RPPLNS: Pay-per-last-N-shares with a Randomised Twist*

Jonathan Katz[3], Philip Lazos[1], Francisco J. Marmolejo-Cossío[2], and Xinyu Zhou[3]

[1] Sapienza University of Rome: plazos@gmail.com
[2] University of Oxford, IOHK: francisco.marmolejo@cs.ox.ac.uk
[3] University of Maryland: jkatz2@gmail.com, xyzhou@terpmail.umd.edu

Abstract. "Pay-per-last-N-shares" (PPLNS) is one of the most common payout strategies used by mining pools in Proof-of-Work (PoW) cryptocurrencies. As with any payment scheme, it is imperative to study issues of incentive compatibility of miners within the pool. For PPLNS this question has only been partially answered; we know that reasonably-sized miners within a PPLNS pool prefer following the pool protocol over employing *specific* deviations. In this paper, we present a novel modification to PPLNS where we randomise the protocol in a natural way. We call our protocol "Randomised pay-per-last-N-shares" (RPPLNS), and note that the randomised structure of the protocol greatly simplifies the study of its incentive compatibility. We show that RPPLNS maintains the strengths of PPLNS (i.e., fairness, variance reduction, and resistance to pool hopping), while also being robust against a richer class of strategic mining than what has been shown for PPLNS.

1 Introduction

In Bitcoin, miners maintain a ledger of transactions and are rewarded for their efforts by the underlying protocol. Successfully appending a block to the ledger is computationally difficult; it can take common computing equipment years to find a single block in expectation. In response to this variability, miners often pool their resources so that rather than rarely earning large rewards they earn smaller rewards at a more consistent rate.

In more detail, we can think of the Bitcoin ecosystem as consisting of n strategic miners, m_1,...,m_n, with hash powers $\alpha_1, ..., \alpha_n > 0$, such that $\sum_i \alpha_i = 1$. Intuitively, α_i represents the proportional computational power that a miner has. Assuming that block mining happens in discrete time-steps, the probability that m_i mines the next block is equal to α_i. The dilemma of the previous section corresponds to a single miner having a small α_i value, and hence only mining a block in $1/\alpha_i$ time steps in expectation. On the other hand, a set of S miners could combine their computational power and have a $\sum_{m_i \in S} \alpha_i$ chance of mining the next block, sharing the rewards if they manage to do so. In order to share rewards, the pool must have a way of identifying the computational contribution of each miner. This is done by accepting partial proofs of work, which are "near-misses" to Bitcoin's desired hash rate and whose number is directly proportional to the computational effort spent on extending the blockchain. The pool operator collects these "near-misses", called shares, reported by every miner in the pool, and uses them to distribute payments once an actual block is found. Ideally, participating in a pool should have (at least) the following guarantees:

1. Fairness: miners earn the same block reward in expectation as mining alone.
2. Variance reduction: miners have lower variance in block reward than when mining alone.
3. Robustness against pool hopping: at no point of time is there is a benefit in leaving the pool for another.
4. Incentive Compatibility: to maximize their reward, each participating miner should always expend maximum effort and report shares/blocks immediately as they are generated.

One of the most popular pool mining protocols which (partially) satisfies these properties is "Pay-per-last-N-shares" (PPLNS). Miners report shares to a pool operator which maintains a queue of the N most recent shares reported to it, and if a block is found and reported by the pool, the owners of these N shares are paid proportionally ($1/N$ times the value of a block for each such share). The structure of PPLNS is such that it satisfies properties 1-3 above [11]. With respect to property 4, [14, 13] demonstrate that miners in a PPLNS are incentivised to act honestly if they are only permitted specific deviations, hence PPLNS only partially satisfies property 4.

* A complete version of this extended abstract can be found [2].

Our Contributions. In this paper, we present a novel modification to PPLNS where we randomise the protocol in a natural way. We call our protocol "Randomised pay-per-last-N-shares" (RPPLNS), and note that the randomised structure of the protocol greatly simplifies the study of its incentive compatibility. We show that RPPLNS maintains the strengths of PPLNS (i.e., fairness, variance reduction, and resistance to pool hopping), while also being robust against a richer class of strategic mining than what has been shown for PPLNS. In particular, [13] showed that PPLNS is robust when strategies are limited to either mining honestly *or* secretly hoarding all accumulated shares and suddenly releasing them if a block is found. For RPPLNS we provide experimental evidence that suggests that mining honestly maximizes rewards among any strategy involving hoarding and releasing shares.

Related Work Strategic mining has been studied from the inception of Bitcoin [8]. In [4], the authors demonstrate that honest mining is not robust to strategic mining in terms of block reward, even when a miner has less than a majority computational stake in the Bitcoin ecosystem. This work is refined in [12], [9] and [5] by generalising selfish mining, pairing selfish mining with network-level attacks and proving limited incentive compatibility of honest mining if miners have low enough hash power. Further incentives at the individual miner level have also been studied in [1] where transaction rewards alone are demonstrated to be unstable for incentive compatibility of mining. At the pool level, the author of [3] demonstrates an infiltration attack pools can wage against each other that leads to an iterated prisoner's dilemma between pool operators (dubbed the miner's dilemma). This has been further refined in [6].

An extensive survey of pool protocols can be found in [11]. In [13], the authors study incentive compatibility in pool protocols that decide how to make payments on the basis of the quantity of shares each miner reported, irrespective of the order in which they are received. Subsequently, the authors of [14] study a different class of strategic deviations in PPLNS where a miner hoards a certain number $x \in \mathbb{N}$ of shares. Subsequent shares are published immediately, and whenever a block is found, those x shares are published immediately before publishing the block. Their analysis makes the assumption that each strategic miner reaches their threshold x, and show when being honest outperforms being strategic in this setting. Finally, the authors of [10] exhibit specific reporting strategies that can be beneficial to strategic miners at high enough hash rates.

2 PPLNS and RPPLNS

Pool Basics In general, the pool operator prepares the 'template' of the next block in advance to send to its pool miners: this contains all the information about where the next block should be added and what it should contain. Then, pool miners try to fill in different nonce values at an attempt to obtain a valid block (i.e., one with a low enough hash value) whose rewards will be claimed by the pool and redistributed to its miners. When mining solo (rather than as a part of a pool), miners obtain an expected reward proportional to their hash power. In order to fairly redistribute rewards, a pool needs a proxy to measure the computational resources a miner is contributing to the pool's operations. This is done by allowing pool miners to send the pool operator near-misses (also called *shares* in this context) to the Bitcoin difficulty threshold. In our full exposition of mining pool rules, we assume the following: that there is a mining pool composed of k miners, which will call pool miners, and that all miners outside the pool (non-pool miners) are honest, and hence can be modelled as a single honest miner. The reason we model the other miners as a single honest entity is due to the fact that we focus on strategic mining deviations *within* the pool protocol only. Pool miners are denoted by $m_1, ..., m_k$, and the non-pool miner by m_0. As before, the i-th miner has hash power α_i.

Valid Share / Block Generation. The first relevant parameter that the pool must set is the *relative difficulty* of blocks to shares, which can be parametrised by a non-negative number $D \geq 1$. We model share and block generation in discrete time-steps. Each turn a share is created, and it is attributed to miner m_i with probability α_i. Furthermore, each share has a $1/D$ probability of being a block. A block can be thought of as a share that gives the pool a reward to be distributed amongst its miners.

Messages to the Pool. At any given time-step, the pool receives messages from miners which can be of the form share_i or block_i if m_i reports a share or block respectively to the pool[4]. We denote the set of messages the pool operator can receive by M.

Honest and Strategic Pool Mining. For all pool mining schemes, we say that a pool miner is honest if they mine upon the block given by the pool operator and if they report shares/blocks immediately. For independent miners, we say they are honest if they publish blocks they find immediately (The concept of a share is irrelevant outside the pool). Consequently, strategic miners have the possibility of arbitrarily hoarding shares and blocks (valid messages of the form share_i or block_i) to be published at a time after they are found. It is crucial to note that the mechanics of pool mining are such that these deviations do not come without a potential cost. Pool miners are all given a block template to work on, and shares are near-misses to the hash of a block within this template. If a miner is hoarding shares/blocks and another honest pool miner publishes a block to the pool, these shares/blocks are no longer valid and the pool will not pay the owner of them. The same scenario happens if an honest miner outside the pool mines a block. Strategic miners must therefore balance this risk while hoarding. Furthermore, if hoarding is beneficial for some miners, the pool's overall quality degrades in two ways: for them to increase their payoff some other miners will have to earn less and there is potential for decreased computational efficiency as some work might be performed twice.

Payments. Pool operators maintain a history of messages received and as a function of this history decide how to redistribute block rewards to pool miners. Notice that since the pool sets the block template for pool miners, only the pool operator can receive and redistribute funds from a valid block found by the pool. Setting a pool payment scheme determines miner incentives, hence it is crucial to analyse whether mining honestly is beneficial to pool miners. In general, pools are not obliged to redistribute the entirety of their block reward, but in this paper we focus on pools which do. We call these pools *budget balanced*.

PPLNS. At a high level, PPLNS pools are simple to understand. A pool operator sets a relative block to share difficulty, D, and a queue size parameter, $N \in \mathbb{N}$. Pool miners report shares and blocks to the pool operator, which in turn maintains a queue of length N consisting of the the most recent shares reported. If ever a block is found, the pool operator pays the shares in the queue proportionally.

Since we are studying strategic mining deviations, we have to precisely define the pool protocol, as this will govern whether miners are incentivised to be honest in the first place. We do this by modelling PPLNS as a deterministic state machine[5]. At any given moment, the pool operator maintains a state $s \in S = (\{*\} \cup [n])^N$. S represents the state space of the PPLNS queue (all possible queue compositions), where "$*$" represents an empty value while the queue is filled in the first N messages the pool operator receives. Furthermore, we let $s_0 = (*)_{i=1}^{N}$ be a distinguished initial state of the pool protocol that corresponds to an empty queue.

As with any state machine, we need to define a transition function between states. We recall that M denotes the set of messages that a pool operator can receive, hence we can specify a transition function $T : S \times M \to S$. For a given message from the i-th pool miner (meaning $i \neq 0$), $x = \mathsf{share}_i$ or $x = \mathsf{block}_i$, $T(s, x) = i : s_{<n}$, which is the concatenation of i with the first $n-1$ elements of s (The queue is filled from the left to the right). As for non-pool miner messages, $T(s, \mathsf{block}_0) = s$.

Finally, we need to specify a payment protocol in the scenario where a pool miner finds a block. This means that the pool receives a message block_i for $i \neq 0$, which in turn implies that if the pool is in state s, it transitions to state $s' = T(s, \mathsf{block}_i)$. State s' contains the identities of miners with the most recent N shares, hence we let $P : S \to \mathbb{R}^k$ be a payment function such that $P(s')_i = \frac{|\{j \mid s'_j = i\}|}{|\{j \mid s'_j \neq *\}|} \geq 0$ is the payment m_i receives from the found block.

RPPLNS. The protocol is similar to PPLNS with the added difference that shares are no longer maintained in a queue but rather a bag that does away with the sequential nature of when which share arrived. The relevant parameters for a pool are still the relative difficulty of blocks to shares, D, and, $N \in \mathbb{N}$, the size

[4] m_1 only sends block messages to the pool due to the fact that the concept of a share only makes sense for pool miners. On the other hand, a block message from m_1 corresponds to the pool becoming aware of said block on the global blockchain itself.

[5] Many common pool protocols can be cast in this framework. This is explored further in [7]

of the bag of shares maintained by the pool operator. Pool miners report shares and blocks to the pool operator. If the bag of shares maintained by the operator is not full (i.e. there are less than N shares in the bag), then the reported share is automatically added to the bag. On the other hand, when the bag is full, the reported share displaces a random share in the bag maintained by the operator. If a block is found, first it is added to the bag as a share via the aforementioned method, and subsequently the pool operator pays the shares in the bag with a proportional value of the block reward.

Once more, we must be rigorous to study incentive compatibility of pool miners. With this goal in mind, we model RPPLNS by using a randomised transition state machine. At any moment of time, the pool operator maintains a state $s \in S = \{s \in [N]^k \mid \sum s_i \leq N\}$. For a given state, s, we let s_i represent the number of shares that m_i owns in the protocol bag. We also let $s_0 = \vec{0} \in [N]^k$ be a distinguished initial state of the pool protocol which corresponds to an empty bag.

We now define the randomised transitions that an RPPLNS pool takes upon receiving messages from miners. In order to do so, we let $\Delta(S)$ denote the set of all probability distributions over S. We let $T : S \times M \to \Delta(S)$ be the randomised transition function of RPPLNS. This means that if the pool is in state $s \in S$ and receives message $x \in M$, then the resulting state s' will be distributed according to $T(s, x)$, which we define as follows:

- If $x = \mathsf{block}_0$, $\mathbb{P}_{T(s,x)}(s') = \mathbb{I}(s' = s)$, the indicator function for s. In other words, the pool does not change state.
- If $x \in \{\mathsf{share}_i, \mathsf{block}_i\}$ such that $i \neq 0$, and in addition $\sum s_{i=1}^k < N$, then $\mathbb{P}_{T(s,x)}(s') = \mathbb{I}(s' = s + e_i)$. In other words, the pool's bag is not full in s, hence it deterministically adds m_i's share to the bag.
- If $x \in \{\mathsf{share}_i, \mathsf{block}_i\}$ such that $i \neq 0$, and in addition $\sum s_{i=1}^k = N$, then $\mathbb{P}_{T(s,x)}(s - e_j + e_i) = \frac{s_j}{N}$. In other words, the pool operator picks a random share from s to kick out to make way for m_i's newly reported share.

Finally, we define the payment that occurs if a block is found. Suppose that upon receiving a block, the pool state transitions (randomly) to state s'. As with PPLNS, we let $P : S \to \mathbb{R}^k$ be a payment function such that $P(s')_i = \frac{s_i}{\sum_{j=1}^k s_j} \geq 0$ is the payment m_i receives from the found block.

3 Properties of RPPLNS

In the analysis of RPPLNS, it suffices to consider a single strategic pool miner, m_1 with hash power α, a single honest pool miner m_2 with hash power β, and a single honest non-pool miner m_0 with hash power γ. Indeed, m_2 and m_0 could be composed of multiple honest miners, but if they are honest, we can model their behaviour as that of a single miner of their aggregate hash power. In addition, to model revenues, we consider a turn-based process. Every turn, either m_1, m_2 or m_0 find a share with probability α, β and γ respectively, and each share has a further $\frac{1}{D}$ probability of being a full block. We wish to point out that we say that m_0 finds shares in the sense that it computes a block with a hash that is a near-miss to the target hash (by a factor of D). m_0 does not actually report this near miss to the pool since it is not a part of the pool. However, m_0 does publish blocks immediately to all agents of the Bitcoin ecosystem, so we consider this as a message to the pool operator. Finally, since m_2 is an honest pool miner, whenever they find a share/block they communicate this immediately to the RPPLNS pool. The exact statements and formal proofs of all that follows can be found in [2].

Fairness and Variance Reduction We show that if m_1 is honest, then their expected block reward per turn is precisely α/D. Since each share has a $\frac{1}{D}$ probability of being a block, this coincides with the expected α block reward m_1 would get (per block mined by the system) mining solo. In addition, we demonstrate that RPPLNS enjoys similar variance reduction in block reward to what characterises PPLNS.

Theorem 1. *Suppose that m_1 is honest with hash power α, then their expected per-turn block reward is $\frac{\alpha}{D}$ in an RPPLNS mining pool. In addition, the variance of their per-turn block reward is $\frac{1}{D^2}(\alpha - \alpha^2) + \frac{\alpha}{ND}$*

In deterministic PPLNS, block reward variance can be computed in an identical fashion, and it is $\frac{\alpha}{2D^2} + \frac{\alpha}{ND} - \frac{\alpha^2}{D^2} - \frac{\alpha}{2ND^2}$. Typically, pools have $N = 2D$, in which case the PPLNS variance becomes $\frac{1}{D^2}(\alpha - \alpha^2) - \frac{\alpha}{4D^3}$. For this difficulty setting, RPPLNS block reward variance becomes $\frac{1}{D^2}(\alpha - \alpha^2) + \frac{\alpha}{2D^2}$. Though this is more than with standard PPLNS, this still vanishes at the same asymptotic rate of $O(1/N^2)$ when $N = \Theta(D)$.

Robustness to Pool-hopping We can show that if a miner is given the choice between mining with two different RPPLNS pools,then in expectation he will always earn the same block reward, irrespective of the initial state of his shares in each pool and how he may choose to partition his mining between said pools.

Theorem 2 (Informal). *RPPLNS is resistant to pool hopping.*

When is Honest Mining a Dominant Strategy. We recall that we are in the setting of a single pool miner being strategic. In other words we have m_1, m_2 and m_0 of hash powers α, β and γ respectively. We wish to find conditions such that m_1 is honest (i.e. publishes shares and blocks to $\mathcal{M}_R$ immediately). To understand the behaviour of the miners, we compute the best possible gain of m_1, assuming $N = 1000, D = 500$ a finite horizon of $k = 150$ steps. Then, we compare against his expected gain in the same number of steps if m_1 follows the protocol honestly. The following graphs shows the best action for m_1 given an *initial* fraction of shares in the bag, which we call F.

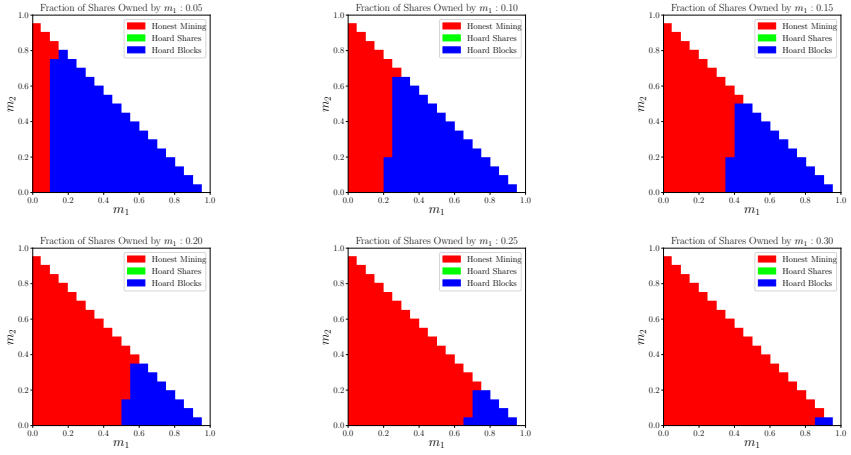

Fig. 1: $F \leq 0.35$

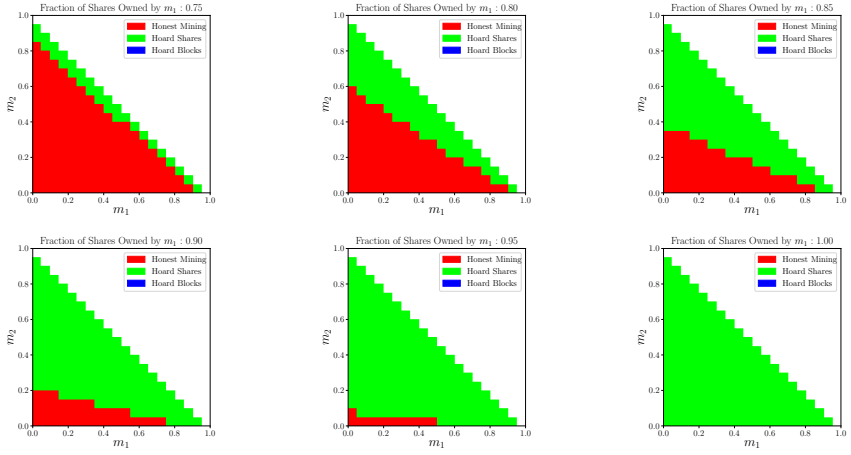

Fig. 2: $0.35 \leq F \leq 0.70$

For $F \leq 0.35$, the strategic miner begins by controlling a few shares of the bag, and his best option is often to *hoard* a block. This is because they expect to be able to add a couple more shares to the bag before

someone else manages to mine another block and invalidate their private block. What is most interesting though, is that if we recall our derivations of the steady state of bag shares from Section **??**, strategic block hoarding only occurs at hash rates and F values such that F is in fact much less than the expected number of bag shares in the steady state. This suggests that if all miners behave honestly, a single pool miner is more likely to find himself at a state where mining honestly is a dominant strategy.

On the other hand, for initial share distributions, $0.35 \leq F \leq 0.70$, our experiments suggest that honest mining is the best option throughout. We have omitted graphs of these cases since they simply paint the simplex red entirely. Finally, when $F \geq 0.70$ we see that strategic behaviour arises once more, though this time in the form of hoarding shares rather than blocks. For example, it is not difficult to see that if $F = 1$, then in both PPLNS and RPPLNS hoarding a share dominates publishing it immediately, as doing so has no effect on the state of the pool. Once again, share hoarding becomes a better strategy at initial share distributions that are far from the expected share distribution of the steady state of bag shares (details can be found in the full version). This further suggests that states where miners act strategically are rare.

Bibliography

[1] Carlsten, M., Kalodner, H., Weinberg, S.M., Narayanan, A.: On the instability of bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 154–167. ACM (2016)

[2] Cossío, F.J.M.: Equilibrium computation in games and strategic aspects of bitcoin mining. Ph.D. thesis, University of Oxford, UK (2020), http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.804398

[3] Eyal, I.: The miner's dilemma. In: 2015 IEEE Symposium on Security and Privacy. IEEE (2015)

[4] Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: International conference on financial cryptography and data security. pp. 436–454. Springer (2014)

[5] Kiayias, A., Koutsoupias, E., Kyropoulou, M., Tselekounis, Y.: Blockchain mining games. In: Proceedings of the 2016 ACM Conference on Economics and Computation. pp. 365–382. ACM (2016)

[6] Kwon, Y., Kim, D., Son, Y., Vasserman, E., Kim, Y.: Be selfish and avoid dilemmas: Fork after withholding (faw) attacks on bitcoin. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 195–209 (2017)

[7] Marmolejo Cossío, F.J.: Equilibrium computation in games and strategic aspects of bitcoin mining. Ph.D. thesis, University of Oxford (2019)

[8] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)

[9] Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. pp. 305–320. IEEE (2016)

[10] Qin, R., Yuan, Y., Wang, F.Y.: A novel hybrid share reporting strategy for blockchain miners in ppln pools. Decision Support Systems **118**, 91–101 (2019)

[11] Rosenfeld, M.: Analysis of bitcoin pooled mining reward systems. arXiv preprint arXiv:1112.4980 (2011)

[12] Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 515–532. Springer (2016)

[13] Schrijvers, O., Bonneau, J., Boneh, D., Roughgarden, T.: Incentive compatibility of bitcoin mining pool reward functions. In: International Conference on Financial Cryptography and Data Security. pp. 477–498. Springer (2016)

[14] Zolotavkin, Y., García, J., Rudolph, C.: Incentive compatibility of pay per last n shares in bitcoin mining pools. In: International Conference on Decision and Game Theory for Security. pp. 21–39. Springer (2017)