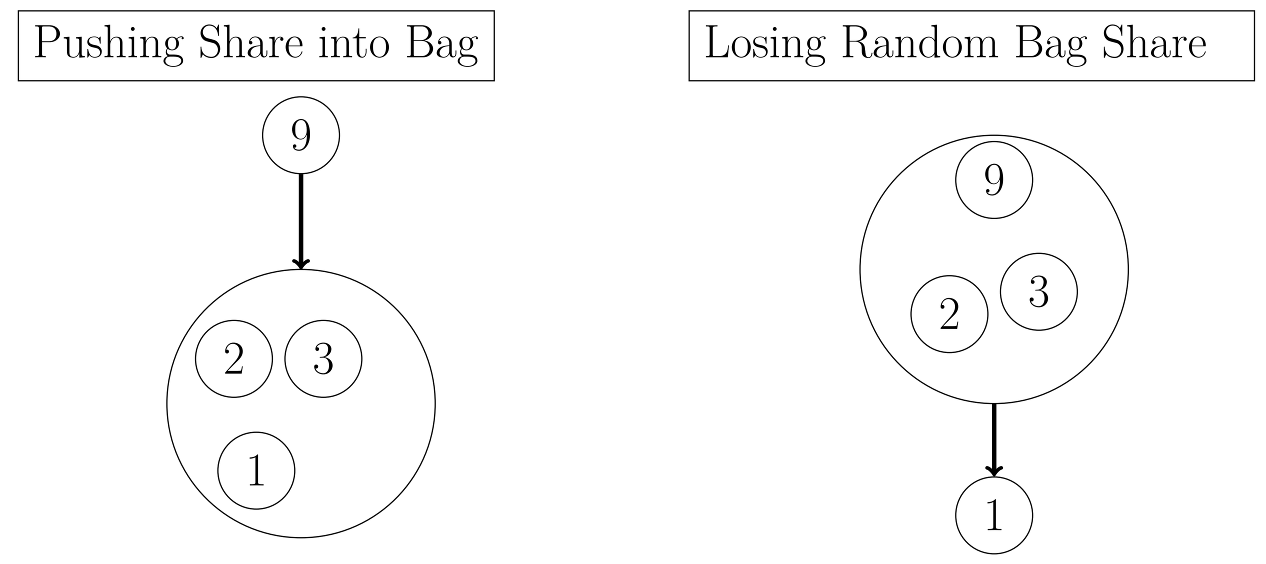
RPPLNS: Pay-per-last-N-shares with a Randomised Twist

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

[1]Sapienza University of Rome [2]University of Oxford, IOHK [3]University of Maryland

Proof-of-Work Pool Mining

Pool Mining A salient feature of the Bitcoin protocol is that honest miners are paid proportional to the computational resources they invest in mining. For smaller miners however, finding a block can take an undesirable amount of time. For this reason, miners often pool their resources together and split rewards in order to reduce payoff variance. In practice, a pool operator sets a template block for pool miners, as well as a lower hash target than that of the Bitcoin protocol. Mining a block with a hash value that is below the pool difficulty threshold is called mining a share, and the threshold is called the share difficulty. We parametrise share difficulty with $D \geq 0$ which represents multiplicatively how much lower the pool target is (equivalently, each share has a $1/D$ probability of being a block). Miners attempt different nonce values within this template, and they report shares, as well as fully valid blocks. Pool protocols must thus decide how to redistribute rewards to pool miners as a function of the shares and blocks they report.



PPLNS/RPPLNS: In PPLNS a pool operator maintains an internal queue of length N (where N is decided by the pool operator). When shares/blocks are found, they push the oldest shares out of the queue. When a block is found, said N shares in the queue are paid a proportional $1/N$ amount of the block reward as well. In RPPLNS the queue is replaced with a bag, where a random share is kicked out rather than the oldest.

Desirable Properties: 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: for any fixed amount of time, miners have lower variance in block reward than when mining alone.
3. Robustness against pool hopping: at no point of time is there a benefit in leaving the pool for another one or leaving the pool to mine individually.
4. Incentive Compatibility: to maximize their reward, each participating miner should always expend maximum effort and report shares/blocks immediately as they are generated.

Our Contributions: Although PPLNS satisfies conditions 1-3 above, the incentive compatibility of honest pool mining has only been shown against specific strategic deviations. With the small randomised twist that RPPLNS provides, we show that it maintains properties 1-3, and demonstrate that honest mining is robust to a larger class of strategic deviations than what is shown with PPLNS.

Properties of RPPLNS:

Suppose that m_1 is honest with hash power α.

1. Fairness: their expected per-turn block reward is $\frac{\alpha}{D}$ in an RPPLNS mining pool.
2. Variance Reduction: the variance of their per-turn block reward is $\frac{1}{D^2}(\alpha - \alpha^2) + \frac{\alpha}{ND}$
3. Robustness to pool hopping: Over a finite time-horizon, the payoff m_1 obtains is only a function of the amount of time they dedicate to mining for the pool.
4. Incentive Compatibility: given the simple state space of RPPLNS, we provide a recursion which allows us to empirically compute hash rates and bag distributions where honest mining dominates an arbitrary strategic deviation

Incentive Compatibility: One advantage of RPPLNS over PPLNS is that the state from the perspetive of the strategic miner is just a tuple (ℓ, s, b), where ℓ is the number of published shares, s is the number of private shares and b the number of private blocks. In PPLNS, ℓ would have to be the whole queue.

We can recursively define the gain of a strategic miner after k steps as $g_k(\ell, s, b)$. This can be computed using dynamic programming and compared against expected the honest mining reward. In the figures to the right we show the optimal strategy for m_1, given α, the hash power of the honest in-pool miners and the fraction of shares controlled by m_1.

Notice that as long as the fraction of shares controlled by m_1 is not too far from his hash power, the result is honest behaviour. For the case where his hash power is either too high or the fraction of shares he controls too different, strategic behaviour (which also happens for PPLNS and can be explained theoretically) does occur. These situations are unlikely to happen in practice.

Experiments