Bobtail: Improved Blockchain Security with Low-Variance Mining

George Bissias Brian N. Levine College of Information and Computer Sciences, UMass Amherst

ABSTRACT

Blockchain systems are designed to produce blocks at a constant average rate. The most popular systems currently employ a Proof of Work (PoW) algorithm as a means of creating these blocks. An unfortunate limitation of all deployed PoW blockchain systems is that the time between blocks has high variance. For example, Bitcoin produces, on average, one block every 10 minutes. However, 5% of the time, Bitcoin's inter-block time is at least 40 minutes.

In this paper, we show that high variance is at the root of several fundamental attacks on PoW blockchains. We propose an alternative process for PoW-based block discovery that results in an inter-block time with significantly lower variance. Our algorithm, called Bobtail, generalizes the current algorithm by comparing the mean of the k-lowest order statistics to a target. We show that the variance of interblock times decreases as k increases. Bobtail significantly thwarts doublespend and selfish mining attacks, and makes detection of eclipse attacks trivial and quick. For example, for Bitcoin and Ethereum, a doublespending attacker with 40% of the mining power will succeed with 53% probability when the merchant sets up an embargo of 1 block; however, when $k \geq 40$, the probability of success for the same attacker falls to less than 1%. Similarly, for Bitcoin and Ethereum currently, a selfish miner with 49% of the mining power will claim about 95% of blocks; however, when $k \ge 20$, the same miner will find that selfish mining is less successful than honest mining. We also investigate attacks newly made possible by Bobtail and show how they can be defeated. The primary costs of our approach are larger blocks and increased network traffic.

1 INTRODUCTION

Blockchain systems are designed to produce blocks at a constant average rate. The most popular systems currently employ a *Proof of Work (PoW)* algorithm as a means of creating these blocks [1], including Bitcoin Cash, Bitcoin Core, Ethereum [2], and Litecoin [3]. For example, Bitcoin and Bitcoin Cash produce one block every 10 minutes on average and self-adjust the difficulty of producing a block if too many or too few have been produced. Unfortunately, a limitation of all deployed PoW blockchain systems is that the time between blocks has high variance and the distribution of inter-block times has a very long tail. For example, 5% of the time, Bitcoin's inter-block time is at least 40 minutes. As we show, high variance enables doublespend [1, 4] and selfish mining [5] attacks, in addition to impeding the consistent flow of validated transactions through the system.

The high variance of inter-block times is a direct consequence of PoW algorithms. Generally, miners in all deployed systems craft blocks by repeatedly changing a nonce in the block header until the hash of that header is less than a target value t. In other words, the hash of each header is a sample taken randomly from a discrete

uniform distribution that ranges between [0, S], where $S = 2^b - 1$ and typically b = 256. A block is discovered when the *first order statistic* (i.e., the minimum value) of all sampled values is less than target t, 0 < t < S.

In this paper, we show that high variance is at the root of several fundamental attacks on PoW blockchains, including doublespend and selfish mining attacks. We propose an alternative process for PoW-based block discovery that results in an inter-block time with significantly lower variance. Our algorithm generalizes the current algorithm by comparing the mean of the k-lowest order statistics to a target. We show that the variance of inter-block times decreases as k increases. For example, if our approach were applied to Bitcoin, nearly every block would be found within 5 to 18 minutes; and the average inter-block time would remain at 10 minutes. As a result, doublespend and selfish mining attack efficacies are drastically reduced. The cost of our approach is larger blocks and increased network traffic. New attacks are possible, but we show how they can be thwarted. We call our approach Bobtail mining.

Problem Statement.

Imagine that the mining process is carried out for exactly h hashes during time interval I, generating hash values H_1, \ldots, H_h from [0, S] uniformly at random. Now define V_i to be the value of the ith order statistic at the end of I, i.e. $V_i = H_{(i)}$ in standard notation. Let W_k be a random variable representing the average value over the lowest k order statistics after h hashes.

$$W_k = \frac{1}{k} \sum_{i=1}^k V_i. {1}$$

 W_k constitutes the collective mining proof (*proof*, for short) for the entire network. Our Bobtail mining criterion says that a new block is discovered when a realized value of W_k meets the target t:

$$w_k \le t. \tag{2}$$

Notably, this approach is a generalization of current systems, which are the special case where k=1.

Our primary goals are therefore to show, given values of k > 1, that: (*i*) there is a significantly reduced inter-block time variance; (*ii*) the costs are relatively small, which include increases in block size and network traffic; (*iii*) selfish mining and doublespend attacks are significantly more difficult to carry out as k increases; and (*iv*) that new attacks made possible by setting k > 1 are easily mitigated.

Contributions

 We derive the statistical characteristics of our approach and validate each empirically. For example, we derive expressions for the expectation and variance of the Bobtail mining proof and the number of hashes performed for any value of k. Using these expressions, we quantify the reduction in variance of inter-block time expected for values of k > 1.

- We show that variance in block discovery time is reduced by O(1/i) when using k = i order statistics relative to the variance when k = 1 (the status quo).
- We show that Bobtail mining results in a lower rate of orphaned blocks.
- We demonstrate that low-variance mining significantly mitigates the threats to security posed by selfish mining and doublespend attacks. For Bitcoin and Ethereum currently (i.e., when k=1), an attacker with 40% of the mining power will succeed with 30% probability when the merchant sets up an embargo of 8 blocks; however, when $k \geq 20$, the probability of success falls to less than 1%. Similarly, for Bitcoin and Ethereum currently, a selfish miner with 45% of the mining power will claim about 95% of blocks; however, when $k \geq 20$, the same miner will find that selfish mining is less successful than honest mining.
- We show that new intra-block withholding attacks are possible for Bobtail. However, by carefully designing a rewards scheme for mining, these attacks are mitigated.

2 RELATED WORK

Our approach is related to previous results in proof-of-work, cryptographic puzzles, and improvements to blockchain architectures.

Proof of work. A large number of papers have explored applications of proof-of-work. Dwork and Naor [6] first suggested proof-of-work (PoW) in 1992, applying it as a method to thwart spam email. A number of subsequent works similarly applied PoW to thwarting denial-of-service (DoS) attacks [7–12]. Our approach can be adopted into many of these past works to improve computational variance. Jakobsson and Juels [13] and Jules and Brainerd [14] examine the security properties of PoW protocols, and base their theorems on the average work required; our approach would provide stronger guarantees under their theorems since the variance is lower. Laurie and Clayton [15] examine the practical limitations in deploying PoW solutions in DoS scenarios.

Douceur [16] noted in 2002 that proofs of work can mitigate Sybil attacks. Also in 2002, Back [17] applied PoW to cryptocurrencies. Back noted the high variance of computational PoW and regarded it as an open problem. Nakamoto's Bitcoin [1] built on these ideas.

In 2003, Abadi et al. [18] suggested memory-bound functions as a better foundation for avoiding the variance in CPU resources among users. Indeed, the ETHASH [19] PoW algorithm in Ethereum [2] adopted a PoW function that requires more memory than is economically profitable for custom ASICSs. In contrast, our goal is to reduce the variance of the entire network's time to solve a PoW problem, and it is not to increase egalitarianism or increase participation by eschewing specialized hardware. In any case, our approach is applicable to ETHASH.

Coelho [20] is the work is closest to ours in terms of goals. That work proposes a PoW puzzle based on Merkle trees that requires an exact number of steps and therefore has no computational variance. Without variance, the same miner would always win, and therefore the method is unsuitable for blockchains.

Bitcoin-NG [21] also offers low-variance transaction announcements via PoW-based leader election. However, because inter-leader time variance would still be from an exponential distribution, unlike

Block

Header version v prior o difficulty d timestamp e	subnonce n_1 transaction root m_1 support s_1 proof root p bounty root b	$\begin{array}{c} \textbf{proof set } i\text{:} \\ \\ \text{prior } o \\ \\ \text{merke root } m_i \\ \\ \text{address } a_i \\ \\ \text{support } s_i \end{array}$
Transactions coinbase $a_1, \ldots,$ coinbase a_k transaction 1, transaction 2,		nonce N_i It must be that $s_i \ge V_I$ for all $i \ge 2$; and $\frac{1}{k} \sum_{i=1}^k V_i \le target$ where $N_i = h(v, d, e_i, n_i,)$ and $V_i = h(o, m_i, a_i, s_i, N_i)$
Proof package proof set 2, proof set 3,		
Bounties bounty 1, bounty 2,		
Signature creator of 1OS signs block with a_I		

Figure 1: Bobtail blocks are a superset of existing PoW schemes. They contain a link to a prior block, a timestamp, the current difficulty, and the Merkle root of a set of transactions; they also add proof sets contributed by other miners as well as bounties that prove claims about individual proof sets.

Bobtail, Bitcoin-NG does not reduce doublespend, selfish mining, or eclipse attack vulnerability. Furthermore, unlike our approach, Bitcoin-NG sets up the elected leader as a single point of failure.

Blockchains without PoW. Several newer blockchains are not based on computational proof of work, and our solution does not apply to them. These include proof-of-storage [22], proof-of-stake [23, 24], and blockless [25] schemes. However, almost all wealth stored in cryptocurrencies are in computational PoW blockchains that our approach does apply to, including Bitcoin, Bitcoin Cash, Litecoin, Ethereum, and Ethereum Classic.

3 PROTOCOL

In this section, we provide a high-level overview of the Bobtail protocol and its fundamental features.

Blocks. Bobtail blocks, illustrated in Figure 1, consist of a block header \mathcal{H} , a set \mathcal{T}_1 of valid, previously unconfirmed transactions, a proof package K, bounties B, and a signature S. Block header ${\cal H}$ contains the same fields as a conventional block header such as the one used in Bitcoin or Ethereum, and three additional fields, which are described below. Proof package K is a collection of k*proof sets*, where each proof set \mathcal{P}_i contains a payout address for the miner, values necessary for creating valid proof of work, and other values used for thwarting attacks. Proof sets are hashed to create PoW, i.e. $h(\mathcal{P}_i) = V_i$. The sets are ranked so that V_1 is defined as the smallest value or first order statistic (1OS, for short). Each bounty $\mathcal{B}(T, \mathcal{T}) \in \mathcal{B}$ is a Merkle proof that some transaction T is in \mathcal{T} for some transaction set $\mathcal{T} \in \{\mathcal{T}_1, \dots, \mathcal{T}_k\}$; it is used to prevent double spend attacks as we describe in detail below. Finally, ${\mathcal S}$ is the signature of ${\cal H}$ and it must be generated with the private key that matches the payout address in \mathcal{P}_i .

Mining. Bobtail mining is a generalization of the procedure implemented in conventional PoW blockchains. Each miner seeks to receive coinbase reward by becoming one of the *k* proof sets included in $\mathcal{K} = [\mathcal{P}_1, \dots, \mathcal{P}_k]$. Proof set \mathcal{P}_i has fields (o, m_i, a_i, s_i, N_i) , where o is the hash of the previous block header, m_i is the root of a Merkle tree containing transactions \mathcal{T}_i , a_i is the miner's payout address, s_i is the supporting proof for \mathcal{P}_i , and $N_i = h(\mathcal{N}_i)$, the nonce. In contrast to typical PoW systems, the nonce is more than just a random number; it contains values that can be omitted from the blockchain for all proofs except the 1OS. Specifically, $\mathcal{N}_i = (v, d, e_i, n_i, \ldots)$, where v is protocol version, d is current difficulty, e_i is a timestamp, n_i is a nonce, and we allow zero or more optional arguments. Finally, header \mathcal{H} has fields $(v, o, d, e, N_1, m_1, s_1, p, b)$, where e is a timestamp, p is the root of a Merkle tree containing every V_i corresponding to $\mathcal{P}_i \in \mathcal{K}$, b is the root of a Merkle tree containing $h(\mathcal{B}(\mathcal{T}, T))$ for each $\mathcal{B}(\mathcal{T}, T) \in \mathcal{B}$, and the remaining fields match the corresponding values from \mathcal{P}_1 .

Like other PoW blockchains, miners select new nonces and generate proofs continuously. In Section 6, we show how a miner can precisely determine the probability that any given proof will eventually be included in the mining package. Once a proof is discovered having sufficient probability of inclusion, the corresponding values $\mathcal{H}, \mathcal{P}_i$, and \mathcal{T}_i are propagated. Because of their large size and high degree of redundancy, transaction sets \mathcal{T}_i can be propagated using Graphene [26] or the weak block protocol [27] to greatly reduce associated network overhead.

The difficulty d for each block is adjusted roughly once every two weeks using the same algorithm deployed in Bitcoin¹. In short, the mean block time for the last 2016 blocks is used to estimate the actual difficulty at which the miners were operating; then the difficulty is adjusted up or down in order to ensure that the expected block time is 10 minutes if miners continuing mining at the same rate. At a given difficulty d, the target t is derived from d in the same manner that it is for Bitcoin², which involves translating integer d into a threshold 256-bit arithmetic value (i.e., one that supports arithmetic operations). We say that a block is assembled when (i) the mean of the k proofs, $V_1, \ldots V_k$, is less than or equal to t; (ii) the package is signed by the miner who generated the 1OS using address a_1 ; (iii) supports $s_2, \ldots s_k$ are greater than or equal to V_1 .

The coinbase transaction of the block creates new coin that is distributed to addresses a_1,\ldots,a_k according to the scheme described in Section 8. If the coinbase of the new block does not contain the correctly awarded coinbase, then the block is ignored by other miners. The assembled block, comprised of \mathcal{H} , \mathcal{T}_1 , \mathcal{K} , \mathcal{B} , and \mathcal{S} is propagated throughout the network. Receivers validate that \mathcal{S} was generated by hashing \mathcal{H} with the private key corresponding to payout address a_1 . They also validate \mathcal{T}_1 , \mathcal{K} , and \mathcal{B} against \mathcal{H} . If validation succeeds, then the block is added to the blockchain.

Bounties Bobtail employs a *bounty* system to disincentivize the use of incompatible transactions (those spending the same UTXO) in transactions sets \mathcal{T}_i . Bounty $\mathcal{B}(\mathcal{T}_i, T')$ contains a Merkle proof that transaction set \mathcal{T}_i contains transaction T'. The Merkle proof

is simply all nodes along a path from the Merkle root m_i to the leaf containing T'. When proof set \mathcal{P}_i is disseminated in the network, it must be accompanied by Merkle root m_i and associated transactions \mathcal{T}_i . The recipient checks that the transactions in \mathcal{T}_i are compatible with the transactions in her own set \mathcal{T}_j . If she discovers incompatible transaction $T' \in \mathcal{T}_i$, then she can create bounty $\mathcal{B}(\mathcal{T}_i, T')$. Later, if one of her proofs is the 1OS, she will include all bounties that implicate transactions in conflict with \mathcal{T}_1 .

Fork-choice rule. If multiple miners generate proofs with value low enough to mine a block with the same parent, then there can arise ambiguity over which extends the *main chain*, i.e. the chain that honest miners will continue to extend. To avoid this ambiguity, we define the main chain to be the one comprising the most *aggregate work*, from the genesis block up to the tip; all competing chains are *orphaned*, i.e. ignored by honest miners. Aggregate work is calculated as the sum of inferred hashes, S/w_k , over each block, where S is the size of the hash space and w_k is the value of the mining statistic. In general, this fork-choice rule ensure that blocks with lower average proof values will be favored over those with higher values. Note that because proof sets reference a specific parent block, they can only be shared between child blocks having the same parent. And according to our fork-choice rule, ultimately the child block with the lowest value w_k will extend the main chain.

Additional rules. In order to reduce the number of orphaned blocks (discussed in Section 7.1) and thwart various attacks (see Section 4), miner M will adhere to the following rules. (i) M rejects proof package \mathcal{K} if V_1 is higher than the lowest proof she has seen announced on the network. (ii) When assembling a proof package as author of the 1OS, M will include proofs from other miners in the same order she received them from the network. Specifically, M begins by identifying all sets of k proofs S_1, \ldots , each with mean value below t. Let t be the maximum reward value, across all S_i , that would be returned to M if the the given set was assembled into a block. She discards any sets that do not return reward value t, and then assembles the proof package from the remaining set with the earliest average proof receipt time.

4 THREAT MODEL

Doublespend and selfish mining. Doublespend [1, 4] and selfing mining [5] attacks are the two most fundamental attacks on blockchains [28, 29]. In both, attackers attempt to mine a fork of the blockchain that is longer than the honest miners' branch. Because the attacker is assumed to have a minority of the mining power, in expectation, the attacker cannot create a longer branch than that of the honest miners. However, just like a person visiting a casino, the attacker is seeking a short-term win. He is attempting to get lucky and find a series of blocks quickly while the honest miners are relatively unlucky and discover blocks slowly, despite the larger amount of mining power. Intuitively the success of the attacks lies in leveraging the inherent variance of mining.

Proof withholding. A proof withholding attack is unique to Bobtail and involves miner A declining to publish some subset of her proofs immediately after they are generated. Instead, A withholds the proofs in order to gain an informational advantage over the remaining miners, M. While A sees all proofs, an honest miner in M sees only proofs generated by members of M. A hopes that this

¹https://github.com/bitcoin/bitcoin/blob/78dae8caccd82cfbfd76557f1fb7d7557c7b5edb/src/pow.cpp#L49

²https://github.com/bitcoin/bitcoin/blob/78dae8caccd82cfbfd76557f1fb7d7557c7b5edb/ src/pow.cpp#L80

advantage will allow her to assemble some proof packages with more than her fair share of proofs and ultimately lead to an increase in her total reward.

Zero-cost zero-confirmation (ZCZC) doublespend. When carrying out a zero-confirmation doublespend attack, the attacker seeks to displace payment transaction T with an incompatible transaction T', which spends the same UTXO. The ZCZC doublespend is a variant of the conventional attack where the attacker reuses proofs between multiple branches of the blockchain to this end. To do so, he releases payment transaction T to a merchant and the Bitcoin network at large. Simultaneously he mines with transaction set \mathcal{T}_i containing T'. If he is lucky enough to mine the 1OS, then $j = 1, T_i$ becomes the canonical transaction set for the block, T'is confirmed, and the merchant payment is negated. But even if he does not mine the 1OS, he can release his proofs containing \mathcal{T}_i at some point before the block is mined and still receive a reward for those that are included in the proof package. Note that this attack also applies to doublespends where the attacker attempts to produce a competing branch of arbitrary length z, however, proofs can only be reused during the creation of the first block on the competing branch.

Denial-of-reward. It is possible for an attacking peer in the network to cheaply weaponize the bounty process against miners. Suppose that peer A creates incompatible transactions T and T'such that both spend the same UTXO. Suppose further that she disseminates T to roughly half the miners and T' to the remaining miners. Then, it is likely that the first group of miners will regard T as a legitimate transaction and T' as a doublespend, while the opposite would be true for the other miners. Thus, in expectation, roughly half the proofs will use a transaction set containing T and half will use a transaction set containing T'. Furthermore, miners will construct bounties for whichever transaction they believe to be a doublespend. Eventually, a proof package will be assembled (containing transaction set \mathcal{T}_1) by one of the miners M, who holds the 1OS. Assume, without loss of generality, that \mathcal{T}_1 contains T. Then miner M is incentivized to include all bounties against proofs whose Merkle trees contain T'. As a result, the miners of those proofs, roughly half the total in expectation, will lose their reward.

5 PROPERTIES OF THE K-OS CRITERION

Recall from Section 1 that V_i represents the ith lowest hash value achieved after h hashes are performed during interval I. Next, define X_i to be the number of hash intervals required for the ith order statistic to fall below target v when h hashes are performed per interval. In Appendix A we show that V_i and X_i are gamma distributed with shape parameter i, only differing in scale parameter. Specifically, $V_i \sim \mathsf{Gamma}(t;i,v)$ and $X_i \sim \mathsf{Gamma}(x;i,1/r)$ where v is the expected value of the minimum hash during interval I, and r is the rate at which hashes are generated below v during interval I. The two parameters are further shown to be related by the following equation

5.1 Target Adjustment

Assuming ideal difficulty adjustment, the mining target t_k corresponding to mining statistic W_k is set so that $t_k = E[W_k]$. In Appendix A, we defined v as the value of the minimum hash during interval I, which implies that $v = E[W_1] = t_1$. That is to say, when k = 1, the target is equal to v. But this same target will produce later blocks for k > 1 because W_j includes higher order statistics than W_i when j > i, meaning that the network must produce more hashes in expectation in order for W_j to achieve the same value as W_i . Therefore, the goal in this section is to determine how to select t_k as a function of v such that the number of hash intervals required to ensure $E[W_1] < v$ is the same as the number required to ensure $E[W_k] < t_k$.

To that end, we begin by deriving an expression for the expected number of hash intervals required for W_k to fall below target t_k , which we find is easiest to determine relative to a slightly different statistic. The sample average of the X_i (analogous to W_k) is an obvious candidate, however, it actually gives the average number of intervals required for each order statistic V_i to independently fall below value v given that h hashes are performed during each interval. Naturally, more intervals are required as i increases because larger order statistics have higher expected value. So what we would like is to tune each X_i so that they are all expected to finish at the same time.

Recall from Appendix A that $E[X_i] = \frac{i}{r}$. Thus, by dividing each X_i by i we can align the X_i to share the same expected value. To that end, define the *normalized* hash interval count by $X_i' = X_i/i$; and analogously define $V_i' = V_i/i$. Each X_i' is interpreted as the number of hash intervals required for V_i to fall below iv (or for V_i' to fall below v). We have $E[X_i'] = \frac{1}{r}$ for all $i \geq 1$, which implies that if we tune the hash threshold to be iv for V_i , then we expect all V_i to cross below their threshold after 1/r hash intervals.

THEOREM 1: In expectation, 1/r hash intervals are required to ensure that $V_i/i < v$ for all i.

The following result is also straightforward.

THEOREM 2: The expected value of W_k is

$$E[W_k] = \frac{k+1}{2}v$$

PROOF:

$$E[W_k] = E\left[\frac{1}{k}\sum_{i=1}^k V_i\right]$$

$$= \frac{1}{k}\sum_{i=1}^k E[V_i]$$

$$= \frac{1}{k}\sum_{i=1}^k iv$$

$$= \frac{k+1}{2}v.$$
(4)

$$v = r\frac{S}{h}. (3)$$

And from Theorem 2 it follows that

$$E\left[\frac{1}{k}\sum_{i=1}^{k}V_{i}'\right] = \frac{1}{k}\sum_{i=1}^{k}E\left[V_{i}'\right]$$

$$= v$$

$$= E[W_{1}]$$

$$= \frac{2}{k+1}E[W_{k}].$$
(5)

Equation 5 implies that the expected sample average of values V_i' is equivalent to the expected value of W_1 , which is in turn equivalent to scaling the expected value of W_k by a factor of 2/(k+1). Recalling that $t_i = E[W_i]$ when targets are tuned optimally, Theorem 1 and Equation 5 give us the following.

THEOREM 3: In expectation, the same number of hash intervals are required to ensure $W_k < t_k$, for all k > 0, provided that t_k is chosen such that

$$t_k = \frac{k+1}{2}v. (6)$$

5.2 Estimating Hash Rate

Because V_i and X_i have the same distribution, up to a change of variables, Equation 5 also implies

$$E\left[\frac{1}{k}\sum_{i=1}^{k}X_{i}\right] = \frac{k+1}{2}\frac{1}{r}$$

$$= \frac{k+1}{2}E\left[\frac{1}{k}\sum_{i=1}^{k}X'_{i}\right].$$
(7)

Therefore, assuming that targets have been adjusted so that $t_k = \frac{k+1}{2}v$, the following is a natural choice of estimator for the overall number of hash intervals required to ensure $W_k < t_k$.

$$Y_k = \frac{2}{k+1} \left(\frac{1}{k} \sum_{i=1}^k X_i \right).$$
 (8)

Note that the estimator Y_k holds the property

$$\frac{2}{k+1}E[W_k] = E[W_1] = \upsilon = \upsilon r E[Y_1] = \upsilon r E[Y_k].$$

In other words, $E[W_k]$ is related to $E[Y_k]$ by the constant transformation $\frac{2}{\upsilon r(k+1)}$. Equation 7 shows that Y_k is the sample average of random variables X_i' , each representing the number of hash intervals required for V_i to fall below $i\upsilon$. On the other hand, Theorem 1 establishes that $E[V_i] = i\upsilon$ after mining for 1/r hash intervals. Therefore, Y_k is biased to the extent that the sample average of the V_i 's deviates from its expected value. But as k increases, the law of large numbers ensures that individual departures from the mean for each V_i are cancelled out. Thus Y_k is a consistent estimator in the limit that k approaches infinity. From this argument and Equations 7 and 8 we have the following.

THEOREM 4: Assuming that $t_k = \frac{v(k+1)}{2}$, Y_k is a consistent estimator of the expected number of hash intervals required for W_k to fall below t_k with

$$E[Y_k] = \frac{1}{r},\tag{9}$$

where r = vh/S.

5.3 Improvement in Variance

We next turn our attention to quantifying the improvement in mining time variance that is realized by using the k-OS criterion over the 1-OS criterion. In Section 5.2 we established that statistic Y_k is a consistent estimator of the number of hash intervals required for mining statistic W_k to fall below target $t_k = \frac{k+1}{2}v$. Here we measure the change in variance of Y_k as k increases, while holding its expected value constant.

THEOREM 5: For fixed expected block discovery time, variance decreases by fraction $\frac{8k+4}{6(k^2+k)} = O\left(\frac{1}{k}\right)$ when using mining statistic W_k instead of W_1 .

PROOF: Theorem 4 establishes that block discovery time Y_k is the same in expectation for all mining statistics W_k provided that $t_k = \frac{v(k+1)}{2}$. Therefore, the ratio of variance in Y_k to the variance in Y_1 estimates the reduction in block time variance due to Bobtail.

$$\frac{Var[Y_k]}{Var[Y_1]} = \frac{Var\left[\frac{2}{k+1}\left(\frac{1}{k}\sum_{i=1}^{k}X_i\right)\right]}{Var[X_1]} \\
= \frac{4}{(k+1)^2} \frac{Var\left[\left(\frac{1}{k}\sum_{i=1}^{k}X_i\right)\right]}{Var[X_1]} \\
= \frac{4}{(k+1)^2} \frac{(k+1)(2k+1)}{6k} \\
= \frac{8k+4}{6(k^2+k)} \\
= O\left(\frac{1}{k}\right), \tag{10}$$

where we use the expression for $Var\left[\left(\frac{1}{k}\sum_{i=1}^{k}X_{i}\right)\right]$ derived in Appendix 5.

Figure 2 shows the distribution of Y_k when $t_k = v(k+1)/2$ so that $E[Y_k] = E[Y_1]$. The plot shows the cumulative distribution function (CDF) based on the results of a Monte Carlo simulation³. As the plots illustrate, the use of Bobtail mining results in a significant decrease in variance for discovering new blocks.

6 NETWORK OVERHEAD

When the mining criterion is W_k , $k \gg 1$, it is not efficient for each miner to send proof of work every time she finds a hash value lower than her previous best. A slight improvement to that scheme is for her to send proof of work only when her hash value is lower than the lowest k hashes produced by all miners cumulatively. But even this approach will result in a large amount of network traffic early in the mining process because hash values are generated uniformly at random throughout the mining interval (see Theorem 7); therefore the k lowest are unlikely to be generated in a short period of time at the beginning.

To improve network efficiency significantly we require that miners do not send or forward proof sets unless there is a high probability that the resulting proof will be part of the next block. To create such a filter, we find the largest proof value that has a probability p

³We will release all simulation source code used in this paper at camera ready.

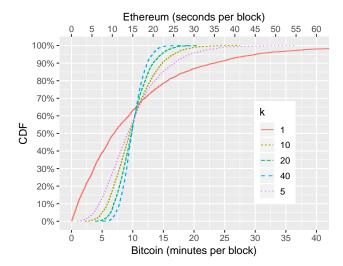


Figure 2: Results of a Monte Carlo simulation showing the CDF of Y_k , the block discovery time under mining statistic W_k , where k varies for each curve and target t_k is chosen so that $E[Y_k] = E[Y_1]$. Each plot's x-axis is shown in terms of the minutes per block for Bitcoin (bottom axis) and seconds per block for Ethereum (top axis).

of being part of the block given k; where typically, p = 0.999999. The following theorem lets us determine the expected number of proofs forwarded per block.

THEOREM 6: For a given threshold probability p, the number of proofs announced to the network is y = Quantile-Gamma(p; k, 1).

PROOF: We know from Theorem 9 that the kth order statistic V_k has distribution Gamma(k, v). Let

$$G(x;k,v) = P(V_k \le x) = p$$

be the CDF of V_k and $G^{-1}(p;k,v)$ be equal to Quantile-Gamma(p;k,v), which returns x, the value for which G(x;k,v)=p. Note from Eq. 3 that v=S/h when t_k is tuned for blocks to be generated in time I (i.e. r=1). We expect h hashes per block interval, and each has probability x/S of being below the threshold. Therefore, the random variable representing the number of proofs forwarded by all miners follows distribution Binomial(n=h,p=x/S), which has expectation:

$$\begin{array}{ll} \frac{hx}{S} & = & \frac{1}{v} \cdot \text{Quantile-Gamma}(p;k,v) \\ & = & \text{Quantile-Gamma}(p;k,v/v) \\ & = & \text{Quantile-Gamma}(p;k,1) \end{array} \tag{11}$$

Notably, the value is quite low, and it is independent of h the expected number of hashes required to mine a block and the size of the hash space, S. We can also use a Chernoff bound for the binomial distribution to bound the deviation in the number of messages M.

Let $y = \frac{hx}{S}$ where $x = G^{-1}(p; k, v)$ as defined in the proof above.

$$P(M \ge (1 + \epsilon)y) \le e^{\frac{-y\epsilon^2}{2+\epsilon}} \tag{12}$$

This is a tight bound, and it decreases exponentially with y and similarly with k. For example, when k=2 and p=0.999999, then $y\approx 16.7$, and we see that $P(M>1.9y)\leq 0.0095$. For k=3, the probability decreases to 0.004, and so on.

7 INCREASING CONSENSUS

Even when all miners operate honestly, current blockchain systems frequently suffer from orphaned blocks during their operation that diminish security and delay consensus. Orphans are generated when the announcement of a new block by one miner takes time to propagate to all other miners. In the interim, a second miner may produce a valid block. At that point, the subset of miners who received the first block first will attempt to build upon it, and the remaining miners will build upon the second. Eventually the blockchain will fork on just one of those blocks, orphaning the other. If the set of transactions in the two blocks is not the same, then consensus is delayed. While the occurrence of orphans in Bitcoin is relatively low, Ethereum's use of a 15 second average block discovery time increases its orphan rate significantly.

In Appendix A, we show that X_i , the number of block intervals required to mine the ith order statistic, has distribution $\mathsf{Gamma}(i,1/r)$, where r is the rate at which hashes are generated below the target. It follows that X_1 represents the block inter-arrival time, and it has distribution $\mathsf{Exponential}(1/r) = \mathsf{Exponential}(T)$, where T is the expected block time. Therefore, in existing POW blockchains, the probability that one or more other blocks will be discovered during propagation time τ is bounded by $1 - \frac{1}{e^{\tau/T}}$. Note that this bound is pessimistic in that it assumes the worst case scenario where the author of the first block has a negligible percentage of the total network hash rate.

In this section, we examine the orphan rate associated with Bobtail mining compared to Bitcoin and Ethereum. We show that when miners follow the Bobtail protocol, orphans are strictly less likely.

7.1 Orphan Prevention Measures

The principal cause of orphans in Bobtail is the fact that, once more than k proofs have been disseminated, there exist a combinatorial number of k-element subsets of those proofs. Thus at the time when there exists some subset of k proofs whose mean falls below target t, there is a reasonable chance that some other subset also exists (or will exist relatively soon). Fortunately, the proof package rules introduced in Section 3 greatly reduce the number of valid subsets. First, all proofs must be tethered to a $supporting\ proof$, the latter of which should be the smallest proof previously received by the miner. Second, no support in the proof package can have value less than the 1OS. And third, the block must be signed by the private key used to generate the 1OS.

Together, these conditions ensure that if at least one of the k proof sets in the proof package points to the 1OS as support (excluding the 1OS itself), then the creator of the 1OS is the *only* miner capable of assembling that proof package. Conversely, although another miner, say the one who generated V_2 , might be capable of

6

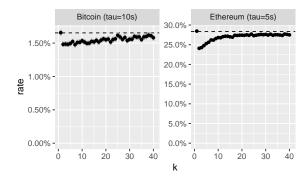


Figure 3: A simulation of Bobtail's orphan rate when proofs and blocks propagate with constant delay of $\tau=10$ seconds and the interblock time is T=600 seconds. The orphan rate of Bobtail is at or below the expected orphan rate, $1-1/e^{\tau/T}$ for k=1, which is shown as a dashed line. Similar results hold for Ethereum where $\tau=5$ and T=15 seconds. Error bars represent a 95% c.i.

collecting a set of proofs that exclude the 1OS but still having mean below target t, that miner cannot assemble a proof package if even a single proof set includes the 1OS as support.

7.2 Performance

We ran a discrete event simulator to determine the efficacy of the orphan prevention measures described in Section 7.1. The simulation includes only honest miners: once it has received a valid proof package, an honest miner does not release a competing proof package of its own. We evaluate attacks on Bobtail subsequently.

The simulation generates blocks by repeatedly selecting values uniformly at random between 0 and 2^{32} . The smallest k values are used as a candidate block given a pre-set target value. The propagation delay of new proofs and blocks is τ seconds. Once a block is found, we assume that the authoring miner drops out and her mining power is replaced by a new honest miner; i.e., the hash rate does not change. For τ seconds, the miners continue seeking a new block following the rules in Section 3. For example, if they find a block is possible with a higher 1OS, they will not release the new block.

Figure 3 (left) shows the results for a Bitcoin-like scenario where the inter-block time is targeted at T=600 seconds and the propagation delay is t=10 seconds. The orphan rate for k=1 follows the expected Poisson result, shown as a dashed line. The experiment shows that Bobtail has an orphan rate at or below the k=1 rate, in terms of statistical significance. Figure 3 (right) shows the same result for a simulation of Ethereum where $\tau=5$ and T=15 seconds, respectively.

8 INCENTIVIZING HONEST BEHAVIOR WITH REWARDS

In this section, we show that there exists a *reward scheme* — the payout of fees and coinbase — that incentivizes miners to (*i*) continue mining for increased reward (rather than stopping once any proof is discovered) (*ii*) use the lowest proof they know of as support, and

(iii) immediately broadcast all sufficiently low proofs. In this section, we evaluate the following reward structure with respect to all three properties assuming honest miner behavior (attack scenarios are considered in Section 9).

- To each proof in the proof package, we assign *primary reward R*, which is the same for each proof in a given package, but may vary from block to block.
- To each proof whose support is the 1OS, we award a *bonus reward B*, which is again the same for every proof pointing to the 1OS, but may vary by block.

One of the major goals of this section is to determine the expected primary and bonus rewards accrued by an honest miner across all proofs in a given block. We further derive the expected *total reward T*, which is the sum of expected primary and bonus rewards for a miner following the honest strategy.

8.1 Idealized Analysis

We begin with a basic result that is useful in contemplating reward distribution.

THEOREM 7: In expectation, a fraction x of the mining power will generate a fraction x of all proofs as well as a fraction x of the k-lowest order statistics.

PROOF: Without loss of generality, assume a single miner M owns fraction x of the mining power. All hashes generated are uniformly distributed throughout space S. Therefore, of all the hashes that fall within an interval of S, miner M will own fraction x. The interval [0, S] contains all proofs; it is therefore clear that M will generate fraction x of all proofs. Moreover, the set of all proofs K that are at or below the kth order statistic defines an interval, [0, k-OS]. Thus M will own fraction x of proofs in K as well, which constitutes fraction x of the set of lowest k order statistics.

We next analyze the reward payout with respect to our desired mining properties under the assumption that all miners behave honestly, i.e. according to the protocol. Consider a miner M who possesses fraction x of the total mining power. According to Theorem 7, M can expect to have generated fraction x of the k proofs in the proof package. Therefore, M will earn xkR primary reward in expectation. The expected bonus reward is straightforward to calculate as well, but requires the following observation.

THEOREM 8: The rank (i.e., hash value) of a proof is uncorrelated with the time it is generated.

PROOF: Let $T = P_1, P_2, \ldots$ be the set of all proofs generated during time interval I, and assume without loss of generality that the proofs are ordered chronologically so that P_i was generated before P_j if i < j. Define V(P) as the hash value of proof P and let V(T) denote the set of all proof values generated during I. It will suffice to show that the conditional probability that P_i achieves a given value $v \in V(T)$ is uniform for all P_i .

Being drawn from a uniform distribution, we have that $P(V(P_i) = v)$ is equal for all proofs P_i . Next define $V_v(T) = V(T) \setminus \{v\}$, which implies $P(V(T) \mid V(P_i) = v) = P(V_v(T))$ because $P(V(P_i))$ and

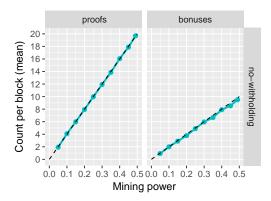


Figure 4: Everyone is honest. The dotted lines show the predicted value of R and B from Eq. 13.

 $P(V(P_i))$ are independent for $i \neq j$. Thus

$$\begin{split} P(V(P_i) &= v \mid V(T)) = \\ \frac{P(V(T) \mid V(P_i) = v)P(V(P_i) = v)}{P(V(T))} &= \\ P(V(P_i) &= v) \frac{P(V_v(T))}{P(V(T))} &= c \end{split}$$

for some constant c.

We can use Theorem 8 to show that half of a miner's proofs in the proof package are expected to be generated after the 1OS. Thus honest miner M, with fraction x of the hash rate, will generate $\frac{xk}{2}$ proofs that use the 1OS as support. It follows that M's expected bonus reward is equal to $\frac{xkB}{2}$. Finally, the expected total reward for the honest miner is given by

$$T_H = xk\left(R + \frac{B}{2}\right). \tag{13}$$

From this expression for total reward, we can see that honest mining delivers all three desired mining properties. First, a miner's reward is proportional to her hash rate, which encourages her to mine as much as possible rather than stopping once a proof is found. Second, her total reward is an increasing function of the number of her proofs that point to the 1OS. And third, because total reward is also an increasing function of the number of proofs in the proof package, she is incentivized to release her proofs as soon as possible so as to give them greatest chance of being included.

Figure 4 shows the results of this rewards scheme from a simulation of honest miners. The dotted lines show the values predicted by Eq. 13. In the next section, we demonstrate that dishonest miners earn only fewer rewards.

9 THWARTING ATTACKS

We next demonstrate quantitatively that its reduced inter-block-time variance allows Bobtail to thwart both *doublespend* [1] and *selfish mining* [5] attacks. We further show that while Bobtail introduces the possibility of a new *proof withholding* attack, a simple protocol policy ensures that attackers receive substantially lower reward when carrying out this attack.

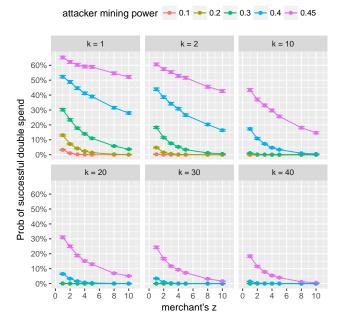


Figure 5: Doublespend attack success given k for various values of attacker mining power (each line) and merchants embargo period z (on the independent axis). Error bars show 95% c.i.'s.

9.1 Doublespend and Selfish Mining Attacks

Figure 5 shows a Monte Carlo simulation of the doublespend attack. The merchant has setup an embargo period of z blocks. The attacker's strategy is to mine until its branch is longer than that of the honest miners, or until the honest branch is ahead by 3z + 5 (to ensure that the attack has finite duration). Each facet of the plot represents a value of k. The results show that as k increases and variance decreases, the probability of attacker success significantly decreases. For example, in today's implementations of both Bitcoin and Ethereum, an attacker with 40% of the mining power will succeed with 30% probability when z = 8; however, using Bobtail with $k \geq 20$, the probability of success falls to less than 1%.

Figure 6 shows a similar result for selfish mining via a Monte Carlo simulation. The attacker follows the selfish mining strategy and it is assumed that, during a block race, the attacker's block always propagates to miners before the block of any other honest miner. The figure shows the proportion of blocks on the main chain won by attackers. The dashed line represents the proportion that would be won by honest mining. For example, a selfish miner with 40% of the mining power will claim about 66% of blocks with Bitcoin and Ethereum currently; however, using Bobtail with $k \geq 5$, the same miner will find that selfish mining is less successful than honest mining.

9.2 Zero-cost Zero-confirmation (ZCZC) Doublespend Attacks

Our approach to mitigating ZCZC doublespends is to simply with-hold reward from the offending party. Suppose that proof set \mathcal{P}_i is associated with transaction set \mathcal{T}_i , which contains T'. Suppose

8

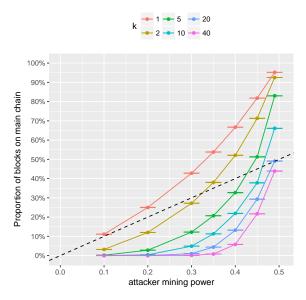


Figure 6: Selfish mining attack success given k (different lines) for various values of attacker mining power (on the independent axis). The dashed line shows the results of honest mining. Error bars show 95% c.i.'s.

further that \mathcal{T}_1 contains T, a transaction incompatible with T'. As stipulated in Section 3, the miner assembling the block can include bounty $\mathcal{B}(\mathcal{T}_i,T')$. We now further stipulate that both the primary and bonus rewards nominally owed to \mathcal{P}_i for this block shall instead be awarded to \mathcal{P}_1 . The implication of this policy is that it is the miner of \mathcal{P}_1 who determines what is the *correct* transaction when there exists an incompatibility. Note that, based on this approach, it becomes possible for the miner of \mathcal{P}_1 to intentionally mine transactions incompatible with the other miners in order to claim their reward. However, this strategy could only be profitable in the long-run if the miner possesses more than 50% of the hash rate. Otherwise, the miner's proofs will most often be something other than \mathcal{P}_1 , in which case they will be the ones to lose reward.

9.3 Proof Withholding Attacks

Bobtail allows for an attack where a malicious miner withholds proofs for a competitive advantage. In this section, we demonstrate that our design of Bobtail ensures the economic rewards for withholding attackers is substantially lower than that of honest miners.

In the withholding attack, the malicious miner does not announce her own proofs to the other miners. This behavior can be advantageous in two ways. First, it gives her more time to become the 1OS, which means she controls the set of transactions included in the block, \mathcal{T}_1 . Second, it allows the attacker to *pack* more of her own proofs into the proof package if she does manage to mine the 1OS. The attacker mines until either she is able to assemble a block as author of the 1OS; or it is clear that the honest miners are more likely to release a block without her withheld proofs. In the latter case, she disseminates her withheld proofs, hoping that some will be included in the proof package of the next block.

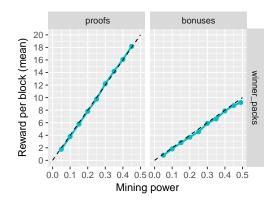


Figure 7: All miners behave honestly. The dashed lines show the predicted value of R and B from Eq. 13.

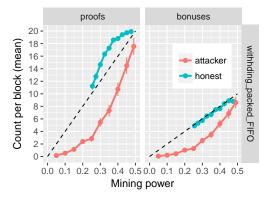


Figure 8: Withholding by attackers results in greater rewards for the honest miners. The dashed lines show the predicted value of *R* and *B* from Eq. 13.

To thwart the attack, Bobtail includes two simple rules. First, it is considered honest behavior for miners to prioritize inclusion of their own proofs when assembling a block. Second, after prioritizing their own proofs, if multiple subsets of k proofs can be used to assemble a block, an honest miner will select proofs from other miners in the order that they were received locally over the network. In other words, the proofs of the withholding attacker will be likely left out if withheld too long.

We evaluated this attack using a Monte Carlo simulation. Figure 7 shows the allocation of rewards and bonuses when all miners act honestly: they are precisely predicted by Eq. 13. Figure 8 shows result of the withholding attack given the two rules described above. Malicious behavior results in the attacking miner receiving lower reward from proofs and bonuses than honest miners. This is because the profit that she loses for releasing her proofs too late when she does not mine the block is greater than the profit she gains by withholding when she does. Furthermore, the rewards to honest miners are actually greater because of the attack.

9.4 Denial-of-Reward Attacks

In contrast to doublespend and selfish mining attacks, a denial-of-reward (DoR) attack can be carried out by any network participant, not just a miner. The attacker releases two incompatible transactions T and T', each to disjoint subsets of the miners. The result is that any proof \mathcal{P}_i generated such that $T \in \mathcal{T}_i$, will receive no reward if \mathcal{P}_1 is generated such that $T' \in \mathcal{T}_1$. In this way, the attacker can lower the profitability of mining for all or a subset of miners.

When network latency between miners is reasonably low, and assuming that an eclipse attack [31] on miners is not possible, DoR attacks can be rendered largely ineffective. Even if half the miners initially receive transaction T, while the other half receive incompatible transaction T', all miners will receive both T and T' within seconds. In Bitcoin, the probability that one of the k lowest OSes is mined within a time-period of a few seconds is very low. In Ethereum, the probability would be much higher, but miners could simply adopt a policy of waiting several seconds before beginning to mine a newly received transaction.

With knowledge of the existence of T and T', the safest strategy for miners is to exclude both. However, this practice can leave an honest transaction creator stuck (who might have accidentally submitted two transactions spending the same UTXO) and also prevents miners from collecting the associated fee. Thus, the best approach is for miners to establish the following convention. If T' is received more than a few seconds after T, then discard T' and mine T exclusively. When T and T' are received within a few seconds of each other, mine the transaction with the lowest hash value if the fees are the same, otherwise mine the transaction with the highest fee. Note that miners have ample incentive to follow this convention because if they do not, then there is a good chance that the proofs they mine will be incompatible with the 1OS, and therefore will receive no reward.

10 CONCLUSION

We have designed and characterized a novel method of low-variance blockchain mining. We have derived expressions for the expectation and variance of the Bobtail mining proof and the number of hashes performed for any value of k. Using these expressions, we have shown that Bobtail reduces variance by a factor of O(1/k), compared to using k=1. We have also shown that forks are created by Bobtail miners no more often than existing systems, and that dishonest miners receive significantly lower rewards due to minor protocol adjustments. Furthermore, we have demonstrated that low-variance mining significantly reduces the effectiveness of doublespend and selfish mining attacks, and that our design thwarts withholding and denial-of-reward attacks. Finally we have introduced a policy for proof dissemination that keeps network traffic to a minimum.

REFERENCES

- [1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," May 2009.
- [2] "Ethereum Homestead Documentation," http://ethdocs.org/en/latest/.
- [3] "Litecoin," https://litecoin.org/.
- [4] A. P. Ozisik and B. N. Levine, "An Explanation of Nakamoto's Analysis of Double-spend Attacks," University of Massachusetts, Amherst, MA, Tech. Rep. arXiv:1701.03977, January 2017.
- [5] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in International conference on financial cryptography and data security. Springer,

- 2014, pp. 436-454.
- [6] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in In 12th Annual International Cryptology Conference, 1992, pp. 139–147.
 [7] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client
- [7] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in Revised Papers from the 8th International Workshop on Security Protocols, 2001, pp. 170–177. [Online]. Available: http://dl.acm.org/citation.cfm? id=647218,720854
- [8] B. Groza and B. Warinschi, "Cryptographic puzzles and dos resilience, revisited," Des. Codes Cryptography, vol. 73, no. 1, pp. 177–207, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1007/s10623-013-9816-5
- [9] D. Dean and A. Stubblefield, "Using client puzzles to protect tls," in *Proceedings of the 10th Conference on USENIX Security Symposium Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251327.1251328
- [10] X. Wang and M. K. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, ser. SP '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 78-. [Online]. Available: http://dl.acm.org/citation.cfm?id=829515.830561
- [11] M. Franklin and D. Malkhi., "Auditable metering with ligthweigth security." in Proc. Financial Cryptography, 1997, pp. 151–160.
- [12] L. Chen and W. Mao, "An auditable metering scheme for web advertisement applications," *Information Security*, pp. 475–485, 2001.
- [13] M. Jakobsson and A. Juels, "Proofs of Work and Bread Pudding Protocols," in Proc. Conference on Secure Information Networks: Communications and Multimedia Security, 1999, pp. 258–272. [Online]. Available: http://dl.acm.org/citation.cfm?id=647800.757199
- [14] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks." in *Proc. Networks and Distributed Security Systems*, 1999, pp. 151–165.
- [15] B. Laurie and R. Clayton, ""Proof-of-work" proves not to work; version 0.2," in Proc. Workshop on Economics and Information Security, 2004.
- [16] J. Douceur, "The Sybil Attack," in Proc. Intl Wkshp on Peer-to-Peer Systems (IPTPS), Mar. 2002.
- [17] A. Back, "Hashcash Amortizable Publicly Auditable Cost-Functions," 2002. [Online]. Available: http://www.hashcash.org/papers/amortizable.pdf
- [18] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," ACM Trans. Internet Technol., vol. 5, no. 2, pp. 299–327, May 2005. [Online]. Available: http://doi.acm.org/10.1145/1064340.1064341
- [19] "Ethash," https://github.com/ethereum/wiki/wiki/Ethash, Aug 3 2017.
- [20] F. Coelho, "An (Almost) Constant-Effort Solution- Verification Proof-of-Work Protocol Based on Merkle Trees," in *Progress in Cryptology – AFRICACRYPT*, June 2008, pp. 80–93.
- [21] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol," in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). Santa Clara, CA: USENIX Association, 2016, pp. 45–59. [Online]. Available: https://www.usenix.org/conference/nsdi16/technicalsessions/presentation/eval
- [22] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in Proc. IEEE Security and Privacy, 2014, pp. 475–490.
- [23] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," in *International Conference on Financial Cryptography and Data Secu*rity. Springer, 2016, pp. 142–157.
- [24] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake [Extended Abstract] y," ACM SIGMETRICS Performance Evaluation Review, vol. 42, no. 3, pp. 34–37, 2014.
- [25] X. Boyen, C. Carr, and T. Haines, "Blockchain-Free Cryptocurrencies: A Framework for Truly Decentralised Fast Transactions," Cryptology ePrint Archive, Report 2016/871, Sept 2016, http://eprint.iacr.org/2016/871.
- [26] A. P. Ozisik, G. Andresen, G. Bissias, A. Houmansadr, and B. N. Levine, "Graphene: A New Protocol for Block Propagation Using Set Reconciliation," in Proc. of International Workshop on Cryptocurrencies and Blockchain Technology (ESORICS Workshop), 2017.
- [27] P. Rizun, "Subchains: A Technique to Scale Bitcoin and Improve the User Experience," Ledger, vol. 1, pp. 38–52, 2016.
- [28] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal Selfish Mining Strategies in Bitcoin," https://arxiv.org/pdf/1507.06183.pdf, July 2015.
- [29] A. Gervais, G. O. Karame, K. Wust, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the Security and Performance of Proof of Work Blockchains," https://eprint.iacr.org/2016/555, 2016.
- [30] G. Casella and R. L. Berger, Statistical inference. Pacific Grove, CA: Brooks Cole, 2002. [Online]. Available: http://opac.inria.fr/record=b1134456
- [31] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in 24th USENIX Security Symposium, 2015, pp. 129–144.

A DISTRIBUTIONS OF MINING PROCESSES

Consider the distribution of H an arbitrary random variable chosen from the sequence of block hashes H_1, \ldots, H_h . We have $f_H(t;S) = 1/S$ and $F_H(t;S) = t/S$. The following result is well known⁴.

LEMMA 1: The probability density function (pdf) of the ith order statistic, V_i , from h samples (i.e., hashes) is

$$f_{V_{i}}(t;S,h) = \frac{h!}{(i-1)!(h-i)!} f_{H}(t) (F_{H}(t))^{i-1} (1 - F_{H}(t))^{h-i}$$

$$= \frac{h!}{(i-1)!(h-i)!} \frac{1}{S} \left(\frac{t}{S}\right)^{i-1} \left(1 - \frac{t}{S}\right)^{h-i}.$$
(14)

When hash interval I corresponds to the desired block time, say 10 minutes for Bitcoin, there will be many hashes performed during the interval. So it is reasonable to consider how the distribution for V_i changes in the limit that h approaches infinity.

THEOREM 9: In the limit that h approaches infinity, $V_i \sim$ Gamma(i, v) where v is the expected value of the minimum hash.

PROOF: Define g(t; i, v) to be the PDF of the Gamma distribution with shape parameter i and scale parameter v. If the number of hashes approaches infinity, then so must the size of the hash space, and yet S must always be larger than h. Therefore, we assume that h = S/v for arbitrary parameter v > 1. Under this assumption we can equivalently consider limit that S approaches infinity. We have

$$f_{V_{i}}(t;S,h) = \lim_{h \to \infty} f_{V_{i}}(t;S,h)$$

$$= \lim_{S \to \infty} \frac{(S/v)!}{(i-1)! \left(\frac{S}{v} - i\right)!} \frac{1}{S} \left(\frac{t}{S}\right)^{i-1} \left(1 - \frac{t}{S}\right)^{\frac{S}{v} - i}$$

$$= \lim_{S \to \infty} \frac{(S/v)!}{S^{i}(i-1)! \left(\frac{S}{v} - i\right)!} t^{i-1} \left(1 - \frac{t}{S}\right)^{\frac{S}{v} - i}$$

$$= \frac{t^{i-1}}{(i-1)!} \left[\lim_{S \to \infty} \frac{\left(\frac{S}{v}\right) \dots \left(\frac{S}{v} - i + 1\right)}{S^{i}}\right] \left[\lim_{S \to \infty} \left(1 - \frac{t}{S}\right)^{\frac{S}{v} - i}\right]$$

$$= \frac{t^{i-1}}{(i-1)! v^{i}} e^{\frac{-t}{v}}$$

$$= g(t; i, v), \tag{15}$$

The second to last step follows from the fact that

$$\lim_{S \to \infty} \frac{\left(\frac{S}{v}\right) \dots \left(\frac{S}{v} - i + 1\right)}{S^i} = \lim_{S \to \infty} \frac{\left(\frac{S}{v}\right)^i}{S^i} = \frac{1}{v^i}, \tag{16}$$

and the common limit

$$\lim_{S \to \infty} \left(1 - \frac{t}{S} \right)^{\frac{S}{v}} = e^{\frac{-v}{v}},\tag{17}$$

which implies that

$$\lim_{S \to \infty} \left(1 - \frac{t}{S} \right)^{\frac{S}{v} - i} = \left[\lim_{S \to \infty} \left(1 - \frac{t}{S} \right)^{-i} \right] \left[\lim_{S \to \infty} \left(1 - \frac{t}{S} \right)^{\frac{S}{v}} \right]$$
(18)
$$= 1 \cdot e^{\frac{-t}{v}}.$$
(19)

When i = 1, $V_1 \sim \mathsf{Gamma}(t; 1, v) = \mathsf{Exponential}(t; v)$. And since the expected value of an exponential random variable is equal to the value of its scale parameter, we can see that v is simply the expected value of the minimum hash.

Next, consider the PDF of X_i , $f_{X_i}(x; S, v)$. After x hash intervals, let E, L, and G be, respectively, the events that the ith order statistic is equal to v, the order statistics below i are less than v, and the order statistics above i are greater than v. Furthermore, let O be the set of all divisions of H_1, \ldots, H_h into distinct sets $\{H \mid H = V_i\}$, $\{H \mid H < V_i\}$, and $\{H \mid H > V_i\}$. We have

$$f_{X_{i}}(x; S, v) = \sum_{o \in O} P[E(x), L(x), G(x) \mid o]$$

$$f_{X_{i}}(x; S, v) = \binom{hx}{i-1}(hx - i + 1)P[E(x) \mid o]P[L(x) \mid o]P[G(x) \mid o]$$

$$f_{X_{i}}(x; S, v) = \frac{(hx)!}{(i-1)!(hx-i)!}P[E(x) \mid o]P[L(x) \mid o]P[G(x) \mid o]$$

$$f_{X_{i}}(x; S, v) = \frac{(hx)!}{(i-1)!(hx-i)!} \frac{1}{S} \left(\frac{v}{S}\right)^{i-1} \left(1 - \frac{v}{S}\right)^{hx-i}$$
(20)

Assuming that I is large, it again makes sense to consider the limit as h approaches infinity.

THEOREM 10: In the limit that h approaches infinity, $X_i \sim \text{Gamma}(i, 1/r)$ where r is the expected number of hashes falling below v during a given interval.

PROOF: The probability that any given hash *succeeds*, i.e. falls below v, is given by $p = \frac{v}{S}$. Again, we would like to consider the limit as h approaches infinity. But in doing so, we must ensure that the probability of hash success remains constant. In other words, the probability of hash success must diminish as h increases. So there must exist some constant r such that $\frac{r}{h} = p = \frac{v}{S}$. It follows that

$$f_{X_i}(x; S, v) = \frac{(hx)!}{(i-1)!(hx-i)!} \frac{r}{h} \left(\frac{r}{h}\right)^{i-1} \left(1 - \frac{r}{h}\right)^{hx-i}$$
 (21)

Arguing in similar fashion as for V_i , we find that

$$\lim_{h \to \infty} f_{X_i}(x; S, v) = g(x; i, 1/r).$$

Thus, $E[X_i] = 1/r$, which implies that r should be interpreted as the expected rate at which hashes fall below v during a single interval I.

We can see that the distributions for V_i and X_i are related through the change of variables v = 1/r, and all four parameters v, r, h, and S are related by

$$v=r\frac{S}{h}.$$

⁴See for example, Casella and Berger [30]

In words, the latter states: the expected value of the minimum hash is related to the expected number of blocks mined per interval by the ratio $\frac{S}{h}$.

A.1 Joint Distributions

THEOREM 11: The joint distribution of the ith and jth order statistic of uniform random samples H_1, \ldots, H_h is given by

$$f_{V_i, V_j}(t_i, t_j; v) = g(t_i; i, v)g(t_j - t_i; j - i, v).$$
 (22)

where v is the expected value of the minimum hash.

PROOF: It is well known⁵ that the joint distribution of the ith and jth order statistics, out of h total samples, is given by

$$f_{V_{i},V_{j}}(t_{i},t_{j};v) = \frac{h!}{(i-1)!(j-1-i)!(h-j)!} f_{H}(t_{i}) f_{H}(t_{j}) [F_{H}(t_{i})]^{i-1} \times [F_{H}(t_{j}) - F_{H}(t_{i})]^{j-1-i} [1 - F_{H}(t_{j})]^{n-j}.$$
(23)

Thus, we have

$$f_{V_{i},V_{j}}(t_{i},t_{j};S,v) = \frac{t_{i}^{i-1}(t_{j}-t_{i})^{j-1-i}}{S^{j}} \frac{h!}{(i-1)!(j-1-i)!(h-j)!} \left(1 - \frac{t_{j}}{S}\right)^{h-j}$$

$$= \frac{\frac{S}{v} \cdots \left(\frac{S}{v}-j+1\right)}{S^{j}} \frac{t_{i}^{i-1}(t_{j}-t_{i})^{j-1-i}}{(i-1)!(j-1-i)!} \left(1 - \frac{t_{j}}{S}\right)^{\frac{S}{v}-j}.$$
(24)

Finally, assuming j > i, and reasoning in the limit as $S \to \infty$ in the same manner as above.

$$f_{V_{i},V_{j}}(t_{i},t_{j};v) = \lim_{S \to \infty} f_{V_{i},V_{j}}(t_{i},t_{j};S,v)$$

$$= \frac{1}{v^{j}} \frac{t_{i}^{i-1}(t_{j}-t_{i})^{j-1-i}}{(i-1)!(j-1-i)!} e^{-\frac{t_{j}}{v}}$$

$$= \frac{t_{i}^{i-1}}{v^{i}(i-1)!} e^{-\frac{t_{i}}{v}} \frac{(t_{j}-t_{i})^{j-1-i}}{v^{j-1}(j-1-i)!} e^{-\frac{t_{j}-t_{i}}{v}}$$

$$= g(t_{i};i,v)g(t_{j}-t_{i};j-i,v).$$
(25)

Because X_i shares the same distribution as V_i , up to the change of variables v = 1/r, the following result follows trivially.

THEOREM 12: The joint distribution of X_i and X_j , j > i, is given by

$$f_{X_i,X_j}(t_i,t_j;1/r)=g(t_i;i,1/r)g(t_j-t_i;j-i,1/r).$$
 (26) where r is the expected rate at which hashes fall below v during a single interval I .

B MOMENTS OF W_K

The goal of this section is to empirically validate our expression for $E[W_k]$ from Section 5 and then derive and validate an expression for $Var[W_k]$. W_k is simply the sample mean over the lowest k order statistics V_1, \ldots, V_k . But, unfortunately, the analysis below is not straightforward because the V_i are neither independent nor identically distributed.

Empirical Validation of Theorem 2. Figure 9 compares Eq. 4 versus a result obtained through a small Monte Carlo simulation

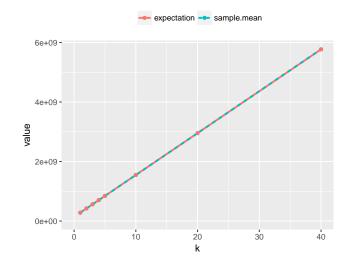


Figure 9: Eq. 4 versus simulation.

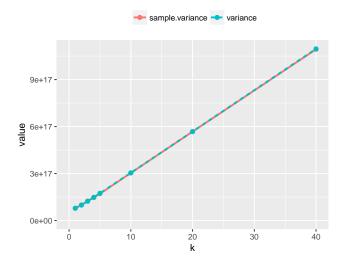


Figure 10: Eq. 27 versus simulation.

of Bobtail mining run tens of thousands of times, with k as the independent variable. In all cases, the results match perfectly.

B.1 Variance of W_k

THEOREM 13: The variance of W_k is $Var[W_k] = \frac{(k+1)(2k+1)}{6k}v^2. \tag{27}$

PROOF: Assuming that j > i, Theorem 11 yields

 $^{^5 \}mbox{See}$ Casella and Berger [30], Theorem 5.4.6.

$$E[V_{i}V_{j}] = \int_{0}^{\infty} \int_{t_{i}}^{\infty} t_{i}t_{j}g(t_{i}; i, v)g(t_{j} - t_{i}; j - i, v)dt_{j}dt_{i}$$

$$= \int_{0}^{\infty} t_{i}g(t_{i}; i, v) \int_{0}^{\infty} (w + t_{i})g(w; j - i, v)dwdt_{i}$$

$$= \int_{0}^{\infty} t_{i}g(t_{i}; i, v)[(j - i)v + t_{i}]dt_{i}$$

$$= v(j - i) \int_{0}^{\infty} t_{i}g(t_{i}; i, v)dt_{i} + \int_{0}^{\infty} t_{i}^{2}g(t_{i}; i, v)dt_{i}$$

$$= iv^{2}(j - i) + iv^{2}(1 + i)$$

$$= iv^{2}(1 + j).$$
(28)

Before continuing, we note that since $V_i \sim \mathsf{Gamma}(i, v)$, it follows that $Var[V_i] = iv^2$. Now, assuming that j > i, and using Eq. 28, we have

$$cov[V_i, V_j] = E[V_iV_j] - E[V_i]E[V_j]$$

$$= iv^2(1+j) - (iv)(jv)$$

$$= iv^2$$

$$= Var[V_i].$$
(29)

Finally, we find the variance of W_k by substituting first Eq. 1 and then Eq. 29:

$$Var[W_k] = Var \left[\frac{1}{k} \sum_{i=1}^{k} V_i \right]$$

$$= \frac{1}{k^2} Var \left[\sum_{i=1}^{k} V_i \right]$$

$$= \frac{1}{k^2} \left(\sum_{i=1}^{k} Var[V_i] + 2 \sum_{j=1}^{k} \sum_{i=1}^{j-1} \text{cov}[V_i, V_j] \right)$$

$$= \frac{1}{k^2} \left(\sum_{i=1}^{k} iv^2 + 2 \sum_{j=1}^{k} \sum_{i=1}^{j-1} iv^2 \right)$$

$$= \frac{v^2}{k^2} \left(\frac{k(k+1)}{2} + \sum_{j=1}^{k} j(j-1) \right)$$

$$= \frac{v^2}{k^2} \left(\frac{k(k+1)}{2} + \frac{k(k+1)(2k+1)}{6} - \frac{k(k+1)}{2} \right)$$

$$= \frac{(k+1)(2k+1)}{6k} v^2.$$

Empirical Validation of Theorem 13. Figure 10 shows Eq. 27 versus our Monte Carlo simulation where k is the independent variable. The results show an exact match.

Because X_i shares the same distribution as V_i , up to the change of variables v = 1/r, the following result follows trivially.

THEOREM 14:

$$Var\left[\frac{1}{k}\sum_{i=1}^{k}X_{i}\right] = \frac{(k+1)(2k+1)}{6k}\left(\frac{1}{r}\right)^{2}.$$
 (31)

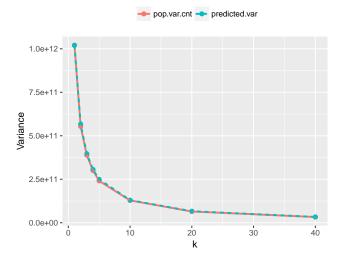


Figure 11: Eq. 10 from Theorem 5 (in blue) versus simulation (in red).