}(s, r, d)$ to

$$\mathrm{Cost}^{u_s}_{u_d}(s, r, d) = \max_{s-u_s \leq s' \leq s,\ |d-d'| \leq u_d} \mathrm{Cost}(s', r, d') \tag{19}$$

The latter computes a worst-case cost for triple (s, r, d) where s and d may vary independently subject to the strict uncertainties u_s and u_d, respectively. We call a triple (s, r, d) *feasible* if it satisfies all constraints of its optimization problem. Costs such as the one in (19) for a triple (s, r, d) are only considered for optimization if all triples (s', r, d') used in (19) are feasible – realized with predicate $\mathit{feasible}^{u_s}_{u_d}$: robust optimization guarantees [2] that the feasibility of solutions is invariant under the specified strict uncertainty (here u_s and u_d).

6 Experiments and Validation

We submitted simple instances of the optimization problem in Fig. 4 to state of the art MINLP solvers. All these solvers reported, erroneously, in their preprocessing stage that the problem is infeasible. These solvers were not designed to deal with problems that combine such small numbers and large powers, and rely on standard floating point implementations. Therefore, we wrote a bespoke solver in Haskell that exploits the fact that we have only few integral variables within limited ranges so that we can explore their combinatorial space completely to determine feasibility and therefore optimality as well.

Experimental Setup. We solve the robust optimization problem for the analytical expressions we derived above with the algorithm depicted in Fig. 5. This

$$\min\{\mathrm{Cost}^{u_s}_{u_d}(s,r,d) \mid \mathit{feasible}^{u_s}_{u_d}(s,r,d)\}$$

subject to the set of constraints $\mathcal{C}$ from Figure 3 together with

$$\begin{array}{lll} 4 = l < s & c = 6 & 0.001 = \delta_3 \\ l^c \leq s^c \cdot \delta_3 & u_s = 5 & u_d = 3 \end{array}$$

Fig. 4. Robust cost optimization for the set of constraints from Fig. 3, where up to $u_s = 5$ miners may be non-functioning, refusing to mine or mining invalid blocks; where the level of difficulty may vary by up to +/−3; and where the probability of any mining *pool* of size $l = 4$ winning $c = 6$ consecutive mining races is sufficiently small (here $\delta_3 = 0.001$). Predicate $\mathit{feasible}^{u_s}_{u_d}(s,r,d)$ characterizes *robustly feasible* triples and is true iff all triples (s', r, d') with $s - u_s \leq s' \leq s$ and $|d - d'| \leq u_d$ are feasible

```
input     : $p$, $\alpha$, and values for all constants in Figure 4
invariant: $list$ lists tuples $(s, r, d, cost)$ in descending order for $d$
begin
    define all constants for constraints in Figure 4;
    $list = [(s,r,d,cost) \mid cost = Cost(s,r,d), feasibleFloat(s,r,d)$ is true];
    $list = [(s,r,d,cost) \in list \mid feasibleFloat^{u_s}_{u_d}(s,r,d)$ is true];
    while $(\exists (s,r,d,cost) \neq (s',r',d,cost') \in list)$ do
        case $(cost' < cost) \vee (r' < r)$ remove $(s,r,d,cost)$ from list);
        case $(cost < cost') \vee (r < r')$ remove $(s',r',d,cost')$ from list);
    end while
    $c_m = \min\{c \mid \exists (s,r,d,cost) \in list\}$;
    while $(\exists (s,r,d,cost) \in list\colon cost > \alpha \cdot c_m)$ do
        remove $(s,r,d,cost)$ from list;
    end while
    results = list of first $p$ tuples from list;
    $results = [(s,r,d,cost) \in results \mid feasibleBigFloat^{u_s}_{u_d}(s,r,d)$ is true];
    return results;
end
```

Fig. 5. Algorithm, written in imperative style of list processing, for reporting the best p robustly feasible tuples $(d, r, s, cost)$ such that d is maximal subject to the cost $cost = Cost(s, r, d)$ satisfying $cost \leq \alpha \cdot c_m$ where c_m is the minimal cost for all robustly feasible tuples (s, r, d) and $\alpha \geq 1$ is a tolerance factor for increasing cost beyond c_m. Predicate $feasibleFloat(s, r, d)$ is true iff all constraints in Fig. 3 are true for this choice of s, r, and d under normal precision floats. Predicates $feasibleBigFloat$ and $feasibleBigFloat^{u_s}_{u_d}$ are true iff their mathematical definition is true under arbitrary-precision floating points (applying package `Data.BigFloat` version 2.13.2).

algorithm has as input the set of constraints, a parameter p and a parameter α. It will output at most p robustly feasible tuples $(s, r, d, cost)$ from a list of all robustly feasible such tuples as follows: it will identify the maximal values of d for which such tuples are robustly feasible, and it will report exactly one such tuple for each value of d where r is minimal, and $cost$ is minimal whilst also bounded above by $\alpha \cdot c_m$ where c_m is the globally minimal cost. This also determines the values of s in these tuples and so the algorithm terminates.

Now, having defined the required analytical expressions and the algorithm to report the best p robustly feasible tuples in Fig. 5, we also want to validate these expressions and the algorithm experimentally. Our setup for this is based on pure Haskell code, as functional – and in particular – Haskell programs offer the advantages of being modular in the dimension of functionality, being strongly typed as well as supporting an easy deconstruction of data structures, particularly lists [3]. Furthermore, the arbitrary-precision verification is handled by the external `Data.BigFloat` package, which is also written in Haskell. Further verification and validation of the received results are pursued by unit testing using an arbitrary precision calculator. Moreover, our experiments ran on a machine with the following specifications: Intel(R) Xeon(R) CPU E5-4650 with 64 cores and 2.70 GHz and 500 GB total RAM. Our machines required between 322.12 and 261.425 s to compute the respective optimizations. The entire experiment took 10,457.58 s.

Table 1. Constants for our experiments. This does not specify the values of τ_u which will vary in experiments. Some experiments will also vary the values of δ, δ_2 or δ_3

$s_l = 4$	$s_u = 80$	$r_l = 24$	$r_u = 64$
$d_l = 4$	$d_u = 64$	$\text{TVC} = 2 \cdot 10^{-12}$	$\text{TFC} = 3000$
$\alpha = 1.5$	$T = 0.002 \cdot 10^{-9}$	$th = 300$	$th' = 300$
$\delta = 10^{-9}$	$\delta_1 = 1$	$\delta_2 = 0.001$	$\delta_3 = 0.001$
$\tau_l = 0$	$\mu = 1/10000$	$\epsilon = 2^{-64}$	$k = 5$
$u_d = 3$	$u_s = 5$	$c = 6$	$l = 4$

We instantiate the model in Fig. 4 with the constants shown in Table 1. We choose T to be $1/(50 \cdot 10^9) = 0.02 \cdot 10^{-9}$ for a mining ASIC from early 2016 with an estimated cost of 2700 USD at that time, so a fixed cost of TFC = 3000 USD seems reasonable. Let us now explain the value $2 \cdot 10^{-12}$, which models the energy cost of a sole hash (we can ignore other costs on that time scale). A conservative estimate for the power consumption of an ASIC is 10 W per Gigahashes per second, i.e. 10 W per Gh/s. We estimate the cost of one kilowatt hour kWh to be about 10 cents. A kWh is 3600 s times kW and one kW is 1000 W. So 10 W per Gh/s equals $10 \cdot 3600$ W, which amounts to 36 kWh. So the cost for this is $36 \cdot 10$ cents per hour, i.e. 360 cents per hour. But then this costs $360/3600 = 0.1$ cents per second. The price for a sole hash is therefore 0.1 divided by $50 \cdot 10^9$, which equals $\text{TVC} = 2 \cdot 10^{-12}$.

We insist on having at least 4 miners and cap this at 80 miners. The *shared* nonce space for miners is assumed to be between 24 and 64 bits. The level of difficulty is constrained to be between 4 and 64. We list optimal tuples that are within a factor of $\alpha = 1.5$ of the optimal cost. We make the value th' irrelevant by setting $\delta_1 = 1$ which makes the constraint for th' vacuously true. The probability for mining failure is not allowed to exceed $\epsilon = 2^{-64}$. Setting

$\tau_l = 0$ means that we don't insist on the average mining time to be above any particular positive time. The probability that mining a block takes more than $th = 300\,\text{s}$ is bounded by 10^{-9}. And the probability that more than one miner finds PoW within $\mu = 1/10000\,\text{s}$ is bounded by 0.001, which we also use as bound for winning 6 consecutive mining races. The algorithm reports the top $k = 5$ optimal tuples – and reports fewer if there are no 5 feasible tuples. The remaining constants for robustness are as given in Fig. 4.

Let us now specify some values of τ_u of interest. As reported in [5], Bitcoin is believed to handle up to 7 transactions per second (although this can be improved [6]), Paypal at least 100 transactions per second (which we take as an average here), and Visa anywhere between 2000 and 7000 transactions per second on average. By *transactions per second* we mean that blocks are mined within a period of time consistent with this. Of course, this depends on how many transactions are included in a block. For sake of concreteness and illustration, we take an average number of transactions in a Bitcoin block, as reported for the beginning of April 2016, that is 1454 transactions.

For a Bitcoin style rate, but in our *governed* setting, this means that a block is mined in about $1454/7 \sim 207.71\,\text{s}$. Since $T \cdot E^s(noR)$ is the expected (average) time to mine a block, we can model that we have 7 transactions per second on average by setting $\tau_u^{Bitcoin}$ to be 1454/7. Similarly, we may compute τ_u^{PayPal} and τ_u^{Visa} based on respective 100 and 7000 transactions per second:

$$\tau_u^{Bitcoin} = 1454/7 \qquad \tau_u^{PayPal} = 1454/100 \qquad \tau_u^{Visa} = 1454/7000 \tag{20}$$

Experimental Results. We now discuss the results of our experiments. Each experiment is conducted in three different configurations:

C1 constants in as Table 1, i.e. $\delta = 10^{-9}$, $\delta_2 = \delta_3 = 0.001$
C2 smaller δ, that is $\delta = 2^{-64}$, $\delta_2 = \delta_3 = 0.001$
C3 smaller δ and δ_3, that is $\delta = 2^{-64}$, $\delta_2 = 0.001$, and $\delta_3 = 0.0001$.

Transactions per Second as in Bitcoin, PayPal, and Visa. We show in Table 2 output for the top 5 optimal robustly feasible tuples for the various values of τ_u in (20) for configuration C1. We see that all three transaction rates can be realized with 18 miners and a 48-bit shared nonce space in our governed setting, and this gives each miner a nonce space of about 43 bits. The achievable level of difficulty (within the uncertainty in u_s and u_d) ranges from 37 to 41 for both the Bitcoin style rate and the PayPal style rate. For the Visa style rate, the feasible levels of difficulty are 34 and 35. For the optimal tuples reported in Table 2, the value of r remains feasible whenever $48 \leq r \leq 64$. Note that these results also imply that, for all three rate styles, feasibility requires at least 18 miners.

Let us run this experiment in configuration C2. This models that the probability of mining to take more than 300 s is very small. We now only report the changes to the results shown in Table 2 for the top rated, optimal tuple. For $\tau_u^{Bitcoin}$, the level of difficulty drops from 41 to 40 but there are still 18 miners and a shared nonce space of 48 bits. This tuple $(s, r, d) = (18, 48, 40)$ is also

Table 2. Output for top 5 optimal tuples for our robust optimization problem run in configuration C1 and with values τ_u as listed in (20): 5 optimal tuples are found for $\tau_u^{Bitcoin}$ and τ_u^{PayPal}, i.e. at least 5 values of d are feasible. The problem has two feasible levels of difficulty for τ_u^{Visa}. Costs are rounded up for three decimal places

$\tau_u^{Bitcoin}(s, r, d, cost)$	$\tau_u^{PayPal}(s, r, d, cost)$	$\tau_u^{Visa}(s, r, d, cost)$
(18, 48, 41, 54004.4)	(18, 48, 41, 54004.4)	(18, 48, 35, 54000.07)
(18, 48, 40, 54002.2)	(18, 48, 40, 54002.2)	(18, 48, 34, 54000.035)
(18, 48, 39, 54001.1)	(18, 48, 39, 54001.1)	
(18, 48, 38, 54000.55)	(18, 48, 38, 54000.55)	
(18, 48, 37, 54000.27)	(18, 48, 37, 54000.27)	

optimal for τ_u^{PayPal} now, whereas the optimal tuple $(s, r, d) = (18, 48, 35)$ for τ_u^{Visa} from configuration C1 remains to be optimal for C2.

Next, we run this experiment for configuration C3, also decreasing the probability that corrupt miners can win 6 consecutive mining races. For $\tau_u^{Bitcoin}$ and for τ_u^{PayPal}, the top 5 optimal tuples are $(s, r, d) = (24, 49, d)$ where $36 \leq d \leq 40$. In particular, this requires at least one more bit for the nonce space and at least 6 more miners. For τ_u^{Visa}, only tuples $(24, 49, 35)$ and $(24, 49, 34)$ are reported, so this also requires at least 24 miners and a 49-bit nonce space, where 35 and 34 are the feasible levels of difficulty.

We may explore the *feasibility boundary* for τ_u for configuration C2. The robust optimization problem is infeasible for $\tau_u = 0.06871$ but becomes feasible when τ_u equals 0.06872. In that case, the only feasible tuples are $(s, r, d) = (18, r, 34, 54000.03)$ where $48 \leq r \leq 64$.

Larger Transaction Rates per Second. Next, we want to vary the average number of transactions *ant* in a block from *ant* = 1454 to larger values. This is sensible for our use case as transactions only record a hash, which may be 8 bytes each. These results are seen in Table 3 for 50000 transactions on average in a block, running in the configuration C1. Let us discuss the impact of changing the *ant* in a block from 1454 to 50000. This has no impact when 7 or 100 transactions per second are desired. For 7000 transactions per second, this robust optimization problem still has the same s and r values in optimal tuples but the level of difficulty (which was 35 or 34) can now be between 36 and 40. This quantifies the security and availability benefits from packing more transactions into a block for mining throughput.

Let us now see how these results change when we run the experiment in configuration C2. Now, all three rate styles report the same optimal 5 tuples which are equal to the tuples listed in the rightmost column in Table 3: $(s, r, d) = (18, 48, d)$ where $36 \leq d \leq 40$. The results for configuration C3 are also idential for all three rate styles, they equal $(s, r, d) = (24, 49, d)$ where $36 \leq d \leq 40$. So this requires one more bit in the nonce space and at least 6 more miners.

Table 3. Output for top 5 optimal tuples for our robust optimization problem running in configuration C1 and with values τ_u given as 50000/7, 50000/100, and 50000/7000 (respectively). Results for the first two columns are identical with those in the first two columns of Table 2. The first 4 optimal tuples for $\tau_u = 50000/7000$ equal that last 4 of the 5 optimal tuples for 50000/7. Costs are rounded up for three decimal places

$Bitcoin \equiv 7(s, r, d, cost)$	$PayPal \equiv 100(s, r, d, cost)$	$Visa \equiv 7000(s, r, d, cost)$
(18, 48, 41, 54004.4)	(18, 48, 41, 54004.4)	(18, 48, 40, 54002.2)
(18, 48, 40, 54002.2)	(18, 48, 40, 54002.2)	(18, 48, 39, 54001.1)
(18, 48, 39, 54001.1)	(18, 48, 39, 54001.1)	(18, 48, 38, 54000.55)
(18, 48, 38, 54000.55)	(18, 48, 38, 54000.55)	(18, 48, 37, 54000.27)
(18, 48, 37, 54000.27)	(18, 48, 37, 54000.27)	(18, 48, 36, 54000.138)

Feasibility Boundary for Transaction Rates per Second. We repeat the last experiment by varying the *ant* from 50000 to half a million, in increments of 50000. We summarize these results as follows:

– *Configuration C1:* For all three rate styles and all transaction values in increments of 50000 up to 500000, the optimal tuples are the same: $(s, r, d) = (18, 48, d)$ where $37 \leq d \leq 41$.
– *Configuration C2:* For all three rate styles and all transaction values in increments of 50000 from 100000 up to 500000, the optimal tuples are the same: $(s, r, d) = (18, 48, d)$ where $36 \leq d \leq 40$. In contrast, for $50000/x$ where x is 7, 100 or 7000, we need at least a 49-bit nonce space and at least 24 miners.
– *Configuration C3:* For all three rate styles and all transaction values in increments of 50000 up to 500000, the optimal tuples are the same: $(s, r, d) = (24, 49, d)$ where $36 \leq d \leq 40$.

Range of Feasible Sizes for Nonce Space. We can compute and validate whether a robustly feasible tuple $(s, r, d, cost)$ has any other values r' for which $(s, r', d, cost)$ is robustly feasible. For example, for all the optimal tuples $(s, r, d, cost)$ we computed above, we conclude that we may change r to any r' satisfying $r < r' \leq 64$.

7 Discussion and Related Work

We made Assumption 1 only for appeal to the ROM model of the hash function used for mining. Implementations may violate this assumption, without compromising the predictive value of our models. Our Assumption 2 is at odds with Proof of Work as used in Bitcoin. But it does simplify the reasoning about mining behavior, and makes that more akin to reasoning about Byzantine fault tolerant consensus protocols [19]: for BFT protocols, network nodes are either honest (and so comply with protocol rules without incentives) or malicious (and so may behave in an arbitrary manner). Assumption 3 is related to the assumption that a communication network be *weakly synchronous*.

The mathematical model we proposed for Proof of Work did not specify details of the communication environment in which Proof of Work would operate. It would be of interest to extend our mathematical model with suitable abstractions of such a network environment, for example to reflect on upper bound on the communication delay between any two network points. This value could then be used to reflect Assumption 3 in finer detail in our model. Such an extension would also allow us to investigate whether consensus protocols can be simplified by providing Proof of Work as a service with specific behavioral guarantees.

Let us discuss related work next. In [6], a quantitative framework is developed for studying the security and performance of blockchains based on Proof of Work. This framework reflects a range of parameter values such as block size and those pertaining to network propagation, and allows to determine implications of such choices on security (double-spending and selfish mining in particular) and performance. It concludes that Bitcoin could well operate at a higher transaction rate while still offering its current level of security.

In [18], the quest for the "ultimate" blockchain fabric is discussed: getting secure blockchains that can process high transaction volumes (performance) but do this with thousands of nodes (security). Bitcoin offers good scalability of nodes, but its transaction rate does not scale. Dually, BFT protocols [4,10,19] can offer high transaction throughput rates but their communication complexity makes use of thousands of nodes impractical. The BFT state-machine replication protocol PBFT reported in [4] is designed to survive Byzantine faults in asynchronous networks – a proven impossibility that is circumvented with the aforementioned weak synchrony assumption in [4]. For a fixed number of $3f + 1$ nodes, this resiliency to faults can be realized if at most f nodes are faulty. A current leader proposes a new record to be added to the database, and three phases of communication arrive at final consensus of that addition. Views manage the transition of leadership, for example when timeouts suggest that the leader is not complying or not able to cooperate.

The cryptocurrency ByzCoin [9] combines ingredients from PBFT, from Bitcoin-NG (which separates leadership election and transaction verification aspects in the blockchain), and from Proof of Work to devise a hybrid blockchain: its *keyblock chain* uses Proof of Work to elect the next leader, whereas the *microblock* chain uses PBFT style consensus to add transactions during the current leadership. The network is open (nodes may join or leave), and the current consensus group is determined by stakes in mining that occurred within a current window of time. It uses a collective signing mechanism to reduce the communication complexity within the prepare and commit phases of the PBFT protocol.

A growing body of work uses blockchains for transactions that are not financial as such. In [20], e.g., a blockchain is used as a manager for access control such that this mechanism does not require trust in a third party. The architecture of our use case can also support transactions that are not financial.

The paper [11] discusses the work we reported in this paper in more detail. In particular, it includes a statistical validation of the random variables used in our mathematical model. In future work, we would like to support instances of our robust optimization problems in which not only d, s, and r are non-constant but also other parameters of interest – for example the time to compute a hash T or the period of time μ during which we want to avoid a conflict in the mining race. Current MINLP tools don't support such capabilities at present.

8 Conclusions

In this paper we considered blockchains as a well known mechanism for the creation of trustworthiness in transactions, as pioneered in the Bitcoin system [13]. We studied how blockchains, and the choice and operation of cryptographic puzzles that drive the creation of new blocks, could be controlled and owned by one or more organizations. Our proposal for such governed and more central control is that puzzle solvers are mere resources procured by those who control or own the blockchain, and that the solution of puzzles does not provide any monetary or other reward. In particular, solved blocks will not create units of some cryptocurrency and there is therefore no inherent incentive in solving puzzles.

We illustrated this idea with a use case in which financial transactions recorded within conventional accounts would be recorded as hashes within a governed blockchain and where it would be impractical to use hash chains, due to non-linearizability of transaction flows, and due to the lack of resiliency that such a solution would give to tampering with the blockchain.

We developed mathematical foundations for specifying and validating a crucial part of a governed blockchain system, the solving of cryptographic puzzles – where we focussed on Proof of Work. In our approach, owners of a blockchain system can specify allowed ranges for the size of the shared nonce space, the desired level of difficulty, and the number of miners used; and they can add mathematical constraints that specify requirements on availability, security, resiliency, and cost containment. This gives rise to MINLP optimization problems that we were able to express in analytical form, by appeal to the ROM model of cryptographic hash functions used for cryptographic puzzles.

We gave an algorithm for solving such MINLP problems for sizes of practical relevance and used it on some MINLP instances to demonstrate our capability of computing optimal design decisions for a governed Proof of Work system, where robust optimization models resiliency. This *mining calculus* also supports change management, such as an increase in mining capacity or mining resiliency.

Our approach and mathematical model are consistent with the consideration of several organizations controlling and procuring heterogeneous system resources, with each such organization having its bespoke blockchain, and with the provision of puzzle solving as an outsourced service. We leave the refinement of our mathematical models to such settings as future work. It will also be of interest to develop mathematical techniques for the real-time analysis of such blockchains, for example, to assess statistically whether the observed history of cryptographic puzzle solutions is consistent with the design specifications.

We hope our work will provoke more thinking about the design, implementation, and validation of blockchains that are centrally – or in a federated manner – owned and controlled and that may fulfil domain-specific needs for the creation of trustworthiness. We believe that many domains have such needs that the approach advocated in this paper might well be able to meet: existing financial processes and payment workflows, but also systems that have governed blockchains at the heart of their initial design.

Open Access of Research Data: The main algorithms needed for reproducing the experimental results reported in this paper were mere prototypes and not optimized very well. We make these algorithms available in a public repository bitbucket.org/lundbaek/haskell-governed-blockchain-optimiser.

Acknowledgements. This work was supported by the UK EPSRC with a Doctoral Training Fees Award for the first and second author and with projects [grant numbers EP/N020030/1 and EP/N023242/1]. We expressly thank Ruth Misener for having run some of our models on state-of-the-art global MINLP solvers – the tools ANTIGONE [12], BARON [15], and SCIP [17].

References

1. Ali, R., Barrdear, J., Clews, R., Southgate, J.: Innovations in payment technologies and the emergence of digital currencies. Q. Bull. (2014). Published by the Bank of England
2. Ben-Tal, A., Ghaoui, L.E., Nemirovski, A.: Robust Optimization. Princeton University Press, Princeton (2009)
3. Bird, R.: Thinking Functionally with Haskell. Cambridge University Press, Cambridge (2015)
4. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of OSDI 1999, pp. 173–186 (1999)
5. Danezis, G., Meiklejohn, S.: Centrally banked cryptocurrencies. CoRR abs/1505.06895 (2015)
6. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the ACM CCS 2016, pp. 3–16 (2016)
7. Horne, D.: Hash chain. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security, 2nd edn, pp. 542–543. Springer, Heidelberg (2011). doi:10.1007/978-1-4419-5906-5_780
8. Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.): 50 Years of Integer Programming 1958–2008 - From the Early Years to the State-of-the-Art. Springer, Heidelberg (2010)
9. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: Proceedings of the USENIX Security 2016, pp. 279–296 (2016)
10. Lamport, L.: Paxos made simple. ACM SIGACT News **32**(4), 18–25 (2001)
11. Lundbaek, L., D'Iddio, A.C., Huth, M.: Optimizing governed blockchains for financial process authentications. CoRR abs/1612.00407 (2016)

12. Misener, R., Floudas, C.A.: ANTIGONE: algorithms for continuous integer global optimization of nonlinear equations. J. Glob. Optim. **59**(2–3), 503–526 (2014)
13. Nakamoto, S.: Bitcoin : A Peer-to-Peer Electronic Cash System. Published under Pseudonym, May 2008
14. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press, Princeton (2016)
15. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**, 225–249 (2005)
16. Vigerske, S.: MINLP Library 2. http://www.gamsworld.org/minlp/minlplib2/html/
17. Vigerske, S.: Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming. Ph.D. in Mathematics, Humboldt-University Berlin (2012)
18. Vukolić, M.: The quest for scalable blockchain fabric: proof-of-work vs. BFT Replication. In: Camenisch, J., Kesdoğan, D. (eds.) iNetSec 2015. LNCS, vol. 9591, pp. 112–125. Springer, Cham (2016). doi:10.1007/978-3-319-39028-4_9
19. Wattenhofer, R.: The Science of the Blockchain. Inverted Forest Publishing (2016)
20. Zyskind, G., Nathan, O., Pentland, A.: Decentralizing privacy: using blockchain to protect personal data. In: Proceedings of the SPW 2015, pp. 180–184 (2015)