On the Instability of Bitcoin Without the Block Reward

Miles Carlsten
carlsten@cs.princeton.edu

Harry Kalodner
kalodner@cs.princeton.edu

S. Matthew Weinberg
smweinberg@princeton.edu

Arvind Narayanan
arvindn@cs.princeton.edu

ABSTRACT

Bitcoin provides two incentives for miners: block rewards and transaction fees. The former accounts for the vast majority of miner revenues at the beginning of the system, but it is expected to transition to the latter as the block rewards dwindle. There has been an implicit belief that whether miners are paid by block rewards or transaction fees does not affect the security of the block chain.

We show that this is not the case. Our key insight is that with only transaction fees, the variance of the block reward is very high due to the exponentially distributed block arrival time, and it becomes attractive to fork a "wealthy" block to "steal" the rewards therein. We show that this results in an equilibrium with undesirable properties for Bitcoin's security and performance, and even non-equilibria in some circumstances. We also revisit selfish mining and show that it can be made profitable for a miner with an arbitrarily low hash power share, and who is arbitrarily poorly connected within the network. Our results are derived from theoretical analysis and confirmed by a new Bitcoin mining simulator that may be of independent interest.

We discuss the troubling implications of our results for Bitcoin's future security and draw lessons for the design of new cryptocurrencies.

1. INTRODUCTION

The security of Bitcoin's consensus protocol relies on miners behaving correctly. They are incentivized to do so via mining revenues under the assumption that they are rational entities. Any deviant miner behavior that outperforms the default is thus a serious threat to the security of Bitcoin.

Miners receive two types of revenue: block rewards and transaction fees. The former account for the vast majority of miner revenues at the beginning of the system, but it is expected to transition to the latter as the block rewards dwindle (specifically, they halve every four years). There has been an unexamined belief that in terms of the security of the block chain (including incentives of the mining game), it is immaterial whether miners receive (say) 25 bitcoins in each block as a block reward or 25 bitcoins *in expectation* as transaction fees.

Illustrative example (Figure 1). Imagine a popula-

This is an extended version of our paper that appeared at ACM CCS 2016. Some of the figures have been updated with more accurate versions due to improvements to our simulator.

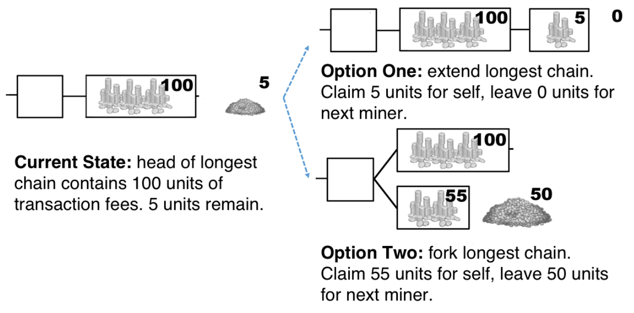
100
5
Current State: head of longest chain contains 100 units of transaction fees. 5 units remain.
100
5
0
Option One: extend longest chain. Claim 5 units for self, leave 0 units for next miner.
100
55
50
Option Two: fork longest chain. Claim 55 units for self, leave 50 units for next miner.

Figure 1: One possible state of the block chain and two possible actions a miner could take.

tion of rational, self-interested miners. Consider a block chain with blocks of exponentially distributed rewards, as we expect when the fixed block reward runs out. A miner has numerous options to consider when mining, but let's focus on just two possibilities. She could extend the longest chain (Option One), obtaining a reward of 5 and leaving a reward of 0 for the next miner (at least until more transactions arrive). Alternatively, she could fork it (Option Two), obtaining reward of 55 while leaving a reward of 50 Bitcoin unclaimed. The Bitcoin protocol dictates Option One, but a quick reasoning suggests that Option Two is better.

To reason about this correctly, we must consider which strategies the *other* miners are using. For instance, if all other miners follow the heuristic of mining on the block they heard about first in the case of a 1-block fork (and if there is no latency in the network), then forking is ineffective, and Option One is clearly superior. On the other hand, since other miners are rational, perhaps they will choose to build on the fork instead of the older block, in which case Option Two would yield more rewards.

Examples like these reveal novel incentive issues that simply don't arise when block rewards are fixed. The goal of this paper is to understand the potential impact on Bitcoin's stability by investigating the mining game in the regime where the block reward has dwindled to a negligible amount, and transaction fees dominate mining rewards. We find new and surprising incentive issues in a transaction-fee regime, *even assuming that transactions (and associated fees) arrive at a steady rate.* To be clear: the incentive issues we uncover arise *not* because transaction fees may arrive erratically, but because the time-varying nature of transaction fees allows for a richer set of strategic deviations that don't arise in the block-reward model.

At a high level, there is an analogy with *pool hopping* [22]. With certain mining pool reward schemes, the miner's expected reward for participation varies over time, depending on how many shares have been contributed since the pool found its last block. The concern is that miners would respond by "hopping" in real time to the pool that maximizes their expected rewards. For another illustration of this theme, consider a future where there are multiple cryptocurrencies with time-varying rewards which can be mined by the same hardware. Perhaps this will give rise to *coin-hopping*, i.e., miners hopping to the cryptocurrency with the largest transaction fee pool.

Contribution 1: A mining strategy simulator. While we establish a number of theoretical results in Sections 5 and 6, the variety of possible parameters and assumptions makes it completely infeasible to pose a perfectly accurate Game-Theoretic model of Bitcoin that is also tractable. To fill the gaps and to confirm our theoretical results, we've built a mining strategy simulator. Theoretical results in simple yet principled models provide good intuition to guide practice, and simulations of more complex scenarios confirm that these results have applicability to more realistic models where mathematical proofs are intractable.

Miners in our simulation learn over time which strategies are successful using *no-regret learning algorithms* that iteratively update a probability distribution over strategies (Section 4.2). Our simulator is versatile and allows modeling different numbers of miners, hash power distributions, network latencies, and reward schemes. We show how it allows researchers to quickly prototype and study new settings within this parameter space. The simulator does have limitations: it cannot model mining pools or a non-constant arrival rate of transactions. We have made the simulator open source.[1]

In addition to the versatility of settings, our simulator allows exploring a large space of mining strategies, defined by the miner's responses to three questions: *which* block to extend, *how much* of the outstanding transactions to include in the block, and *when* to publish found blocks. We define a formal language to compactly express any strategy in this space (Section 4).

Contribution 2: Undercutting attacks. The focus of this paper is on analyzing deviant mining strategies in the transaction-fee regime that can harm Bitcoin's security. We begin with the observation that if there is a 1-block fork, it is more profitable for the next miner to break the tie by extending the block that *leaves the most available transaction fees* rather than the oldest-seen block. We call this strategy PettyCompliant.

Once any non-zero fraction of miners is PettyCompliant, it enables various strategies that are more aggressive and harmful to Bitcoin consensus. We call this the undercutting attack, where miners will actively fork the head of the chain and leave transactions unclaimed in the hope of incentivizing PettyCompliant miners to build on their block.

In some scenarios, our simulation reveals a non-equilibrium with increasingly aggressive undercutting. But with an expanded strategy space, and suitable assumptions, we are able to prove that an equilibrium exists. However, it is one where miners include only a fraction of available transactions into their blocks. This results in a backlog of transactions whose size grows indefinitely with time. We confirm this result using simulation.

Accurately predicting the steady-state mining behavior requires modeling a vast number of variables such as miners' cost structure, and is not the goal of our work. Instead, our results can be seen as an informal "lower bound" on the departures from compliant behavior that are likely in a transaction-fee regime. We can realistically predict that PettyCompliant miners will arise, and that the existence of such miners opens the field for various more aggressive strategies (Section 5).

Contribution 3: Revisiting selfish mining. We revisit the selfish mining strategy of Eyal and Sirer [9] and show that, contrary to intuition, it performs even better in the transaction-fee regime than in the block-reward regime. Next, we propose a more sophisticated selfish mining strategy that accounts for the non-uniformity of rewards and outperforms both default mining and "classic" selfish mining. Worse, unlike classic selfish mining, this strategy works for miners with arbitrarily low hash power and regardless of their connectedness in the Bitcoin network. Moreover, the attack is profitable as soon as it is deployed, whereas classic selfish mining only becomes profitable after a two-week difficulty adjustment period, arguably giving the community a crucial window of time to detect and respond to such an attack [10]. We validate these results via both theory and simulation (Section 6).

Impact on Bitcoin security. If any of the deviant mining strategies we explore were to be deployed, the impact on Bitcoin's security would be serious. At best, the block chain will have a significant fraction of stale or orphaned blocks due to constant forks, making 51% attacks much easier and increasing the transaction confirmation time. At worst, consensus will break down due to block withholding or increasingly aggressive undercutting.

This suggests a fundamental rethinking of the role of block rewards in cryptocurrency design. Nakamoto appears to have viewed the block reward as a necessary but temporary evil to achieve an initial allocation of bitcoins in the absence of a central authority, with the transaction fee regime being the ideal, inflation-free steady state of the system. But our work shows that incentivizing compliant miner behavior in the transaction fee regime is a significantly more daunting task than in the block reward regime. Perhaps instead, designers of new cryptocurrencies must resign themselves to the inevitability of monetary inflation and make the block reward permanent. Transaction fees would still exist, but merely as an incentive for miners to include transactions in their blocks.

2. RELATED WORK

Several recent works analyze incentives in Bitcoin mining. Some examples include [12] and [8], which analyze how strategic mining pools may attack competing pools in various ways, and [16], which analyzes how strategic Ethereum miners can trick others into wasting their computational power verifying the validity of complex scripts. Understanding miner incentives in the Bitcoin system is important — there is empirical evidence that miners/mining pools are willing to attack others in order to maximize their own profits (e.g. launching DDoS attacks against other pools) [24].

[1]https://github.com/citp/mining_simulator

Eyal and Sirer develop the selfish mining attack [9], a deviant mining strategy that enables miners to get more than their fair share of rewards. We build on their results in Section 6. Other works, notably Sapirshtein et al. [23] have analyzed selfish mining in more detail using Markov Decision Processes (MDP). In an MDP, a player moves through a discrete state space and tries to maximize reward (the state-transition function and reward function are probabilistic). This makes it a good fit for modeling Bitcoin mining. In the fixed-reward model, states are discrete. In the transaction fees model, states are continuous, so we cannot apply MDP machinery directly. Still, our analysis takes an MDP-like approach. In more recent work, Kiayias et. al. [13] perform a theoretical analysis of various selfish mining strategies in the fixed-reward model, and proves that when miners are sufficiently small, the default mining behavior is an equilibrium.

There is some work on understanding the market for transaction fees and its relation to the block size (i.e. what fees will users have to pay in order for transactions to be included in a block?) [14, 11, 21, 18]. Our work avoids this discussion; we show that undesirable behavior emerges *even if* the market reaches an equilibrium where transaction fees are non-negligible, and arrive steadily and reliably. Interestingly, Möser and Böhme reach the same conclusion as us (that monetary inflation is a preferable mechanism to transaction fees) through very different methods [18].

On the simulation side, numerous prior works have developed simulators for some aspect of Bitcoin. Some simulators are aimed at aspects of Bitcoin aside from strategic mining, such as privacy [3], or the peer-to-peer network [17]. Those developed in [9] and [8] also focus on simulating deviant mining strategies, but our understanding is that these simulators are tailor-made for the specific deviant strategies they wish to test. In comparison, our simulator allows for easy implementation of a broad range of strategies in various environments. Indeed, the versatility of our simulator is crucial for getting intuition for every result in this paper. We have made it open-source and hope it will be a useful tool for future research on strategic miner behavior.

3. MODEL AND STRATEGIES

In this section, we cover the model of Bitcoin that we investigate. We will use this model to quickly illustrate how the switch to transaction-fee dominated rewards may lead to interesting and potentially harmful effects for Bitcoin. We also introduce a formal language for describing Bitcoin strategies that we will use throughout the paper.

3.1 Model of the system

Briefly, let us describe the theme of our model before getting into specific details. The goal of this work is *not* to accurately predict exactly what mining behavior will arise in practice, but instead to uncover incentive issues that arise solely due to the time-varying nature of transaction fees versus block rewards. To this end, our model is intentionally simple because we want to isolate the effects of time-varying versus fixed rewards. As an example, we will assume that transactions (and their associated fees) arrive at a constant and continuous rate. We make this assumption not because we necessarily predict it will hold in practice, but because without it we can't guarantee that we've isolated time-varying transaction fees as the cause for any incentive issues we uncover. Put another way, our results are only made stronger by simplifying assumptions, because we are claiming that weird and undesirable consequences arise *even if* one is willing to grant simplifying assumptions.

Getting to details, the model of Bitcoin that we analyze is after the block reward has dropped to zero. That is, transaction fees are the only source of revenue for miners, and we model available transaction fees as arriving to the Bitcoin system at a constant rate. Specifically, we assume that for any time interval I of length t, the total sum of transaction fees for transactions announced during I is t (the choice of t instead of ct for some constant c is just normalization). This is different from Bitcoin as it is today with a large block reward compared to the small transaction fees, but this scenario is consistent with the vision of the long-term steady state behaviour of Bitcoin after all Bitcoins have eventually been minted.

We also assume that the difficulty is set so that a hash puzzle is solved by someone in the network every one time unit in expectation (this is again just a normalization). Additionally, for simplicity, in our theoretical results and reported simulations we model the network having no latency (unless otherwise stated). Once a miner publishes a block, all other miners immediately gain knowledge of it. Similarly, once a transaction is announced, all miners immediately learn of its existence. However, our simulator is capable of simulating latency of both types, and we do not see any substantive change in our results as latency changes.

Finally, we assume that when there are R transaction fees available, the miner can choose to include any real-valued number of transaction fees between 0 and R in their block. That is, transactions are fine-grained enough that a miner can selectively choose a set of transactions whose fees are very close to whatever real-valued target they have in mind. We believe this is a reasonable approximation due to the large number of transactions per block.

We also assume that miners always have space to include *all* available transactions. If the block size is not large enough to meet demand for transactions, we believe the qualitative content of all our results continue to hold, but the quantitative impact is mitigated. This belief is supported by the following data, taken from the most recent 1000 blocks (roughly one week's worth) as of July 11, 2016: of these 1000 blocks, 702 are full. Of the full blocks, the total sum of transaction fees ranges from 0.03 BTC to 4.51 BTC. The mean is 0.49 BTC and the standard deviation is 0.25 BTC, more than half the mean. It's unclear how to extrapolate these data to the future, but it is clear that there will indeed be fluctuation in the available fees that fit in a block. So if the block size is not large enough to meet demand for transactions, even though the available fees immediately after a block is found will not be zero (as in our analysis), they may be significantly lower than (say) ten minutes later. So even though our exact analysis will not apply in this setting, the intuition does carry over.

3.2 What could go wrong? The mining gap

Without a block reward, immediately after a block is found there is zero expected reward for mining but nonzero electricity cost, making it unprofitable for any miner to mine.

In order to provide insight as to how time-varying rewards

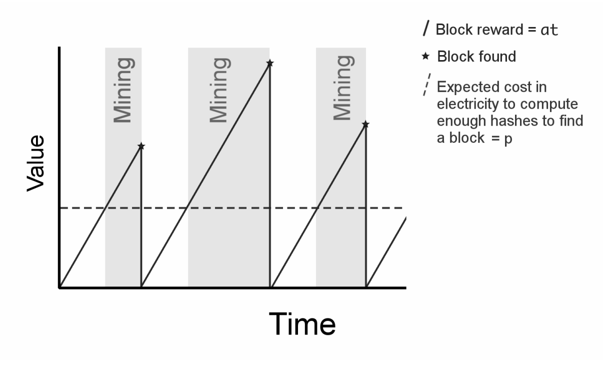

Figure 2: Illustration of Mining Gaps. Miners will only mine when the instantaneous expected reward exceeds the instantaneous cost.

could be harmful for Bitcoin, let's walk through an example. Imagine that we are in the model previously described, that all miners are using the default compliant strategy (mine on top of the longest chain, authorize all available transactions, publish immediately), but also that that miners have some cost in electricity to run their mining rigs (i.e., running one rig for t units of time costs pt Bitcoin worth of electricity). Now, immediately after a block is found, there will be no more transactions in the network to be claimed by a miner making the next block. This means that for the instant following the discovery of a new block, there is actually *zero* expected reward for mining, but a non-zero electricity cost for doing so! Figure 2 shows how to extend this reasoning to the time period beyond. Essentially, every instant your rig is running, you claim some expected reward, which increases depending on the available transaction fees. But every instant your rig is running, you also have to pay a constant amount for electricity. So the expected reward for running your rig won't exceed the cost of electricity until some minimum number of transaction fees are available to include. If a is the fraction of the total (effective) hash power that a single rig generates, then a miner must wait $t = p/a$ time steps after a block is found before mining becomes profitable again.

In Appendix A, we discuss in more detail the effects of such a *mining gap*, and find that it leads to miners mining for a smaller and smaller fraction of the time between the arrival of blocks (with the difficulty dropping to compensate). Clearly, this would have a negative impact for Bitcoin security, as the effective hash power in the network would drop, and it would become easier for a malicious miner to fork. Of course, turning a rig on and off every ten minutes may be practically infeasible. Nevertheless, this analysis illustrates that strategic miners might look for ways to deviate when the default protocol would have them wasting electricity to mine a near-valueless block.

3.3 Formal language for mining strategies

In the rest of this paper, we focus on mining strategies that always mine within the same cryptocurrency, but may deviate from the default protocol in choosing how to build blocks and what to do with them once they're found. We consider a variety of known and novel Bitcoin mining strategies. All of these can be formalized into the same general structure. At each instant, every miner makes several distinct decisions:

- Which block to extend.
- How much of the available transactions (and associated fees) to include in the block they are solving.
- For each unpublished block, whether or not to publish.

The first decision is which block to extend. As an example, the default compliant miner chooses to mine on the longest chain that they are aware of, and in the case of multiple blocks that are tied for the longest chain, they will favor mining on the first of these blocks that they became aware of. This decision forms the basis for how a mining strategy will determine which side of a fork it wants to support, or, alternatively, if the miner wants to create a new fork. The next decision is how much of the available transaction fees to claim. Again, as an example, the default compliant miner will include all of the unclaimed transaction fees they are aware of in their block. The final decision is when to publish blocks. When a miner mines a block, only they are aware of its existence. At each moment, miners can choose whether or not to alert the other miners of the block that they have found. This allows for mining strategies where miners intentionally choose to not reveal their blocks (such as selfish mining [9]).

We define the following concepts in order to more rigorously describe the mining strategies: First, for a set of transactions T, we will abuse notation and use T to also denote the total transaction fees included for transactions in T. For a block, B, we will denote $\textsc{Tx}(B)$ to be the set of transactions included in block B, and $\textsc{Rem}(B)$ to denote the *remaining* transactions after block B. That is, $\textsc{Rem}(B)$ contains all announced transactions in that are not included in B or any of its predecessors (thus, this is a set that varies over time). We will also use $\textsc{Height}(B)$ to denote the height of a block (i.e. the height of a chain that ends at block B), denoting by H the height of the current longest chain *that has been announced*,[2] and $\textsc{Owner}(B)$ to denote the miner that produced block B.

When a miner m is deciding which block of height i to extend in the case of a tie, all strategies considered in this paper first select a block that they themselves mined ($\textsc{Owner}(B) = m$). Also, all strategies in this paper avoid mining multiple blocks at the same height, so if a block with $\textsc{Owner}(B) = m$ at height i exists, it would be unique. If m did not produce any blocks at height i, the default client would then select the first block that m became aware of. So we define $\textsc{Oldest}_i^m$ to be the unique block of height i produced by miner m if it exists, or the first block of height i that m became aware of. Note that if $i = \mathrm{H}$, then this is the block m would extend using the default strategy. We also define $\textsc{Most}_i$ to be the block of height i that maximizes the remaining transaction fees (formally: $\mathrm{argmax}_{B|\textsc{Height}(B)=i}\{\textsc{Rem}(B)\}$). Note that while $\textsc{Rem}(B)$ changes over time, the block $\textsc{Most}_i$ can only change if a new block of height i is published. Finally, we denote by $\textsc{Most}_i^m$ the block of height i produced by m (if it exists), or the block of height i that maximizes the remaining transaction fees otherwise.

[2]So for instance, if a chain of height 2 has been announced, but some miner is privately storing a chain of length 10, we would define $\mathrm{H} = 2$.

We can now formally define mining strategies we consider. We model strategies as time-driven (rather than event-driven): in every infinitesimally small time step, the miner must decide which block to extend (denoted by $\textsc{Mining}(m)$), what set of transactions to include, and for each of their own unpublished blocks, whether to publish. Note that by publishing a block B, we mean ensuring that every node in the network is aware of B and all its predecessors, and aren't concerned with exactly what physical measures m takes to ensure this. In this language, the default mining strategy would be formalized as follows:

DefaultCompliant:
The default Bitcoin mining strategy, including all available transactions, mining on the end of the longest chain, choosing the older block in a tie, and publishing all blocks.

Which Block: $\textsc{Mining}(\mathrm{m}) = \textsc{Oldest}^{m}_{\mathrm{H}}$.

How much: include $\textsc{Rem}(\textsc{Mining}(m))$.

Publish(B)?: yes.

4. MINING STRATEGY SIMULATOR

In order to more clearly analyze what the game theoretic landscape will look like once the Bitcoin mining incentive becomes transaction fee based instead of block reward based, we have developed a versatile Bitcoin mining strategy simulator.[3] Here we discuss the strategies our simulator is capable of implementing, the process by which our simulator can explore a strategy space, the configurable parameters of the simulator, and its limitations.

4.1 Strategies, Rounds, and Games

We first describe the basic units of our simulator and how they interact with each other before getting into details.
Strategies. The simulator is designed in such a way to be able to run any strategy that fits the strategy space detailed in Section 3.3. That is, every strategy is fully defined by a function that outputs a *block* to extend, a *set of transactions* to include, and a *rule to decide* whether to publish any found blocks. All of these functions may take as input any public information, including all published blocks and all announced transactions.
Rounds. Our simulator is *time-driven*, as opposed to event-driven. We made this decision because we want it to be easy to add new strategies to the simulator. In an event-driven simulation, new strategies would be limited by the current list of possible events. However, in our time-based simulations, any strategy that details how to make the decisions above at any moment can be easily implemented.

A *round* is the smallest unit of time in our simulator (currently, 1/600 of the time it takes for the entire network to find a block). During a round, every miner first takes as input the block chain (that they're aware of) and all transactions (that they're aware of) and decides which block to (try to) extend, and which transactions to include. Then there is a random check (as a function of that miner's hash rate and the current network difficulty) to determine whether the miner successfully found a block or not. Then, the miner decides which unpublished blocks to publish. The duration of a round is a configurable parameter, which we discuss shortly in Section 4.3.
Games. A game involves setting parameters such as choosing a number of miners, assigning their strategies and hash power, etc. (all detailed in Section 4.3). Once these parameters are set, a game runs for several rounds, and keeps track of the rewards earned by each miner.
Simulations. A simulation might consist of a single game (to see how certain strategies fare against each other), or several games with parameter adjustments in between. For example, in order to model miners who learn over time, we have them play several games and decide which strategies to use in future games based on results of past games. In principle, any parameters can be adjusted between games.

4.2 Strategy exploration

For several of our simulations we want miners to utilize the strategies that are doing the best, to simulate how strategic miners might adapt over time. In order to accomplish this, we run several games, with hundreds of miners in each game. Miners choose strategies proportional to how successful those strategies have historically done. Formally, miners in our simulator perform *no-regret learning*, a standard notion of learning that is popular in game theoretic contexts. This is due to the fact that in any repeated game where each player separately performs no-regret learning, the repeated play converges to a *coarse correlated equilibrium* [1, 2]. Moreover, numerous simple no-regret learning algorithms are known that converge quickly (i.e. in a number of rounds sublinear in the number of possible strategies) [5, 6, 4, 15]. If a miner has no regret, their total reward across all of time is at least as good as had they instead picked "the best" strategy and used it in every game. Similarly, a coarse correlated equilibrium is a joint distribution over strategy profiles such that every miner gets more expected payoff by following the equilibrium than deviating to any possible strategy.

These learning algorithms all maintain a *weight* for every strategy, and adjust the weights of the strategies from game to game depending on how well they're doing. Our simulator offers two alternatives for these update rules. The first alternative is an exact implementation of the EXP3 algorithm for learning with adversarial bandits [5, 6]. This update rule provides a theoretical guarantee on the regret of each miner as a function of the number of games played and a tunable parameter in the update rule, ϵ. The second alternative is based on the multiplicative weights update rule (MWU) for learning with experts [4, 15]. We find that MWU is computationally expensive, so we use a less expensive proxy instead. That means there is no theoretical guarantee on the regret bounds. But in practice this update rule is significantly faster and does converge quickly to coarse correlated equilibrium. For a further discussion of these update rules, see Appendix B.

All of the figures included in this paper were generated from simulations using EXP3, so they come with a theoretical guarantee that all miners in the simulation have no regret.

[3]While this is the original motivation for developing our simulator, it is indeed capable of simulating non-zero block reward as well — more on that in Section 4.3.

4.3 Versatility

Our simulator has many configurable parameters:

Strategies. Just to reiterate: every miner in our simulator is assigned a time-driven strategy that chooses which block to extend, how many transactions to include, and whether to publish any found blocks. Any strategy that fits this framework can be implemented in the simulator. To design a new strategy, a user would create a new function that takes as input the current public state of Bitcoin network (the blockchain and available transaction fees), and the miner who is using the strategy. The function would then use this information to determine which block to extend, and how many of the transaction fees to include in the next block. Finally, the user would go to the publication rules and add a rule for how the strategy should choose whether or not to publish any found blocks.

Hash Power. Every miner m is assigned a hash power α_m. Any number of miners, and any α_m such that $\sum_m \alpha_m = 1$ can be supported.

Round Duration. The size of a round can be set so the network finds a block every r rounds in expectation, for any $r > 0$.

Rewards. At the end of each game, miners are rewarded based on their blocks within the longest chain. The reward they receive is b per block (fixed reward), plus any transaction fees. a transaction fees accrue in the system every round. Both of these parameters are configurable.

Costs. There is a configurable parameter c_m for every miner m that denotes the cost (i.e. in electricity) for miner m to mine. For our simulations, we always set $c_m = 0$ because we aren't looking at this aspect of mining.

Latency. If desired, latency can be introduced to the simulation. There is a configurable parameter λ such that when blocks are published, it takes λ rounds before other miners are aware of this blocks existence. Latency in hearing about transactions can also be implemented — it is currently easiest to do this by modifying strategies to randomly "pretend" they haven't heard of some transactions.

Learning parameter. Our learning rules are parameterized by an $\epsilon \in [0, 1/2]$. For EXP3, it is customary to set $\epsilon \approx \sqrt{n \ln n/T}$, where n is the number of strategies considered and T is the number of games played. For MWU (and our "MWU-like" update rule), it is customary to set $\epsilon \approx \sqrt{\ln n/T}$. Larger ϵ encourages beliefs (about the strength of strategies) to be updated rapidly in response to recent games. Smaller ϵ encourages waiting for more evidence before updating beliefs.

Atomic versus Non-Atomic Miners. We say miners are *atomic* if there are finitely many of them, and each has a finite fraction of the total hash power. Such miners may have an interest in sacrificing immediate gains related to a block mined now in order to achieve greater gains for blocks mined in the future. Non-atomic miners are infinitesimally small, but there are infinitely many of them. When such miners find a block, they are only interested in maximizing their gains related to that block (because they will never find another block in the future).

Obviously our simulation cannot create infinitely many miners, but we can functionally simulate them. To simulate that an α fraction of non-atomic miners are using strategy s, we instead create a single atomic miner with an α fraction of the hash power, and ensure that all of this miner's strategic decisions take as input only the public information available to the entire network, and does not treat "their own" blocks any differently than generic blocks.

Of course, the real world is atomic. But it is extremely helpful to compare simulation results between the two models to isolate behavior that arises only when miners are atomic (example: selfish mining), as intuitively this behavior "gets worse" with big miners (as with selfish mining).

4.4 Implementation and performance.

The simulator is written in C++, and has a running time proportional to the product of the number of games, the number of rounds per game, and the number of miners. We find that for accurate results, the games need to include enough rounds so that that for every strategy, the miners using it together find tens of blocks. We also find that it takes on the order of a few hundred thousand games for our learning algorithms to converge to an equilibrium. On a commodity laptop with a 2.7 GHz Intel Core i5 processor, running a simulation of 1000 games with 200 miners, an average interarrival time of 600 rounds, and a total of 6,000,000 rounds ($\approx$10,000 blocks will be created), takes approximately 22 seconds.

Limitations. A current limitation of the simulator is that the transaction fees can only be modeled as coming in at a uniform rate in time. Additionally, the simulator is not capable of modeling mining pool dynamics beyond treating them as a single miner with hash power equal to that of the pool. This doesn't allow for consideration of attacks such as those presented in [8].

5. NEW DEVIANT MINING BEHAVIOR

In this section, we examine what deviant mining behavior might unfold in the transaction fees model that doesn't arise in the block-reward model. Specifically, we argue that:

- It is reasonable to expect self-interested miners to become PettyCompliant instead of DefaultCompliant once transaction fees take over.
- The existence of PettyCompliant miners in the network opens the field for a range of aggressive strategies with detrimental effects to Bitcoin's stability.

5.1 Phase One: Petty compliant

Observation: The default client behavior of mining on the oldest block is not optimal. Miners can do strictly better by mining on the block that leaves the most transactions fees unclaimed.

Consider the case where there is a fork: two blocks are tied for longest chain. The traditional behavior, and the one programmed into the default client,[4] would have the miner select the older of the two potential block heads. However, there is really no cost for that miner instead to tie-break arbitrarily. In particular, if the miner is planning to include all unclaimed transactions in their block, it would be in that miner's interest not to mine on the oldest block, but instead the block that leaves the most remaining fees. Therefore, a strategic miner would want to mine on $\text{Most}_{\text{H}}^{m}$ instead of $\text{Oldest}_{\text{H}}^{m}$. We call this strategy *petty compliant*,

[4]Note: this is not a self-enforcing part of the protocol. It's purely client-side behavior.

as it is still mining on a longest chain, including all available transactions, and publishing all blocks that are found (like a default compliant miner). It is just tie-breaking between longest chains in a "petty" way to achieve greater revenue.

PettyCompliant:
Mine like a default compliant miner, except when choosing between two sides in a fork; mine on the block that has claimed the fewest transaction fees.

Which Block: $\text{Mining}(m) = \text{Most}_{\text{H}}^{m}$.

How much: include $\text{Rem}(\text{Mining}(m))$.

Publish(B)? yes.

If forks ever exist, then PettyCompliant strictly outperforms DefaultCompliant. The two are identical except for the case where the miner is required to choose between two equal height blocks to mine on. In this case PettyCompliant always makes the decision to mine in a location that maximizes their rewards, and DefaultCompliant might not. In our mining strategy simulator, we compare DefaultCompliant to PettyCompliant and do in fact see that PettyCompliant outperforms DefaultCompliant, regardless of the breakdown of other miners in any simulation where there is enough latency (in learning of both new blocks and transactions) that forks naturally occur.

Note that the existence of petty compliant miners is not necessarily harmful by itself: so what if miners are tie-breaking differently in the rare event that forks naturally occur? The problem arises when other strategic miners notice the existence of petty compliant miners and choose to exploit this with more aggressive tactics. We'll see some examples of this in the remainder of this section. The existence of PettyCompliant miners impact other deviant strategies in surprising ways too. For example, a selfish miner (discussed more in Section 6), performs better against PettyCompliant miners than DefaultCompliant.

5.2 Phase Two: Lazy Undercutting

Observation: Once some fraction of miners is petty compliant, other miners may profit by intentionally forking the chain.

The key insight for more aggressive strategies is that a deviant miner can incentivize petty compliant miners to extend their block, even if an older block of the same height was discovered several minutes earlier, for instance, by extending that block's direct predecessor and including slightly fewer transaction fees. If the current unauthorized transaction fees are substantially fewer than those included by the current Most_{H}, then maybe it is in a miner's interest to try and replace Most_{H} with a new block of height H, instead of continuing on top of it. We call this *undercutting.*

So what might a strategic miner do to take advantage of this? They might first compare between the maximum rewards they could get by continuing versus undercutting (while still becoming the new Most_{H}), and mine on top of whichever block yields greater rewards. Then, to protect themselves with certainty against future undercutters using the same rule, they could take half of the remaining transactions. Because of the somewhat lax reasoning used to choose these parameters, we call this strategy LazyFork.

While the existence of PettyCompliant miners themselves is relatively benign, the existence of LazyFork miners would be bad: they frequently decide to intentionally orphan blocks in order to achieve greater rewards. In addition to creating uncertainty about when blocks are "safely" in the eventual longest chain, this decreases the effective hash power of the network and makes Bitcoin more prone to double spend attacks. For cleanliness in formally defining LazyFork and other undercutting strategies, we introduce the notation $\text{Gap}_i = \text{Rem}(\text{Most}_{i-1}) - \text{Rem}(\text{Most}_i)$, the maximum transaction fees that a miner could include while mining on top of Most_{i-1} to become the new Most_i.

LazyFork:
Forks the blockchain if the head block is more valuable than the unclaimed transaction fees it leaves behind. Only takes half of the possible transaction fees to prevent other lazy forkers from forking their block.

Which Block:
```
if $\text{Owner}(\text{Most}_{\text{H}}^{m}) = m$ or $\text{Rem}(\text{Most}_{\text{H}}^{m}) \geq \text{Gap}_{\text{H}}$
    $\text{Mining}(m) = \text{Most}_{\text{H}}^{m}$.
else
    $\text{Mining}(m) = \text{Most}_{\text{H}-1}^{m}$.
```

How much: include $\text{Rem}(\text{Mining}(m))/2$.

Publish(B)?: yes.

5.3 Phase Three: Aggressive Undercutting

Simulation result: increasingly aggressive undercutting behavior evolves when miners strategize.

Once miners consider undercutting, they may also try to aggressively optimize the tradeoff between maximizing the transaction fees included in blocks they mine and minimizing the chance that their block will be undercut by other miners in the system (as opposed to using the less-principled reasoning of LazyFork). We define these strategies so that when they are presented with $\text{Rem}(\text{Mining}(m)) = x$, they will authorize $f(x)$ transactions, for some $f(\cdot)$ with $f(x) \in [0, x]$ for all x, and call them *forkers.*

While in principle, forkers could consider going back several blocks to undercut, the strategies we study only consider mining on top of a block of height H or H − 1. Certainly, it would be an interesting direction for future work to see if any additional gains can be achieved by considering blocks of height H − 2 or less, but already we uncover interesting behavior when forkers go back just a single block.

A function forking miner looks at potential blocks at height H that they could extend, and within this set considers extending only $\text{Most}_{\text{H}}^{m}$, since it leaves the most remaining transaction fees. If a miner indeed chooses to mine on top of $\text{Most}_{\text{H}}^{m}$, we call this *continuing.* They also look at potential blocks of height H − 1, again considering only extending the block $\text{Most}_{\text{H}-1}^{m}$ from this set. If a miner indeed chooses to mine on top of $\text{Most}_{\text{H}-1}^{m}$, we call this *undercutting.* When deciding whether to continue or undercut, a forker simply observes that they will choose to claim $f(\text{Rem}(\text{Most}_{\text{H}}^{m}))$ by continuing, versus $\min\{f(\text{Rem}(\text{Most}_{\text{H}-1}^{m})), \text{Gap}_{\text{H}}\}$ if they undercut (the min is taken because they must actually undercut in order to incentivize future miners to select their

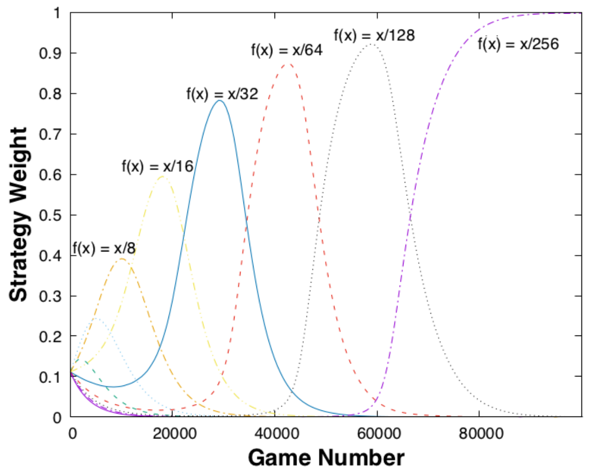

Figure 3: Normalized weights of different linear coefficient function forking strategies over a series of games. Strategies that are slightly more aggressive than the most common strategy perform the best and have their normalized weights increase. This simulation had 200 miners, 9 strategies, 10,000 blocks per game and an ϵ value of $.01$.

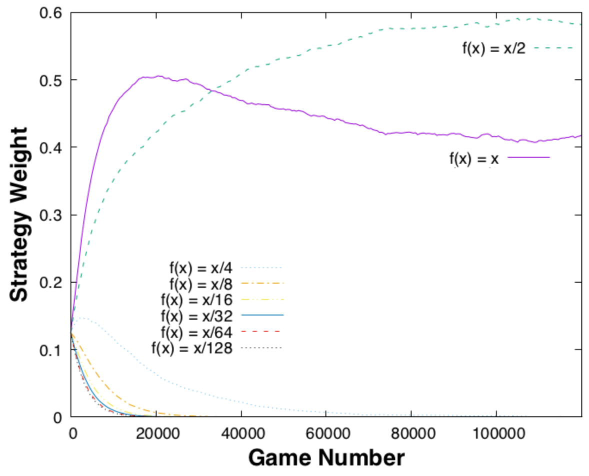

Figure 4: This is a simulation of 8 *atomic* miners. The simulation parameters are otherwise configured the same way as Figure 3. We see that when there are a small number of atomic miners the more aggressive undercutters are no longer effective since they are beaten by more gentle forkers who are lucky enough to mine two blocks in a row.

block). So for a given f, we can define:

$$\textsc{Val}_{\text{cont}}(f) = f(\textsc{Rem}(\textsc{Most}^m_{\text{H}}))$$
$$\textsc{Val}_{\text{under}}(f) = \min\{f(\textsc{Rem}(\textsc{Most}^m_{\text{H}-1})), \textsc{Gap}_{\text{H}}\}$$

If $\textsc{Val}_{\text{cont}}(f) > \textsc{Val}_{\text{under}}(f)$, then more rewards can be achieved by continuing. Otherwise, more rewards can be achieved through undercutting. Formally, for any function $f(\cdot)$, this induces the following formal strategy:

```
FUNCTION-FORK(f):
Always takes a certain function, f(·), of the possible transactions it could claim. Always mines in the location to maximize the size of the block they would make, with the constraint that if they fork, they must undercut.

Which Block:
  if $\textsc{Owner}(\textsc{Most}^m_{\text{H}}) = m$ or $\textsc{Val}_{\text{cont}}(f) > \textsc{Val}_{\text{under}}(f)$
      $\textsc{Mining}(m) = \textsc{Most}^m_{\text{H}}$.
  else
      $\textsc{Mining}(m) = \textsc{Most}^m_{\text{H}-1}$.

How much:
  if $\textsc{Mining}(m) = \textsc{Most}^m_{\text{H}}$
      include $\textsc{Val}_{\text{cont}}(f)$.
  else
      include $\textsc{Val}_{\text{under}}(f)$.

Publish(B)?: yes.
```

Any reasonable choice of $f(\cdot)$ will be monotonically increasing, which means that $f(\textsc{Most}^m_{\text{H}-1})$ will always be larger than $f(\textsc{Most}^m_{\text{H}})$, so the decision on whether to continue or undercut will come down to a comparison of $f(\textsc{Most}^m_{\text{H}})$ versus $\textsc{Gap}_{\text{H}}$.

One natural family of $f(\cdot)$ to consider is linear functions (that is, $f(x) = kx$ for some $k \in [0, 1]$). If we take a group of these strategies, and let *non-atomic* strategic miners learn over many games which perform best, we get the plot in Figure 3. What we see is the following: when the majority of miners are using $\textsc{Function-Fork}(kx)$, the best response is to use $\textsc{Function-Fork}(k'x)$, for k' a little smaller than k, (i.e. to undercut just a little bit more aggressively). So eventually the smallest coefficient in our simulation becomes dominant.

If we instead consider *atomic* miners, we observe the behavior in Figure 4 — less aggressive undercutters remain dominant. This is because even when other miners are aggressively undercutting, each miner still has a decent chance to get their block accepted "for free," by mining two blocks in a row. Note that simulation is vital to this understanding due to the large number of parameters to consider.

5.4 An Undercutting Equilibrium

Analytical result: An equilibrium exists where all miners use the same undercutting strategy. It induces a growing backlog of transactions.

Linear function-forking is of course a natural class of strategies to consider, but our simulations in the previous section show that long-term behavior may be erratic if miners only consider these strategies. Our goal in this section is to understand what undercutting behavior is stable.

Our approach is to find a function $f(\cdot)$ such that $\textsc{Function-Fork}(f)$ is an *equilibrium*. That is, as long as every other miner is using the strategy $\textsc{Function-Fork}(f)$, it is in your interest to do so as well. In other words, we would like to find an f such that $\textsc{Function-fork}(f)$ is a best-response to the case when all other miners themselves use $\textsc{Function-fork}(f)$. We provide now intuition for why the $f(\cdot)$ we present yields an equilibrium.

So what does it mean for a strategy to be a best-response to other miner behavior? Recall that a strategy proposes which block to extend, how many transaction fees to claim, and which blocks to publish as a function of the currently held information. A strategy is a best response if it maximizes the miner's expected reward (taking into account future events, and in particular the probability that the current block is in the eventual longest chain) over all potential strategies that miner could have used instead. In particular, a best-response must be at least as good as all other strategies that mine at the same location and publish the same blocks (but differ in which transactions to include).

To get some intuition for what conditions a potential equilibrium must satisfy, let's first consider the decision facing a miner who has already decided to continue on top of the longest chain and is just deciding how many transaction fees to include. If F denotes the number of transaction fees included, define $\pi(F, f, x)$ to be the probability that this block is included in the eventual longest chain, conditioned on including F BTC worth of transaction fees in the block, all other miners using strategy Function-Fork(f), and $x = \text{Rem}(\text{Most}_{\text{H}}^{m})$ (note that π is well-defined). Then the miner's expected reward, should they be fortunate enough to find a block right now would be $F \cdot \pi(F, f, x)$.

A best-response would then be to include $\text{argmax}_{F \leq x}\{F \cdot \pi(F, f, x)\}$ transaction fees. The strategy Function-Fork(f) would recommend including $f(x)$ transaction fees. So for Function-Fork(f) to be a best-response to other miners using Function-fork(f), it better be the case that $f(x) \in \text{argmax}_{F \leq x}\{F \cdot \pi(F, f, x)\}$ for all x. Note that this is a somewhat strong condition on f, as the fact that the *other* miners are using Function-fork(f) affects $\pi(F, f, x)$, whereas we also want *this* miner's best response to have $f(x) \in \text{argmax}_{F \leq x}\{F \cdot \pi(F, f, x)\}$.

At this point, we show that there is a continuous and piece-wise differentiable function $f(\cdot)$ that satisfies this condition. We also show that combined with the fact that $f(\cdot)$ is monotonically non-decreasing, this is sufficient for Function-fork(f) to be an equilibrium under some assumptions (which we will discuss post-theorem). In the theorem statement below, W_0 is the upper branch of the Lambert W function which satisfies $W_0(xe^x) = x$ for all $x \in [-1/e, \infty)$, and $W_0(x) \in [-1, \infty)$. The "Furthermore..." portion of the theorem is proved by showing a connection between the number of backlogged transactions and an unbiased single-dimensional random walk.

Theorem 5.1. *For any constant $y \leq 1/2$ such that $2y - \ln(y) \geq 2$,*[5] *define:*

$$f(x) = x, \quad \forall\, x \leq y \tag{1}$$

$$f(x) = -W_0(-ye^{x-2y}), \quad \forall\, y < x < 2y - \ln(y) - 1 \tag{2}$$

$$f(x) = 1, \quad \forall\, x \geq 2y - \ln(y) - 1 \tag{3}$$

Then it is an equilibrium for every miner to use the strategy Function-fork(f) *as long as:*

- *Every miner is* non-atomic.
- *Miners may only mine on top of chains of length* H *or* H − 1.

Furthermore, in any such equilibrium, the expected number of backlogged transactions after n time steps is $\Theta(\sqrt{n})$.

[5]Such y exist. This range is $(0, \approx 0.2]$.

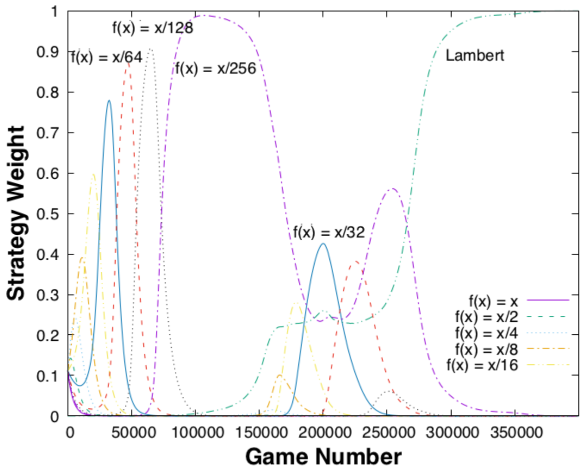

Figure 5: Plot of the Lambert function fork starting with a weight of 0.0001 and becoming the strongest strategy in a learning simulation with $\epsilon = .01$. This simulation had 100 miners, and 10,000 blocks per game. These miners are *non-atomic*.

A proof of Theorem 5.1 appears in Appendix C. To understand the impact of Theorem 5.1, first consider the *block reward* model. With non-atomic miners, DefaultCompliant is trivially an equilibrium, and this result is robust to general models of latency (proof in Appendix D). But as we move to atomic miners, strategies like selfish mining arise and equilibria get messy (if they exist at all). Now, in the transaction-fee model, *even when miners are non-atomic, equilibrium behavior is complex and undesirable*, as we have just shown. Therefore, we should expect that analysis with atomic miners should conclude with even more chaos.

Figure 5 shows miners learning to play this equilibrium, even with various other strategies available. Observe the interplay between theory and simulation: Theorem 5.1 guides us towards a potentially strong strategy, but it is intractable to prove that the equilibrium will actually arise via learning even when (say) 99% of miners are already there. Simulation fills the gap and shows an equilibrium will indeed even when only .01% of the miners initially use the equilibrium strategy.[6] Simulation alone could not search through the infinitely many possible strategies, and theory alone cannot prove that learning converges to the desired equilibrium.

5.5 Undercutting Non-strategic Miners

Analytical and simulation result: even if 66% of miners remain default compliant, undercutting is profitable.

Our analysis and simulations in the previous sections assumed that *all* miners were strategic learners. While we clearly learn a lot from this analysis, it is perhaps more realistic to also consider a setting where some miners will stubbornly (or honestly, depending on your perspective), continue running DefaultCompliant even if it is suboptimal. If a large fraction of the miners are non-strategic,

[6]It is hard to see in Figure 5, but the weight assigned to "Lambert" is initially .0001.

then function-forking becomes immediately less profitable, because only a small fraction of the network will actually mine on top of your block when you undercut. In particular, if 100% of other miners are non-strategic, undercutting serves no purpose.

In this section, we detail results from our simulation when varying fractions of miners are non-strategic. In these simulations, we fix a fraction of the network to always mine DefaultCompliant, and play enough games until the distribution of learned strategies stabilize.[7]Figure 6 shows a stacked area plot of our simulation results for equilibria at different fractions of miners refusing to abandon DefaultCompliant. There are many interesting features of the plot, but we focus on one: even if the majority of miners choose to stay DefaultCompliant (and the rest strategize), then forking strategies start to become viable.

A theoretical analysis indeed predicts the continuing presence of FunctionFork(x) until 2/3 of the miners remain DefaultCompliant. To see this, imagine that every miner is the system is currently DefaultCompliant or PettyCompliant, and we want to see if it is profitable for a PettyCompliant miner to switch to FunctionFork(x). At any point in time, consider the current Most_H. Then if the miner runs PettyCompliant, they will always try to continue, and will get $\text{Rem}(\text{Most}_\text{H})$ should they find a block (because no one else in the network is undercutting). If instead they run FunctionFork(x), they will continue whenever $\text{Rem}(\text{Most}_\text{H}) > \text{Gap}_\text{H}$ and undercut otherwise. When they continue, they will always get $\text{Rem}(\text{Most}_\text{H})$. When they undercut, they would include Gap_H transaction fees. If the next miner to find a block is PettyCompliant (or this miner), then the undercut will be successful and the miner will receive Gap_H in rewards. But if the next block is found by a DefaultCompliant miner, the undercut fails and they get nothing. So if y is the fraction of the network that remains DefaultCompliant, we see that the expected reward obtained by FunctionFork(x) is proportional to:[8]

We emphasize that while the theory gives us a crisp understanding of what should happen when exactly 2/3 of the miners are DefaultCompliant, it is intractable to rigorously analyze the equilibria at various other fractions of DefaultCompliant miners. Thus our simulation both confirms and extends our theoretical understanding (Figure 6).

$$\mathbb{E}[\text{Rem}(\text{Most}_\text{H}) \cdot \mathbb{I}(\text{Rem}(\text{Most}_\text{H}) > \text{Gap}_\text{H})] \\ + (1-y) \cdot \mathbb{E}[\text{Gap}_\text{H} \cdot \mathbb{I}(\text{Gap}_\text{H} > \text{Gap}_\text{H})]$$

Finally, because $\text{Rem}(\text{Most}_\text{H})$ and Gap_H are i.i.d. exponential random variables with mean 1, we have that $\mathbb{E}[\text{Gap}_\text{H} \cdot \mathbb{I}(\text{Gap}_\text{H} > \text{Rem}(\text{Most}_\text{H}))] = \mathbb{E}[\text{Rem}(\text{Most}_\text{H}) \cdot \mathbb{I}(\text{Rem}(\text{Most}_\text{H}) > \text{Gap}_\text{H})] = 3/4$. Therefore, whenever $y \leq 2/3$, the reward from FunctionFork(x) is at least one, and therefore it is a better choice than PettyCompliant (which gets expected reward exactly one).

[7]Note that learning is by no means guaranteed to result in a static equilibrium at all, although in these simulations that happens to be the result.

[8]$\mathbb{E}[X]$ denotes the expectation of the random variable X, and $\mathbb{I}(E)$ denotes the indicator random variable for event E (that is 1 when E occurs and 0 otherwise).

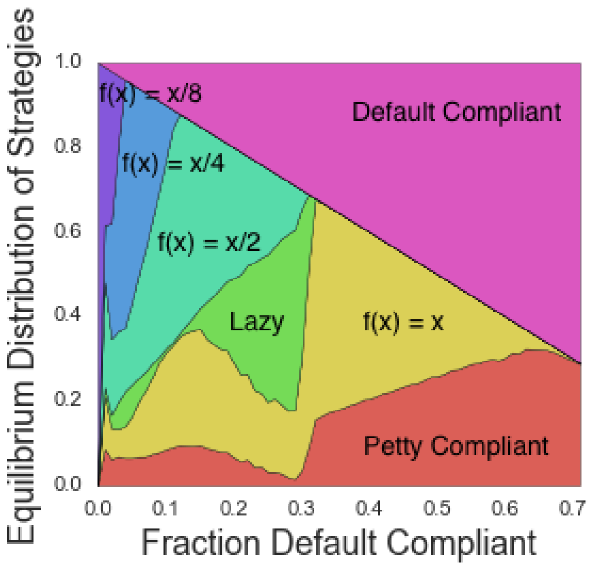

Figure 6: Stacked area chart showing the equilibrium distributions of strategies covered thus far, given that a fraction of miners will always use the default strategy. These simulations involved 100 miners, with 10,000 blocks per game. We found that the strategies would reach an equilibrium around 300000 games with $\epsilon = 0.01$.

6. SELFISH MINING WITH TRANSACTION FEES

Selfish mining is a deviant strategy first identified by Eyal and Sirer [9]. Essentially, a selfish miner chooses not to release blocks immediately upon being found, instead withholding them in hopes of tricking the rest of the network into wasting their mining power mining blocks that will be orphaned.

We find that the selfish mining strategy performs *even better* in the transaction fees model than the block-reward model. A priori, there's no reason to expect this. In this section we provide simulation results, along with some intuition and a theoretical analysis proving this. Essentially what winds up happening is that while the selfish miner mines the same *fraction* of blocks in either reward model, the selfish miner's blocks will tend to be *larger*. In the block-reward model, this doesn't matter because all blocks are worth the same, but in the transaction fees model this means the selfish miner gets greater reward.

6.1 The Selfish Mining Strategy

Analytical and simulation result: selfish mining performs slightly better in the transaction fee model.

The goal of a miner employing the selfish mining strategy is to essentially trick the other miners in the Bitcoin network to mine on top of a block that will be orphaned. By having other miners waste their power, the selfish miner is capable of exaggerating their own portion of the overall network hash-rate. Selfish miners do this by maintaining a chain in private that only they know about. When the selfish miner initially finds a block, they will not announce their block to the rest of the network. They will continue to mine on their private block, hoping to find a second block before the rest of the network finds a block.

If the miner succeeds, now they're in a very strong position: they know of a block with height $\mathrm{H}+2$, whereas the rest of the network only knows a block of height H. If the rest of the network finds the next block at height $\mathrm{H}+1$, the selfish miner can reveal their private chain and the public block will be immediately orphaned. Of course, maybe the selfish miner will find the third block as well. In this case, they're in an even better position and can waste even more of the network's power. But the point is that with a lead of two or more, the selfish miner can guarantee that the rest of the network is wasting power.

Of course, the selfish miner might also fail to find a second block before the rest of the network finds their first. In this case, they immediately release their block and hope that others hear about theirs first. Obviously this is not ideal: had they released their block immediately, they could have guaranteed that it was heard about first. So there's a tradeoff — withholding the block has a chance to give the selfish miner a private chain of length two or more, in which case the selfish miner benefits, but it could also cause their block to be orphaned, resulting in less profits.

```
SELFISH-MINE:
Selfish mining strategy from [9]. This miner hides their
blocks, which risks losing their first block, in order to try
to get the rest of the network mining in a useless location,
amplifying their own apparent hash power.

Which Block: $\textsc{Oldest}^{m}_{\textsc{Private}^m}$.

How much: include $\textsc{Rem}(\textsc{Mining}(m))$.

Publish($B$)?:
  if $\textsc{Height}(B) = \mathrm{H}$
    yes.
  elseif $\textsc{Racing}^m_{\mathrm{H}}$, and $\textsc{Private}^m = \mathrm{H}+1$
    yes.
  else
    no.
```

Assuming the selfish miner has less than half of the overall hash power of the network, they will eventually need to publish their private chain. In order to maintain our focus on the difference between transaction fees and fixed block-rewards, we consider just "vanilla" selfish mining, although it is an interesting consideration for future work to consider selfish miners who also undercut, or various other generalizations (e.g. [7, 20, 23]). Similarly to [9], we examine the potential rewards a selfish miner would receive assuming that the rest of the network is default mining. In our analysis, we also use α to denote the fraction of the total mining power possessed by the selfish miner, and γ to be the probability that in the event of a race (selfish miner is triggered to release a private block of length one) that ends with the honest portion of the network finding the next block, that the selfish miner's block is not orphaned. We introduce notation $\textsc{Private}^m$ to denote the height of the longest chain that m is aware of (at least as long as H, and possibly longer if m is keeping any blocks private). We also introduce notation $\textsc{Racing}^m_i$ to be a boolean variable that is true iff there exist two blocks B_1, B_2 with $\textsc{Height}(B_1) = \textsc{Height}(B_2) = i$, and $\textsc{Owner}(B_1) = m \neq \textsc{Owner}(B_2)$. In other words, $\textsc{Racing}^m_i$ denotes whether or not there are two competing blocks of height i, one of which was produced by m.

Analysis.

We proceed now with an analysis of the rewards obtained in the transaction fee model by a selfish miner. Parts will look similar to the analysis done in [9]. For every infinitesimally small transaction fee that arrives, we wish to compute the probability that it winds up in a block mined by the selfish miner. Note that if the selfish miner just used default mining instead, this probability would be exactly α.

The determining factor in this probability will be the size of the selfish miner's private chain. To this end, let's define the following states (same states used in [9]), and we'll compute this probability separately for each state.

- State 0: Everyone agrees on the longest chain — $\textsc{Racing}^m_H =$ `false`.
- State $i > 0$: The selfish miner m has a private chain of length i — $\textsc{Private}^m = \mathrm{H}+i$.
- State $0'$: There are competing blocks of height H, one of which was produced by the selfish miner, and the selfish miner has no private blocks — $\textsc{Racing}^m_{\mathrm{H}} =$ `true` and $\textsc{Private}^m = \mathrm{H}$.

Let f_s denote the probability that a transaction winds up in a block mined by the selfish miner in the eventual longest chain, conditioned on the system being in state s when the transaction is announced. We compute there probabilities below. If we then define p_s to be the probability that the system is in state s, we can then observe that the expected fraction of transaction fees claimed by the selfish miner is exactly $\sum_s f_s \cdot p_s$. Eyal and Sirer [9] have already computed p_s for all s. The values for p_s are:

$$p_0 = \frac{1-2\alpha}{2\alpha^3-4\alpha^2+1}$$
$$p_{0'} = \frac{(1-\alpha)(\alpha-2\alpha^2)}{2\alpha^3-4\alpha^2+1}$$
$$p_i = (\frac{\alpha}{1-\alpha})^{i-1}\frac{\alpha-2\alpha^2}{2\alpha^3-4\alpha^2+1},\ i>0$$

To complete the analysis, we just need to compute f_s for each s. Appendix E contains the derivation of f_s for all s, which are stated below:

$$\begin{aligned} f_0 &= \alpha^2+\alpha(1-\alpha)\left(\alpha+\gamma(1-\alpha)\right). \\ f_{0'} &= \alpha. \\ f_1 &= \alpha+(1-\alpha)\alpha = \alpha(2-\alpha). \\ f_i &= 1-((1-\alpha)^{i-1}(1-f_0)). \end{aligned}$$

Finally, when $\alpha \in (0, .5)$ and $\gamma \in [0, 1]$, we show in the Appendix E that the selfish miner's rewards are given by

$$\textsc{Reward}(\alpha,\gamma) =$$
$$\frac{5\alpha^2-12\alpha^3+9\alpha^4-2\alpha^5+\gamma(\alpha-4\alpha^2+6\alpha^3-5\alpha^4+2\alpha^5)}{2\alpha^3-4\alpha^2+1}$$

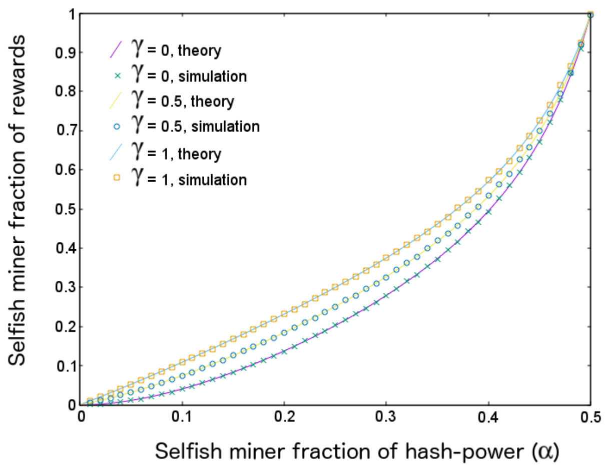

Figure 7: We see simulation matching the theory for selfish mining in a transaction based model for $\gamma = 0$, 0.5, and 1.

We make the following observations:

- Simulation confirms the above analytical formula for $\textsc{Reward}(\alpha, \gamma)$ (Figure 7)
- This function is extremely close to the reward function with block rewards ($\frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha))-\alpha^3}{1-\alpha(1+(2-\alpha)\alpha)}$) from [9]. We find, numerically, that the absolute difference never exceeds 0.026 in the region of interest.
- For $0 \le \gamma < 0.55$ (in particular, for $\gamma = 0$), for all $\alpha \in (0, 0.5)$, the reward is strictly greater in the transaction fee model than in the block reward model.

We provide some intuition for this last point. First, it is clear that the *fraction* of blocks mined by the selfish vs. default miners is independent of the reward model. So the gap must come from the size of blocks found by the respective miners. Let's assume just for the sake of example that we are in state 100 and the selfish miner has an $\alpha = 1/10$ fraction of the mining power. Almost certainly, the next un-orphaned block will be found by the selfish miner. How long will it take for this block to be found? The answer is approximately 10 time steps. This is because while the entire network finds a block roughly every time step, because the selfish miner is the only miner extending his chain (and he mines at 1/10 the speed of the full network) it will take ten times as long. What this means is that blocks found by the selfish miner while the selfish miner has a huge lead are disproportionately large compared to blocks found when the selfish miner has no lead (or a tiny lead). So even though the selfish miner wins the same fraction of blocks, some of these blocks are much larger than those won by the default miners.

A brief discussion. The main point of this section is to highlight one example of surprising incentive issues that differ between the transaction fees model and the block-reward model, *not* to argue that selfish mining becomes significantly better (the improvement is minor). Still, we wish to point out two possibly salient differences between selfish mining in the two models. First, in the block-reward model, selfish mining is actually not ever immediately profitable — it only becomes profitable once the difficulty readjusts to account for the fact that the effective mining power in the network is lower. This is because before the difficulty adjusts, the selfish miner is literally just throwing blocks away, but tricking the rest of the network into throwing blocks away at a higher rate. In the transaction fees model, selfish mining is immediately profitable — every transaction that arrives goes somewhere, so neither the selfish miner nor the default miners are throwing rewards away. Note also that our analysis in no way requires the difficulty to adjust before it becomes accurate — our analysis would hold no matter how the difficulty of hash puzzles adjusted or didn't adjust over time. Moreover, if some of the rest of the network has switched to the PettyCompliant strategy, then the selfish miner's block is actually more likely to win when a race is triggered (because it was mined earlier and therefore contains fewer transactions). So the existence of Petty-Compliant miners in the transaction fees regime indirectly improves Selfish-Mine's performance by increasing γ.

6.2 An Improved Selfish-Mine

Analytical and simulation result: in the transaction fee model, selfish miners can make the decision whether to hide their first block based on the value of the block. This improved selfish mining strictly and always outperforms both default mining and traditional selfish mining.

In this section we develop an improved selfish mining strategy. Essentially, we observe that in the transaction fees model, a selfish miner has additional information when deciding whether to hide or publish their private chain (namely, how many transactions are included). We show that, *for all* $\alpha, \gamma < 1$, our strategy strictly outperforms both default mining and "vanilla" selfish mining in the transaction fees model. Our strategy will decide to hide only "small" blocks, with at most β (some cutoff parameter chosen by the strategy as a function of α, γ) transaction fees included, but will immediately publish any "large" blocks, with more than β transaction fees in order to avoid the risk of losing them.

Selfish-Mine(β):
An improvement to the selfish mining strategy, where the miner will chose to mine as a selfish miner or a default compliant miner based on the value of the block they risk losing.

Which Block: $\textsc{Oldest}^m_{\textsc{Private}^m}$.

How much: include $\textsc{Rem}(\textsc{Mining}(m))$.

Publish(B)?

```
if $\textsc{Height}(B) = \text{H}$ or $\textsc{Tx}(B) \ge \beta$
    yes.
elseif $\textsc{Racing}^m_H$, and $\textsc{Private}^m = \text{H} + 1$
    yes.
else
    no.
```

Intuitively, imagine you are mining and find yourself solving a new block immediately after a previous block was announced and before any new transactions have been announced. This block is literally worthless, so instead of publishing, why not use it to try and selfish mine? There is

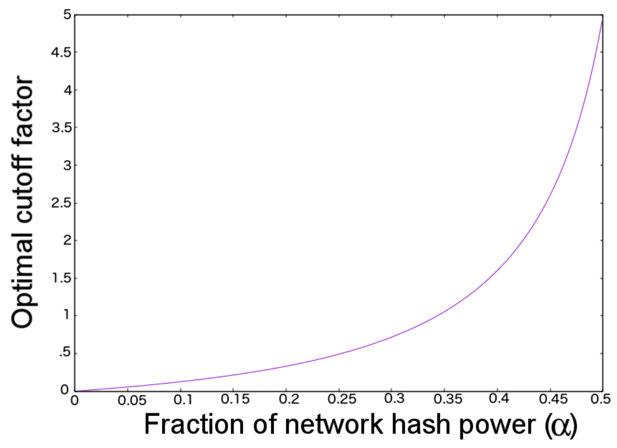

Figure 8: We show the ideal cutoff factor, β, for a selfish miner with mining power α, and $\gamma = 0$.

no cost, but a positive probability that you build a lead of two, no matter your hash power. Similarly, imagine instead that just by chance an hour goes by since the last block was found and you just solved a new block including all transactions that arrived during that period. This block is worth roughly six "normal" blocks, so why risk losing it? Unless your hash power is very close to 50%, the expected gains from selfish mining are dwarfed by the possibility of losing this unusually wealthy block. So the trick is just choosing the proper cutoff β as a function of your hash power α and network connectivity γ.

Note that SELFISH-MINE(0) = DEFAULTCOMPLIANT, and that SELFISH-MINE(∞) = SELFISH-MINE. So clearly, taking the optimal choice of β will result in a strategy that equals or outperforms both. Using an analysis similar to that of Section 6.1, we are able to compute the expected reward achieved by a miner with an α fraction of the mining power, a γ success probability of winning a race, and using strategy SELFISH-MINE(β). A derivation is included in Appendix E.2.

$$\text{REWARD}(\alpha,\gamma,\beta) = \left(\frac{1+\beta(1-\alpha)^2(1-\gamma)}{e^{\beta}-1} + 5\alpha + (1-\alpha)^2\gamma + \frac{2\alpha^2}{1-2\alpha} - 2\alpha^2\right) \times \left(\frac{\alpha(1-2\alpha)(1-e^{-\beta})}{1-2e^{-\beta}\alpha - 3(1-e^{-\beta})\alpha^2}\right)$$

Figure 8 contains a plot showing the optimal choice of β as a function of α when $\gamma = 0$. A few noteworthy points from this plot: as $\alpha \to 0$, so does the optimal β. As $\alpha \to 1/2$, the optimal β approaches ∞. Figure 9 plots our theoretical predictions against simulation results, confirming that the analysis is correct.

We conclude this section with Figure 10 plotting the (theoretical) performance of default mining, selfish mining, and selfish mining with the optimal cutoff for a range of α and $\gamma = 0$. Note that in some ranges, the gains are quite significant. Specifically, when $\alpha = 1/3$, both selfish mining and default mining achieve expected reward of $\approx 1/3$, but selfish mining with the optimal cutoff achieves an expected reward of $\approx .38$, a 13.6% increase!

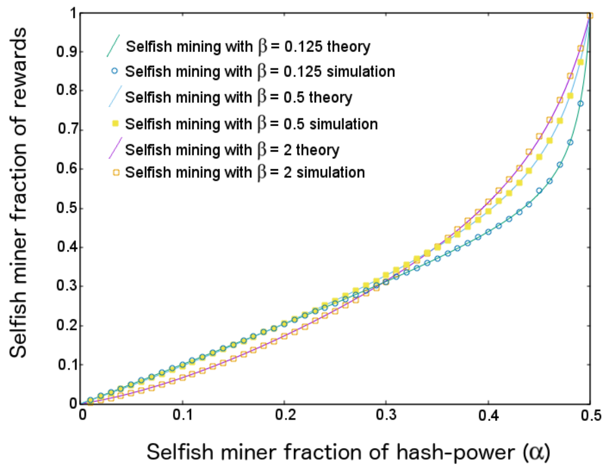

Figure 9: Theory matching simulation for a variety of cutoff thresholds for selfish mining, all with $\gamma = 0$. The smaller cutoffs do better for a miner with a smaller hash-power (α) and the larger cutoffs do better with a larger hash-power. Intuitively, this makes sense as a more powerful miner should be willing to risk a larger block to try to selfishly mine.

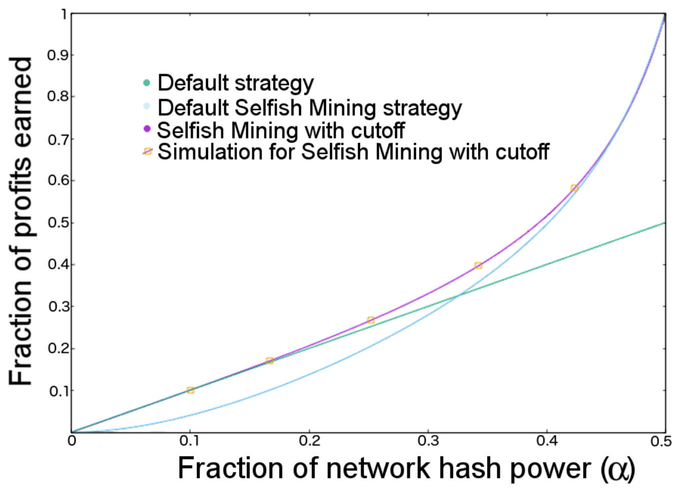

Figure 10: A selfish miner using the optimal cutoff outperforms both the original selfish mining protocol and default mining for all values of α, with $\gamma = 0$. The simulation points confirm that the theory is accurate.

7. IMPACT ON BITCOIN AND LESSONS FOR CRYPTOCURRENCY DESIGN

We have argued that deviant mining strategies in a transaction-fee regime could hurt the stability of Bitcoin mining and harm the ecosystem. In a block chain with constant forks caused by undercutting, an attacker's effective hash power is magnified because he will always mine to extend his own blocks whereas other miners are not unified. This would make a "51%" attack possible with much less than 51% of the hash power.

Many other unanticipated side-effects may arise. In the block size debate, it is frequently argued or assumed that space in the block chain will be a scarce resource and a market will emerge, with users being able to speed up the confirmation of a transaction by paying a sufficiently large transaction fee. But if miners intentionally "leave money on the table" when solving blocks, as is the case in undercutting attacks, it breaks this assumption. That is because undercutting miners are not looking to maximize the transaction fee that they can claim, and don't have a strong reason to prioritize a transaction with a high fee.[9] Put another way, the block size imposes a constraint on the total size of transactions in a block and the threat of being undercut imposes another constraint on the total fee. The two interact in complex ways. We believe that qualitatively our results will continue to hold in a world where the available block size is much smaller than the demand, but quantitatively the impact of undercutting will be mitigated (see end of Section 3.1). Still, it is an important direction for future research to understand this connection more rigorously.

Despite the variety of our results, we believe we have only scratched the surface of what can go wrong in a transaction-fee regime. To wit: we have not presented an analysis of miners whose strategy space includes both undercutting and selfish mining, primarily due to the complexity of the resulting models.

There has been scant attention paid to the transition to a transaction-fee regime. The Nakamoto paper addresses it briefly: "The incentive can also be funded with transaction fees... Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free" [19]. Similar comments on the Bitcoin Wiki and other places suggest that the community views the transition as unremarkable. Some altcoins (Monero, Dogecoin) have even opted to hasten the block reward halving time.

Our results suggest a different view. We see the block reward as integral to the stability of the mining game. At a minimum, analyzing equilibria in the transaction-fee regime appears dramatically harder than in the block-reward regime, which is a cause for concern by itself. The monetary inflation resulting from making the block reward permanent, as Ethereum does, may be a small price to pay to ensure the stability of a cryptocurrency.

8. ACKNOWLEDGMENTS

We are extremely grateful to Jiechen Chen, Kira Goldner, Anna Karlin, and Rainer Böhme for very detailed feedback on an earlier draft of this paper.

[9]They do have a weak reason: miners benefit from creating the smallest possible block for a given value of the total transaction fee they seek to claim, since smaller blocks propagate faster through the network and are less likely to be orphaned.

9. REFERENCES

[1] Calibrated learning and correlated equilibrium. *Games and Economic Behavior*, 21(1):40–55, 1997.

[2] A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[3] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in bitcoin. In *Proceedings of Financial Cryptography*, 2013.

[4] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[5] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 32(1):48–77, 2002.

[6] A. Blum and Y. Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8:1307–1324, 2007.

[7] N. T. Courtois and L. Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. *CoRR*, abs/1402.1718, 2014.

[8] I. Eyal. The miner's dilemma. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 89–103. IEEE, 2015.

[9] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[10] K. Hill. Bitcoin is not broken. Forbes, 2013. http://www.forbes.com/sites/kashmirhill/2013/11/06/bitcoin-is-not-broken/#55d4a8812568.

[11] N. Houy. The economics of bitcoin transaction fees. *Working Paper GATE 2014-07. halshs-00951358.*, 2014.

[12] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore. Game-theoretic analysis of ddos attacks against bitcoin mining pools. In *Proceedings of the First Workshop on Bitcoin Research*, 2014.

[13] A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis. Blockchain mining games. In *ACM Conference on Economics and Computation (EC)*, 2016.

[14] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of the Twelfth Annual Workshop on the Economics of Information Security (WEIS)*, 2013.

[15] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994.

[16] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2015.

[17] A. Miller and R. Jansen. Shadow-bitcoin: scalable simulation via direct execution of multithreaded

applications. In *Proceedings of the eighth workshop on Cybersecurity Experimentations and Test (CSET)*, 2015.

[18] M. Möser and R. Böhme. Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In *Workshop on Bitcoin Research*, pages 19–33, 2015.

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

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

[21] R. Peter. A transaction fee market exists without a block size limit. 2015.

[22] M. Rosenfeld. Analysis of bitcoin pooled mining reward systems. *CoRR*, abs/1112.4980, 2011.

[23] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security*, 2016.

[24] M. Vasek, M. Thornton, and T. Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *Proceedings of the First Workshop on Bitcoin Research*, 2014.

APPENDIX

A. MINING GAP

This appendix contains a theoretical analysis of the mining gaps referenced in Section 3.2. Let's consider the following simplified model: there is one style of "rig" available to miners, which costs p BTC per time unit in electricity to run. Let's first analyze what effect this has in the fixed reward model, where each block found is worth one BTC, and the difficulty is adjusted so that the time between successive blocks is one unit in expectation.

Then if there are k rigs in the network, the expected reward from running a rig for one time unit is exactly $1/k$, whereas the cost in electricity is p. So the network is sustainable as long as $1/k \geq p$, or $k \leq 1/p$. In other words, the cost of electricity imposes a hard cap on the total effective mining power of $1/p$ rigs worth. Of course, this can always be adjusted if necessary by changing the fixed reward per block. Also, it is important to point out that as long as $k \leq 1/p$, the effective hash power in the network will be k rigs worth.

Now let's consider what happens in the transaction fee model, where transaction fees arrive continuously at a rate of 1 per time unit. Miners will always turn off their rigs (/coin-hop) immediately after a block is found, because the instantaneous expected reward of running a rig is 0, but the cost is non-zero. If the current effective hash power in the network is c rigs worth, then the miner needs to wait until $x = cp$ transaction fees have arrived in order for mining to be profitable.

Now, assuming that miners are cleverly turning their rigs on and off at the right times, how many rigs must be in the network in order to attain an effective hash power of c? The rigs are all off for cp units of time, and then all k of them are turned on, and the expected time to find a block is 1 unit of time. This means that the expected time to find a block with all k units running must be $1 - cp$ (due to difficulty adjustment), whereas the expected time to find a block with c units running is 1 (because the effective hash power is c). Finally, we observe that for a fixed difficulty, if x denotes the number of rigs running, and y_x denotes the expected time for x rigs to find a block, then $x_1 \cdot y_{x_1} = x_2 \cdot y_{x_2}$ for all possible number of rigs x_1, x_2. Together, this yields the following equation:

$$k \cdot (1 - cp) = c \cdot 1$$

$$\Rightarrow k = \frac{c}{1 - cp}.$$

What do we learn from this? First, we see that no $c \geq 1/p$ can possibly be supported, just like in the fixed-reward model. On the other hand, we see that it takes an *additional* factor of $\frac{1}{1-cp}$ rigs in order to get the effective hash power of $c \leq p$ rigs. As $c \to 1/p$, the maximum possible effective hash power, this ratio approaches ∞! More quantitatively, if we plug in $c = x/p$ for $x < 1$, we see that the blow-up is $\frac{1}{1-x}$. This means the following: **In the transaction fees model, to obtain an x fraction of the maximum possible effective hash power, a multiplicative blow-up of $\frac{1}{1-x}$ rigs are necessary**. Recall that in the fixed-reward model, no blow-up is necessary.

We can also reason in the other direction: for a fixed k number of rigs in the network, what is the effective hash rate in the fixed reward model versus the transaction fees model with mining gaps? In the fixed reward model, this is easy: it's just $\min\{k, 1/p\}$. In the transaction fees model, for a fixed k, we need to solve for the c such that $k = \frac{c}{1-cp}$. This is:

$$k - kcp = c$$

$$\Rightarrow c = \frac{k}{1 + pk}$$

So for fixed k, the effective mining power of k rigs degrades by a factor of $\frac{1}{1+pk}$, which is always < 1. Note that at $k = 1/p$, every rig is 100% effective in the fixed reward model, whereas the effective mining power is just $k/2$ in the transaction fees model. We can again make a quantitative statement: **In the transaction fees model, when the raw hash power in the network is an x fraction of the maximum possible, the effective hash power degrades by a factor of $\frac{1}{1+x}$.** Recall that in the fixed rewards model, there is no degradation in effective hash power when $x \leq 1$.

B. LEARNING MINERS IN SIMULATOR

As referenced in Section 4.2, we provide two options for learning in our simulator. Let's introduce these with a clear set-up for learning.

Let there be a set of strategies a learner can use, indexed by k. At each round $i \in [T]$, the learner receives/would have received some reward $r_k^i \in [0, 1]$, which may be arbitrary. The goal is to select a sequence of strategies s_i guaranteeing:

$$\sum_{i=1}^{T} r_{s_i}^i \geq \max_k \{\sum_{i=1}^{T} r_k^i\} - c.$$

In other words, we would like to select a sequence of strategies that does nearly as well as the best strategy, assuming we knew it from the beginning. It is well-known [4] that setting $w_k^i = w_k^{i-1}(1 - \epsilon)^{r_k^i}$ and selecting s_k^i proportional to the weights w_k^i results in a guarantee with $c = \epsilon T + \ln(\text{\# strategies})/\epsilon$. Similarly, [5] shows that even if we don't learn r_k^i for

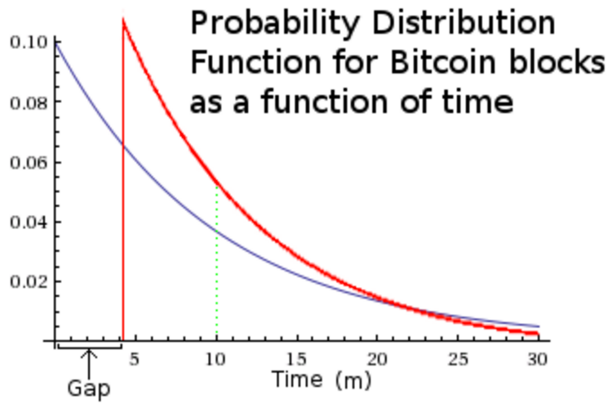

Figure 11: Illustration of a mining gap. The blue line shows the current P.D.F. of the time to next block. If the block reward by itself is too small to incentivize mining, rational miners will wait until enough transactions have accumulated before starting to mine. This will lead to a P.D.F. of a different shape (red line). Note that in either scenario the mean time to the next block is 10 minutes (green line)

strategies k that we didn't choose in round i, there is an algorithm (namely, EXP3, see [5] for description) that guarantees $c = 2\epsilon T + \#\text{ strategies} \cdot \ln(\#\text{ strategies})/\epsilon$.

So option one in our simulator is just to run EXP3 in earnest: whenever a miner uses some strategy k during game i, they learn their payoff and update their weights accordingly. Still, MWU converges faster, so it would be nice if we could learn how much payoff the miner would have received if they used strategy k during game i for all k, but this is computationally very expensive as it essentially requires us to rerun the entire game for all miners and strategies k (thereby becoming more expensive than just running the additional games to let EXP3 converge).

Instead, we make the following observation: even if this miner is not using strategy k during game i, maybe some other miner is - could we use that miner's payoff instead of recomputing exactly what payoff this miner would have received? The answer is of course we can, we just won't get a theoretical guarantee like if we used MWU in earnest. The payoff from different miner perspectives are of course different, but not wildly so. Specifically, the difference is that miner 1 is facing opponents $2, 3, \ldots$, whereas miner 2 faces opponents $1, 3, \ldots$. If miner 1 and miner 2 use different strategies in round i, then strategy k would yield slightly different rewards when used by each of them. With many small miners, this difference should be small, so we include this learning option as it seems to converge faster than EXP3, even though there is no theoretical guarantee. Specifically what we mean is the following: instead of learning the payoff that the miner would have received had they used strategy k during round i, they simply take the average payoffs of all miners that used strategy k during round i instead.

It is certainly possible that improvements to the learning aspect of the simulation are possible (and we encourage future work on this aspect once the simulator is open-source), but we note that the current implementations sufficed for the settings we studied.

C. PROOF OF THEOREM 5.1

Below is a complete proof of Theorem 5.1. Some quick notation: for an increasing function $f(\cdot)$, we'll denote by $f^{-1}(x) = \min\{y | f(y) \geq x\}$. If no such y exists, then we'll denote $f^{-1}(x) = +\infty$.

First, we make an extremely useful observation about when miners will receive payment for their blocks. Essentially, because miners only consider mining on $\textsc{Most}_{\mathrm{H}}$ or $\textsc{Most}_{\mathrm{H}-1}$, once a block is a predecessor of both such blocks, it is guaranteed to be in the eventual longest chain.

Observation 1. *As long as miners only consider mining on top of blocks* $\textsc{Most}_{\mathrm{H}}$ *or* $\textsc{Most}_{\mathrm{H}-1}$*, a miner receives eventual payment for mining a block if and only if the next block found chooses to continue her chain instead of undercutting.*

Proof. Because miners only consider chains $\textsc{Most}_{\mathrm{H}-1}$ or $\textsc{Most}_{\mathrm{H}}$, immediately after producing a new block B, B is in the longest chain. Either B goes on top of $\textsc{Most}_{\mathrm{H}-1}$, in which case it is in a chain of length H, which is the longest. Or it goes on top of $\textsc{Most}_{\mathrm{H}}$, which creates a new longest chain of length $\mathrm{H}+1$. Let H_{new} denote the new length of the longest chain (H if the miner undercut, and $\mathrm{H}+1$ if she continued).

Either the newly minted B is equal to $\textsc{Most}_{\mathrm{H}_{new}}$, or it isn't. If it isn't, then neither the next miner, nor any other miner in the future will ever mine on top of it, because there is a "better" chain of length H_{new} to mine on top of instead. If it is, then the next miner will either undercut or continue. If the next miner continues, then B is now equal to $\textsc{Most}_{\mathrm{H}_{new}}$ *and* the predecessor of $\textsc{Most}_{\mathrm{H}_{new}+1}$. This means that all future miners will continue a chain containing B, and therefore it will certainly be in the eventual longest chain.

If instead the next miner undercuts, then there will be a new chain of length H_{new} that leaves more available BTC, meaning that $\textsc{Most}_{\mathrm{H}_{new}}$ does *not* contain B as a predecessor. $\textsc{Most}_{\mathrm{H}_{new}-1}$ clearly does not contain B either, as B was mined on top

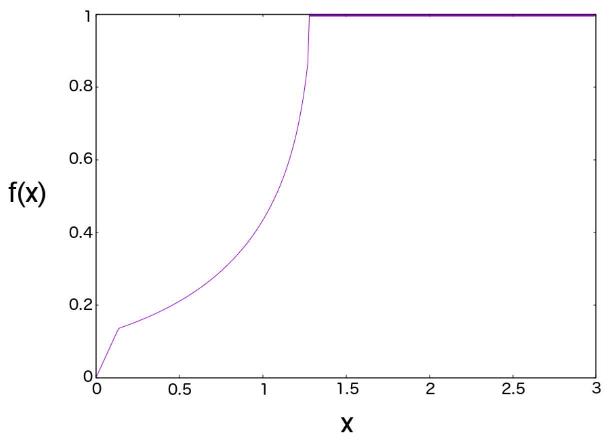

Figure 12: One example of the function that function forking miners might use that leads to an equilibrium. Recall, the function is $f(x) = x$ on the range $[0, y]$, the $-W_0(-ye^{x-2y})$ on $[y, 2y - \ln(y) - 1$, and 1 on $[2y - \ln(y) - 1, \infty)$.

of this chain. So B is contained in neither $\textsc{Most}_{\mathrm{H}_{new}}$ nor $\textsc{Most}_{\mathrm{H}_{new}-1}$, and therefore no future miners will ever consider a chain containing B.

In conclusion, whether or not a miner receives payment for block B depends entirely on whether or not the subsequent miner decides to mine on top of B or not. $\square$

We now want to figure out a best response for an individual non-atomic miner, conditioned on all other miners using $\textsc{FunctionFork}(f)$. So we need to figure out the probability that a miner will get undercut when authorizing B BTC in transactions, assuming that all other miners are using $\textsc{FunctionFork}(f)$. Note that as more and more new BTC of transactions arrive, other miners become less inclined to undercut. What we need to figure out is exactly how many new BTC of transactions need to arrive before the next miner switches from preferring to undercut to preferring to continue the longest chain.

Lemma C.1. *If a miner authorizes B BTC of transactions on $\textsc{Most}_i$ (of course, i will be in $\{\mathrm{H}-1, \mathrm{H}\}$), then other $\textsc{FunctionFork}(f)$ miners will try to undercut her until $\max\{0, f^{-1}(B) + B - \textsc{Rem}(\textsc{Most}_i)\}$ new BTC of transactions arrive ($\textsc{Rem}(\textsc{Most}_i)$ taken at the instant that the miner authorizes her block).*

Therefore, the expected BTC obtained by authorizing B BTC of transactions is $Be^{-\max\{0, f^{-1}(B)+B-\textsc{Rem}(\textsc{Most}_i)\}}$.

Proof. First, observe that because the miner chooses to build upon $\textsc{Most}_{\mathrm{H}-1}$ or $\textsc{Most}_{\mathrm{H}}$, then the chain containing their block is the new $\textsc{Most}_{\mathrm{H}}$, and that same chain minus their block is the new $\textsc{Most}_{\mathrm{H}-1}$. So the gap between the number of available BTC in $\textsc{Most}_{\mathrm{H}}$ versus $\textsc{Most}_{\mathrm{H}-1}$ ($\textsc{Gap}_{\mathrm{H}} = \textsc{Rem}(\textsc{Most}_{\mathrm{H}-1}) - \textsc{Rem}(\textsc{Most}_{\mathrm{H}})$) for the *next* miner is exactly B.

Now, immediately when the miner publishes her block, there are $\textsc{Rem}(\textsc{Most}_i)$ BTC of transactions available on $\textsc{Most}_{\mathrm{H}-1}$, and $\textsc{Rem}(\textsc{Most}_i) - B$ BTC of transactions available on $\textsc{Most}_{\mathrm{H}}$. So at this point, other miners would choose to undercut iff $f(\textsc{Rem}(\textsc{Most}_i) - B) < B$. As more new BTC of transactions arrive (call it x), the other miners would choose to undercut iff $f(\textsc{Rem}(\textsc{Most}_i) - B + x) < B$. As $f(\cdot)$ is increasing, we can look for the minimum x where this ceases hold, which is exactly when $\textsc{Rem}(\textsc{Most}_i) - B + x = f^{-1}(B)$, or $x = f^{-1}(B) + B - \textsc{Rem}(\textsc{Most}_i)$. $\square$

We now prove three corollaries of Lemma C.1 regarding what choices of B might possibly be optimal.

Corollary C.2. *If every other miner is playing $\textsc{FunctionFork}(f)$, then the optimal choice B^* of BTC to authorize when building upon chain $\textsc{Most}_i$ satisfies*

- $B^* \in argmax_{B\in[0,\textsc{Gap}_{\mathrm{H}}]}\{Be^{-\max\{0, f^{-1}(B)+B-\textsc{Rem}(\textsc{Most}_{\mathrm{H}-1})\}}\}$, *if* $i = \mathrm{H} - 1$.
- $B^* \in argmax_{B\in[0,\textsc{Rem}(\textsc{Most}_{\mathrm{H}})]}\{Be^{-\max\{0, f^{-1}(B)+B-\textsc{Rem}(\textsc{Most}_{\mathrm{H}-1})\}}\}$, *if* $i = \mathrm{H}$.

Proof. This is an immediate corollary of Lemma C.1, combined with the fact that a miner who chooses to undercut can authorize at most $\textsc{Gap}_{\mathrm{H}}$ BTC, while a miner who chooses to continue can authorize at most $\textsc{Rem}(\textsc{Most}_{\mathrm{H}})$. $\square$

Corollary C.3. *If $B_1 \geq B_2$, and $B_1 e^{-B_1 - f^{-1}(B_1)} \geq B_2 e^{-B_1 - f^{-1}(B_1)}$, then for all X, the expected reward from authorizing B_1 BTC in transactions is at least as large as the expected reward from authorizing B_2 BTC when $\textsc{Rem}(\textsc{Most}_{\mathrm{H}}) = X$.*

Proof. There are two cases to consider. First, maybe $X > B_1 + f^{-1}(B_1)$ (the miner guarantees that she is not undercut by authorizing B_1 BTC in transactions). In this case, because $B_1 \geq B_2$ and $f^{-1}(\cdot)$ is increasing, we clearly have $X > B_2 + f^{-1}(B_2)$ as well, meaning that the expected reward by authorizing B_1 BTC is exactly B_1, and that the expected reward by authorizing B_2 BTC is exactly B_2, by Lemma C.1. As $B_1 \geq B_2$, the reward from B_1 is at least as large.

In the second case, maybe $X \leq B_1 + f^{-1}(B_1)$ (the miner is undercut with positive probability by authorizing B_1 BTC in transactions). In this case, the reward from authorizing B_1 BTC is $B_1 e^{-B_1 - f^{-1}(B_1)+X}$, by Lemma C.1. Also by Lemma C.1, the reward from authorizing B_2 BTC is $B_2 e^{-\max\{0, B_2 + f^{-1}(B_2) - X\}} \leq B_2 e^{X - B_2 - f^{-1}(B_2)} = e^X \cdot B_2 e^{-B_2 - f^{-1}(B_2)}$. By hypothesis, this is upper bounded by $e^X B_1 e^{-B_1 - f^{-1}(B_1)}$, which is exactly the reward obtained by authorizing B_1 BTC. So authorizing B_1 BTC provides at least as much reward.

In both cases, we see that authorizing B_1 BTC is at least as good as B_2. □

Corollary C.4. *If $B_1 \geq B_2$, and $B_1 e^{-B_1 - f^{-1}(B_1)} \leq B_2 e^{-B_2 - f^{-1}(B_2)}$, then for all $X \leq B_2 + f^{-1}(B_2)$, the expected reward from authorizing B_2 BTC in transactions is at least as large as the expected reward from authorizing B_1 BTC when* $\text{REM}(\text{MOST}_\text{H}) = X$.

PROOF. By hypothesis, $X \leq B_2 + f^{-1}(B_2)$ (the miner is undercut with positive probability by authorizing B_2 BTC in transactions). Therefore, the expected reward from authorizing B_2 BTC is $B_2 e^{-B_2 - f^{-1}(B_2)+X}$. As $B_1 > B_2$ and $f^{-1}(\cdot)$ is increasing, we have $X \leq B_1 + f^{-1}(B_1)$ as well. This means that the expected reward from authorizing B_1 BTC is $B_1 e^{-B_1 - f^{-1}(B_1)+X}$. By hypothesis, this is less than the reward of authorizing B_2. □

We now recall quickly properties of $W_0(\cdot)$:

- The domain of $W_0(\cdot)$ is $[-1/e, \infty)$ and the range is $[-1, \infty)$.
- $W_0(\cdot)$ is increasing.
- $W_0(xe^x) = x$ for all $x \in [-1, \infty)$.

We will need to make use of some technical facts about $f(\cdot)$ (our specific choice from the statement of Theorem 5.1) that we first prove below.

Fact 1. *$f(x) \leq x$ everywhere.*

PROOF. Clearly, $f(x) \leq x$ on $[0, y]$. Also clearly, $f(x) \leq x$ on $[2y - \ln(y) - 1, \infty)$ iff $f(2y - \ln(y) - 1) \leq 2y - \ln(y) - 1$. So we just need to check the range $[y, 2y - \ln(y) - 1]$. The derivative of $W_0(x) = \frac{W_0(x)}{x(W_0(x)+1)}$. So the derivative of f on this range is (by the chain rule):

$$-\frac{W_0(-ye^{x-2y})}{-ye^{x-2y}(W_0(-ye^{x-2y})+1)} \cdot -ye^{x-2y}$$

$$= \frac{-W_0(-ye^{x-2y})}{1 + W_0(-ye^{x-2y})}$$

$$= \frac{f(x)}{1 - f(x)}$$

As $f(\cdot)$ is increasing and positive on $[y, 2y - \ln(y) - 1]$ (because of the form for $f'(x)$ we just derived above - not all positive, increasing $f(\cdot)$ have increasing derivatives), this means that $f'(\cdot)$ is also increasing and positive on $[y, 2y - \ln(y) - 1]$. As the derivative of x is constant (1), this means that if $f(x) > x$ anywhere on this interval, $f(2y - \ln(y) - 1) > 2y - \ln(y) - 1$ or $f(y) > y$. We can clearly see that $f(y) = -W_0(-ye^{-y}) = y$, and $f(2y - \ln(y) - 1) = -W_0(-ye^{2y - \ln(y) - 1}) = -W_0(-1/e) = 1$. So we can't have $f(y) > y$, and we have $f(2y - \ln(y) - 1) > 2y - \ln(y) - 1$ if and only if $2y - \ln(y) - 1 < 1$, which is the same as $2y - \ln(y) < 2$. As this is exactly the range of y we disallow, we see that we also can't have $f(2y - \ln(y) - 1) > 2y - \ln(y) - 1$ for any y we allow. Therefore, $f(x) \leq x$ everywhere. □

Fact 2. $Be^{-B - f^{-1}(B)} =$

- $Be^{-2B}, B \in [0, y]$.
- $ce^{-2c}, B \in [y, 1]$.
- $0, B > 1$.

PROOF. We first observe that $f^{-1}(B) = B$ for all $B \in [0, y]$, which immediately proves the first bullet. We next observe that $f^{-1}(B) = +\infty$ for all $B > 1$, which immediately proves the last bullet.

For the middle bullet, observe that:

$$-W_0(-ye^{(2y + \ln(z/y) - y) - 2y}) = -W_0(-ye^{\ln(z/y) - z})$$

$$= -W_0(-ze^{-z}) = z.$$

Note that the last equality is due to the fact that $W_0(\cdot)$ is the inverse of xe^x. This proves that $f^{-1}(B) = 2y + \ln(B/y) - B$ when $B \in [y, 1]$ and completes the middle bullet. □

Corollary C.5. *If $y \in (0, 1/2]$, then $Be^{-B-f^{-1}(B)}$ is strictly increasing on $[0, y]$ and constant on $[y, 1]$.*

PROOF. $Be^{-B-f^{-1}(B)}$ is clearly constant on $[y, 1]$, so we just need to confirm that it's strictly increasing on $[0, y]$. The derivative of Be^{-2B} is $(1-2B)e^{-2B}$, which is strictly positive on $[0, 1/2]$ (and therefore on $[0, y]$ for all $y \leq 1/2$), as desired. □

Proof of Theorem 5.1: We want to invoke Corollary C.3 combined with Corollary C.5. Together, these immediately say that for any $1 \geq B_1 > B_2 \geq 0$, it is at least as good to authorize B_1 BTC as B_2. As authorizing $B > 1$ BTC always results in expected reward of 0, this immediately implies by Corollary C.2 that for any b, $\min\{1, b\} \in \text{argmax}_{B\in[0,b]}\{Be^{-\max\{0,B+f^{-1}(B)-\text{REM}(\text{MOST}_i)\}}\}$.

Now, we also want to invoke Corollary C.4 to show that there may exist other maximizers as well if $\text{REM}(\text{MOST}_i) \in [y, 2y - \ln(y) - 1]$. Note that $f(\cdot)$ is strictly increasing in this range, meaning that $f^{-1}(f(\text{REM}(\text{MOST}_i))) = \text{REM}(\text{MOST}_i)$. Therefore, we see that $\text{REM}(\text{MOST}_i) \leq f(\text{REM}(\text{MOST}_i)) + f^{-1}(f(\text{REM}(\text{MOST}_i)))$ (the miner will be undercut with positive probability when authorizing $f(\text{REM}(\text{MOST}_i))$ BTC) on this entire range. Together with Corollary C.5, this means that the hypotheses of Corollary C.4 are satisfied taking $B_2 = f(\text{REM}(\text{MOST}_i))$ and any $B_1 \geq B_2$. Combined with the reasoning above, this means that when $\text{REM}(\text{MOST}_i) \in [y, 2y - \ln(y) - 1]$ and $b \geq f(\text{REM}(\text{MOST}_i))$, we also have $f(\text{REM}(\text{MOST}_i)) \in \text{argmax}_{B\in[0,b]}\{Be^{-\max\{0,B+f^{-1}(B)-\text{REM}(\text{MOST}_i)\}}\}$.

Therefore, when $b = \text{REM}(\text{MOST}_\text{H})$, we recover that $f(\text{REM}(\text{MOST}_\text{H}))$ is an optimal choice of BTC to authorize when continuing. When $b = \text{GAP}_\text{H}$, we recover that $\min\{1, \text{GAP}_\text{H}, f(\text{REM}(\text{MOST}_{\text{H}-1}))\} = \min\{\text{GAP}_\text{H}, f(\text{REM}(\text{MOST}_{\text{H}-1}))\}$ is an optimal choice of BTC to authorize when undercutting.

So FUNCTIONFORK(f) correctly chooses how many BTC to authorize when continuing and when undercutting, we just need to check that it also chooses when to undercut and when to continue. If $\text{GAP}_\text{H} > f(\text{REM}(\text{MOST}_\text{H}))$, then $\min\{\text{GAP}_\text{H}, f(\text{REM}(\text{MOST}_{\text{H}-1}))\} \geq f(\text{REM}(\text{MOST}_\text{H}))$ as well, and we can invoke Corollary C.3 with $B_1 = \min\{\text{GAP}_\text{H}, f(\text{REM}(\text{MOST}_{\text{H}-1}))\}$ and $B_2 = f(\text{REM}(\text{MOST}_\text{H}))$. By the argument above, because $1 \geq B_1 \geq B_2$, the hypotheses of Corollary C.3 are satisfied, and the expected reward is at least as high when authorizing B_1 as B_2, so undercutting is at least as good as continuing. Similarly, if $f(\text{REM}(\text{MOST}_\text{H})) \geq \text{GAP}_\text{H}$, then $f(\text{REM}(\text{MOST}_\text{H})) \geq \min\{\text{GAP}_\text{H}, f(\text{REM}(\text{MOST}_{\text{H}-1}))\}$. So we may again invoke Corollary C.3, this time with $B_1 = f(\text{REM}(\text{MOST}_\text{H}))$ and $B_2 = \min\{\text{GAP}_\text{H}, f(\text{REM}(\text{MOST}_{\text{H}-1}))\}$.

So now we have shown that the FUNCTIONFORK(f) correctly chooses how many BTC to authorize when continuing and when undercutting, and also chooses correctly whether to continue or undercut. So it is an equilibrium.

The last part we need to reason about is the connection to random walks. Observe that the number of transaction fees grows continuously at a rate of 1 per unit. Every time a block is found, it drops by at most 1. So definitely the backlogged transactions will be at least as bad as a random walk that drops by exactly 1 (because it will only drop further).

Lemma C.7 below proves that with constant probability, the number of blocks found in a time interval of length $n + \sqrt{n}$ is at most n. When this occurs, there is a backlog of at least $\sqrt{n}$ transactions at time $n + \sqrt{n}$. Therefore, the expected backlog is at least $\Theta(\sqrt{n})$ (in fact, it is exactly $\Theta(\sqrt{n})$). During this time, new transactions take $\Theta(\sqrt{n})$ time steps before they are included in a block. □

Before proving Lemma C.7, we recall the Berry-Esseen theorem:

Theorem C.6 (Berry-Esseen)**.** *Let $X_1, \ldots, X_n$ be i.i.d. random variables with mean 0, $\mathbb{E}[X_i^2] = \sigma^2$, $\mathbb{E}[X_i^3] = \rho$. Then for all x:*

$$Pr[\frac{\sum_i X_i}{\sigma\sqrt{n}} \geq x] - \Phi(x) = O(\frac{\rho}{\sigma^3\sqrt{n}}),$$

Where $\Phi(x)$ denotes the probability that a Gaussian random variable with mean 0 and standard deviation 1 exceeds x.

Lemma C.7. *Define X_i to be an exponential random variable with mean 1. Then:*

$$Pr[\sum_{i=1}^{n} X_i > n + \sqrt{n}] = \Theta(1).$$

In particular, this implies that probability that fewer than n blocks are found in $n + \sqrt{n}$ time steps is $\Theta(1)$.

PROOF. Define $Y_i = X_i - 1$. Then the Y_i are i.i.d. random variables with mean 0, $E[Y_i^2] = \sigma < 2$, and $E[Y_i^3] = \rho < 6$. Plugging into Berry-Esseen (stated below), we get:

$$Pr[\sum_{i=1}^{n} Y_i > \sqrt{n}] = Pr[\frac{\sum_{i=1}^{n} Y_i}{\sigma\sqrt{n}} > \frac{1}{\sigma}] \geq \Phi(\frac{1}{\sigma}) - O(\frac{1}{\sqrt{n}}).$$

As σ is a constant independent of n, $\Phi(\sigma)$ is also independent of n, so $\Phi(\frac{1}{\sigma}) - O(\frac{1}{\sqrt{n}}) = \Theta(1)$, as desired. □

D. WHEN DEFAULT MINING IS AN EQUILIBRIUM FOR NON-ATOMIC MINERS

In the absence of latency, default mining is an equilibrium for non-atomic miners regardless of the reward model, and the reasoning is simple: if you do anything except extend the unique longest chain, your block will be orphaned and you will receive reward zero. If you wait to publish your block, you risk losing the option to publish it without being orphaned. All other miners ignore the transactions included in your block when deciding where to extend, so you may as well include as many transactions as possible.

In the presence of latency, forks will naturally occur, so PettyCompliant outperforms DefaultCompliant in the transaction fees model. In the fixed reward model, DefaultCompliant remains an equilibrium under quite general models of latency (still assuming non-atomic miners). Consider, for instance, any model of latency with the following property. Whenever miner m finds a block, and miner m' finds a block at a later time, we have $B_m \subseteq B_{m'}$, where B_m denotes the set of blocks that miner m had heard of when they found their block. In other words, by the time miner m' solves their block, they have become aware of at least every block that m was aware of when they solved their block earlier (but perhaps not m's block, nor any blocks that m was not herself aware of).

It is easy to see that simple latency models (such as all announcements being grouped into chunks of λ seconds) have this property, as well as much more general latency models. It is also easy to see that in the transaction fees model, the simple latency model where announcements are grouped into chunks of λ seconds is rich enough so that DefaultCompliant is strictly outperformed by PettyCompliant and therefore not an equilibrium.

Proposition D.1. *When miners are non-atomic, even in the presence of any latency of the form described above, it is an equilibrium for every miner to use* DefaultCompliant.

Proof. The proof is actually very straight-forward: assuming that all other miners are DefaultCompliant, mining anywhere except on top of a longest chain guarantees that your block will be orphaned and you will receive a reward of zero (because our latency assumptions guarantee that the next miner and all future miners will have heard about the blocks you chose to undercut before yours, and they are all DefaultCompliant). So the only choices are how to tie-break among multiple longest chains. But this choice neither affects your rewards (they are fixed!), nor the likelihood that your block will be chosen by the next miner (as this depends only on how quickly they hear about your block and not on its contents). So tie-breaking in favor of the earliest chain is at least as good as any other tie-breaking rule.

Finally, it is also easy to see that publishing as soon as possible is optimal, as this maximizes the likelihood that your block is chosen to be extended. □

The point of Proposition D.1 is again just to contrast the difference between transaction fees and fixed rewards. In the non-atomic regime, even in quite general latency models, DefaultCompliant mining is an equilibrium in the fixed-reward model. The proof is simple and matches exactly our intuition for why DefaultCompliant should make sense. But in the transaction fees model, whenever there exists a possibility for forks, DefaultCompliant is strictly outperformed by PettyCompliant, and the space of equilibria is therefore much more complex. In particular, it would be interesting for future work to identify an equilibrium for non-atomic miners in the transaction fees model in any non-trivial latency model.

E. SELFISH MINING

E.1 Classic Selfish Mining with transaction fees

Here we provide details on how to analyze selfish mining in the transaction fee regime. Recall that Eyal and Sirer [9] have already computed p_s for all s, the probability that the block chain is in state s. Below we compute f_s for all states s, the probability that a transaction winds up with the selfish miner conditioned on that transaction arriving while the blockchain is in state s.

Computing f_0: Let's consider the possible outcomes when a transaction arrives in state 0:

- If a default miner mines the next block, it will contain this transaction, and this block will definitely be in the eventual longest chain. This happens with probability $(1-\alpha)$.
- Alternatively, the selfish miner could find the next block. If the selfish miner finds the next block, they will include the transaction in their block, but they keep this block private after they find it. This happens with probability α, but this block is *not* guaranteed to make it into the eventual longest chain, yet.
- From here, maybe the selfish miner finds the next block as well. This happens with probability α. Once this happens, both blocks are guaranteed to be in the eventual longest chain. So this event contributes a probability α^2 that the transaction winds up in the selfish miner's block.
- Alternatively, a default miner might find the next block, which triggers a race. This happens with probability $(1-\alpha)$. Both racing blocks contain the transaction being considered, so whoever wins the race receives the corresponding transaction fees. The selfish miner wins the race with probability $\alpha + \gamma(1-\alpha)$, so this event contributes $\alpha(1-\alpha)(\alpha+\gamma(1-\alpha))$ in total.

Therefore, we see that:

$$f_0 = \alpha^2 + \alpha(1-\alpha)\left(\alpha + \gamma(1-\alpha)\right) \quad (4)$$

Computing $f_{0'}$: If a new transaction is announced in state $0'$, then the next block found is certainly contained in the eventual longest chain because it is always announced and every miner chooses to mine on top of it. So this transaction is won by whichever miner finds the next block, which is the selfish miner with probability α. Therefore:

$$f_{0'} = \alpha \tag{5}$$

f_1**:** Consider now a transaction announced in state 1, and where it might wind up:

- If the selfish miner finds the next block, they will have a private chain of length 2, in which case both blocks are guaranteed to make it into the final block chain. Therefore, this transaction will certainly wind up in a block mined by the selfish miner. This happens with probability α.

- Alternatively, the rest of the network might find the next block. This happens with probability $(1-\alpha)$. But we don't yet know whether or not this block will make it in the eventual longest chain because this triggers the "race," and puts us in state $0'$. Note though that the racing selfish block does *not* contain this transaction that arrived once we were already in state 1. Therefore, even if the selfish miner wins the race, but because a default miner chose their block, the selfish miner will not get this transaction. So the only way for the selfish miner to win this transaction is to find the block that ends the race. This happens with probability α.

$$f_1 = \alpha + (1-\alpha)\alpha = \alpha(2-\alpha). \tag{6}$$

Computing f_i**:** Finally, consider a transaction arriving to the system in state i, $i > 1$. In these states, it is easier to consider what must happen in order for the transaction to **not** end up in a block the selfish miner owns. For the transaction to wind up in a default miner's block, it needs to be the case that the selfish miner releases their entire private chain *before* mining a new block (which would contain this transaction). This is because any blocks found by default miners before this trigger are all orphaned. For a release to be triggered, a default miner must find each of the next $i-1$ blocks, which happens with probability $(1-\alpha)^{i-1}$.

If this happens, we still don't know where this transaction winds up, because each of the $i-1$ blocks found will be orphaned. But we have now returned to state 0, and the remainder of the analysis concludes as if the transaction had been announced during state 0. So the probability that a default miner winds up with a transaction arriving in state i is $(1-\alpha)^{i-1}(1-f_0)$, and therefore:

$$f_i = 1 - ((1-\alpha)^{i-1}(1-f_0)) \tag{7}$$

Summing everything together, we get the following:

Theorem E.1. *If all other miners remain* DefaultCompliant*, a selfish miner in the transaction fees model with an* $\alpha \in (0, .5)$ *fraction of the mining power and racing parameter* $\gamma \in [0,1]$ *achieves reward* Reward(α,γ) *with:*

$$\textsc{Reward}(\alpha,\gamma) =$$

$$\frac{5\alpha^2 - 12\alpha^3 + 9\alpha^4 - 2\alpha^5 + \gamma(\alpha - 4\alpha^2 + 6\alpha^3 - 5\alpha^4 + 2\alpha^5)}{2\alpha^3 - 4\alpha^2 + 1}$$

Proof. The only remaining part of the proof is summing $p_0 f_0 + p_{0'} f_{0'} + p_1 f_1 + \sum_{i>1} p_i f_i$.

$$
\begin{aligned}
p_0 f_0 &= \frac{1-2\alpha}{2\alpha^3-4\alpha^2+1}\cdot\left(\alpha^2+\alpha(1-\alpha)(\alpha+(1-\alpha)\gamma)\right)\\
&= \frac{2\alpha^2-5\alpha^3+2\alpha^4+\alpha\gamma-4\alpha^2\gamma+5\alpha^3\gamma-2\alpha^4\gamma}{2\alpha^3-4\alpha^2+1}\\
p_{0'}f_{0'} &= \frac{(1-\alpha)(\alpha-2\alpha^2)}{2\alpha^3-4\alpha^2+1}\cdot\alpha\\
&= \frac{\alpha^2-3\alpha^3+2\alpha^4}{2\alpha^3-4\alpha^2+1}\\
p_1 f_1 &= \frac{\alpha-2\alpha^2}{2\alpha^3-4\alpha^2+1}\cdot\alpha(2-\alpha)\\
&= \frac{2\alpha^2-5\alpha^3+2\alpha^4}{2\alpha^3-4\alpha^2+1}\\
p_i f_i &= (\frac{\alpha}{1-\alpha})^{i-1}\frac{\alpha-2\alpha^2}{2\alpha^3-4\alpha^2+1}-\alpha^{i-1}(1-f_0)\frac{\alpha-2\alpha^2}{2\alpha^3-4\alpha^2+1}\\
\sum_{i>1}(\frac{\alpha}{1-\alpha})^{i-1} &= \frac{\alpha}{1-2\alpha}\\
\Rightarrow \sum_{i>1} p_i f_i &= \frac{\alpha^2}{2\alpha^3-4\alpha^2+1}-\sum_{i>1}\alpha^{i-1}(1-f_0)\frac{\alpha-2\alpha^2}{2\alpha^3-4\alpha^2+1}\\
\sum_{i>1}\alpha^{i-1} &= \frac{\alpha}{1-\alpha}\\
\Rightarrow \sum_{i>1} p_i f_i &= \frac{\alpha^2}{2\alpha^3-4\alpha^2+1}-\frac{\alpha(1-\alpha+\alpha(1-\alpha)^2(1-\gamma))(\alpha-2\alpha^2)}{(1-\alpha)2\alpha^3-4\alpha^2+1}\\
&= \frac{\alpha^2-\alpha(1+\alpha(1-\alpha)(1-\gamma))(\alpha-2\alpha^2)}{2\alpha^3-4\alpha^2+1}.\\
&= \frac{2\alpha^3-\alpha^2(\alpha-2\alpha^2-\alpha^2+2\alpha^3-\gamma\alpha+2\gamma\alpha^2+\gamma\alpha^2-2\gamma\alpha^3)}{2\alpha^3-4\alpha^2+1}\\
&= \frac{\alpha^3+3\alpha^4-2\alpha^5+\gamma\alpha^3-3\gamma\alpha^4+2\gamma\alpha^5}{2\alpha^3-4\alpha^2+1}
\end{aligned}
$$

The proof concludes by just summing the four terms. □

E.2 Improved Selfish Mining with a cutoff

In this section, we complete our analysis of our improved selfish mining with a cutoff. In order to keep the analysis of this strategy tractable, we choose to slightly tweak our analysis (but our theory-matches-simulation plot in Figure 6.2 shows that this tweak is essentially irrelevant). The only tweak we make is that right after the selfish miner releases a chain of length two simultaneously, they immediately publish the next block (if they find it), and then return to selfish mining. In the language of Eyal and Sirer, this is like adding an additional state $0''$ where the selfish miner honestly mines. No matter who finds a block in this state, the next state is 0. The only transition into this state is when the honest portion of the network finds a block when the selfish miner has a lead of 2 13 shows an updated Markov chain with state $0''$.

Again, note that this modification is *just for analysis*. The selfish mining with cutoff that is implemented in our simulator is as described in the body.

In order to calculate the selfish miner's expected revenue, we must again calculate the probability of the system being in any given state, and the chance that a transaction arriving to the system while in one of these states eventually ends up in a block mined by the selfish miner. From looking at the state transitions in Figure 13, we can derive the following formulas relating the probabilities of being in each state:

$$p_i\alpha = p_{i+1}(1-\alpha) \tag{8}$$

$$\implies p_i = (\frac{\alpha}{1-\alpha})^{i-1}p_1 \tag{9}$$

$$p_{0''} = (1-\alpha)p_2 = \alpha p_1 \tag{10}$$

$$p_{0'} = (1-\alpha)p_1 \tag{11}$$

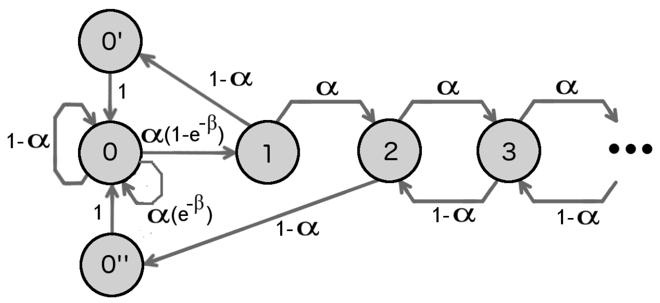

Figure 13: State machine for selfish mining with a cutoff, introducing state $0''$.

$$p_0 = \frac{p_1}{\alpha(1-e^{-\beta})} \tag{12}$$

We also know that the system is guaranteed to be in some state, which means the following.

$$p_0 + p_{0'} + p_{0''} + \sum_{i=1}^{i=\infty} p_i = 1 \tag{13}$$

Which together imply that

$$p_1 = \frac{\alpha(2\alpha-1)(e^{\beta}-1)}{3\alpha^2(e^{\beta}-1)+2\alpha-e^{\beta}} \tag{14}$$

With equations 9-12, this gives expressions for the probabilities of all the possible states the system could be in. Now we need to compute the probability that a transaction that arrives when the system is in state s winds up with the selfish miner.

Unfortunately, this is not a clean approach: because in state 0 the selfish miner will sometimes publish and sometimes hide their block, depending on how much time has passed since the last block was found, we need actually to introduce a continuum of states for each amount of time x for the size of the block that is building during state 0.

So let's define a new variable, $p_0(x)$ which denotes the probability that the system is in state 0 and x units of time have passed since the system entered state 0. Because we introduced this new state $0''$, whenever we enter state 0, the initial block is empty. Therefore, the probability that we wind up in state 0 with a block of size at least x is p_0e^{-x}, and we have:

$$p_0(x) = p_0e^{-x}dx \tag{15}$$

We must now calculate the associated f_s (probability that a transaction winds up with the selfish miner conditioned on arriving during state s) in order to calculate the expected fraction of the rewards claimed by the selfish miner.

Computing $f_0(x)$. If a new transaction arrives in state 0, let's look at where this transaction might wind up. Note that this depends on how long it's been (x) since the last block was found.

- If the next block is found by the honest miner, then this transaction will certainly wind up with the honest miners. This happens with probability $1-\alpha$.
- If $x \geq \beta$, and the next block is found by the selfish miner, then it certainly winds up with the selfish miner. This happens with probability α.
- If $x < \beta$, and the next block is found by the selfish miner after time $\beta - x$ as passed, then it certainly winds up with the selfish miner. This happens with probability $\alpha e^{-\beta+x}$.
- If $x < \beta$, and the next block is found by the selfish miner within $\beta - x$ time, then this transaction isn't determined yet because the selfish miner chooses to hide that block. But this happens with probability $\alpha(1-e^{-\beta+x})$.
- If *both* of the next two blocks are found by the selfish miner, than this transaction is contained in a block of the selfish miner that will certainly be included in the eventual longest chain. This happens with probability $\alpha^2(1-e^{-\beta+x})$.
- If the next block is found by the selfish miner, followed by a block by the honest miner, then a race is triggered. This transaction is contained in the two racing blocks, so whoever wins the race gets this transaction. The race occurs with probability $\alpha(1-e^{-\beta+x})(1-\alpha)$, and the selfish miner wins the race with probability $\alpha + (1-\alpha)\gamma$.

So in total, we see that $f_0(x) = \alpha$, when $x \geq \beta$, and $f_0(x) = \alpha e^{-\beta+x} + \alpha^2(1-e^{-\beta+x}) + \alpha(1-\alpha)(1-e^{-\beta+x})(\alpha+(1-\alpha)\gamma)$ if $x \leq \beta$.

Computing $f_{0'}$. If a new transaction arrives when there are two chains competing of the same length, then the next block found is certainly contained in the eventual longest chain (because both miners choose to mine on top of it). So if the next block is found by the selfish miner, this transaction is won by him. Otherwise, it's won by the honest miner. So we have $f_{0'} = \alpha$.

Computing $f_{0''}$. If a new transaction arrives during the state $0''$, the next block found is certainly contained in the eventual longest chain again. So we again have $f_{0''} = \alpha$.

Computing f_1. If a new transaction arrives when the sefish miner has a private chain of length 1, let's consider where the transaction might wind up:

- If the next block is found by the selfish miner, then this transaction is contained in a block of the selfish miner that will certainly be included in the eventual longest chain. This happens with probability α.
- If the next block is found by the honest miner, then this triggers a release of the private block and a race. **But**, the racing selfish block does *not* contain this transaction, whereas the racing honest block does. So if the racing honest block wins, the honest miner gets this transaction. If the racing selfish block wins, whoever finds the block that ends the race gets this transaction. So the selfish miner gets the transaction in this case only if he finds the block that ends the race. This happens with probability $(1-\alpha)\alpha$.

So we see that $f_1 = \alpha + (1-\alpha)\alpha = \alpha(2-\alpha)$.

Computing f_i, $i > 1$. If a new transaction arrives when the selfish miner has a private chain of length $i > 1$, let's again consider where this transaction might wind up:

- If the next block is found by the selfish miner, then this transaction is contained in a block of the selfish miner that will certainly be included in the eventual longest chain. This happens with probability α.
- If the next $i-1$ blocks are *all* found by the honest miner, then this triggers a release of the private chain, and all those blocks found by the honest miner are immediately ignored. At this point, the transaction has still not been included in any block, so it is as if the transaction arrived in state $0''$. So the selfish miner gets this transaction with probability $f_{0''}$ in this case.
- If *any* of the next $i-1$ blocks are found by the selfish miner, then this block is certainly included in the eventual longest chain, because it is found when the selfish miner has a lead of at least two.

So we see that the only way the selfish miner might possibly lose the transaction is if each of the next $i-1$ blocks are found by the honest miner, and even in this case the selfish miner still wins the transaction with probability $f_{0''} = \alpha$. So the honest miner only wins this transaction with probability $(1-\alpha)^{i-1}(1-\alpha)$, and we have $f_i = 1-(1-\alpha)^i$.

Now, we just have to sum/integrate over all states and success probabilities to compute the fraction of transactions that go to the selfish miner.

$$f_{0'}p_{0'} = \alpha(1-\alpha)p_1.$$

$$f_{0''}p_{0''} = \alpha^2 p_1.$$

$$f_1 p_1 = \alpha(2-\alpha)p_1.$$

$$f_i p_i = \frac{(1-(1-\alpha)^i)\alpha^{i-1}p_1}{(1-\alpha)^{i-1}},\ i > 1.$$

$$f_0(x)p_0(x) = \frac{p_1 e^{-x}dx}{1-e^{-\beta}},\ x \geq \beta.$$

$$f_0(x)p_0(x) = \frac{p_1 e^{-x}dx(e^{-\beta+x} + \alpha(1-e^{-\beta+x}) + (1-\alpha)(1-e^{-\beta+x})(\alpha+(1-\alpha)\gamma))}{1-e^{-\beta}},\ x \leq \beta.$$

$$\sum_{i>1} \frac{\alpha^{i-1}}{(1-\alpha)^{i-1}} = \frac{\alpha}{1-2\alpha}.$$

$$\sum_{i>1} \alpha^{i-1} = \frac{\alpha}{1-\alpha}.$$

$$\Rightarrow \sum_{i>1} f_i p_i = p_1 \left(\sum_{i>1} \frac{\alpha^{i-1}}{(1-\alpha)^{i-1}} - (1-\alpha)\sum_{i>1}\alpha^{i-1} \right) = \left(\frac{\alpha}{1-2\alpha} - \alpha \right) p_1 = \frac{2\alpha^2 p_1}{1-2\alpha}.$$

$$\int_{x\geq\beta} f_0(x)p_0(x) = \frac{p_1}{1-e^{-\beta}}\int_{x\geq\beta} e^{-x}dx = \frac{e^{-\beta}p_1}{1-e^{-\beta}}.$$

$$\int_{x=0}^{\beta} f_0(x)p_0(x)$$
$$= \int_{x=0}^{\beta} \frac{p_1}{1-e^{-\beta}}\left(\left(e^{-\beta}-\alpha e^{-\beta}-(1-\alpha)(\alpha+(1-\alpha)\gamma)e^{-\beta}\right)+\left(\alpha e^{-x}+(1-\alpha)(\alpha+(1-\alpha)\gamma)e^{-x}\right)\right)dx.$$

$$= \frac{p_1\beta e^{-\beta}(1-\alpha-(1-\alpha)(\alpha+(1-\alpha)\gamma))}{1-e^{-\beta}} + \frac{p_1(1-e^{-\beta})(\alpha+(1-\alpha)(\alpha+(1-\alpha)\gamma))}{1-e^{-\beta}}$$
$$= \frac{p_1\left(\beta e^{-\beta}(1-\alpha)(1-\alpha-(1-\alpha)\gamma)+(1-e^{-\beta})(\alpha+(1-\alpha)(\alpha+(1-\alpha)\gamma))\right)}{1-e^{-\beta}}.$$

Summing everything together, we then get:

$$\int_0^\infty p_0(x)f_0(x) + \sum_{i>1} p_i f_i + p_{0'}f_{0'} + p_{0''}f_{0''} + p_1 f_1$$

$$= \left(\frac{\beta e^{-\beta}(1-\alpha)(1-\alpha-(1-\alpha)\gamma)}{1-e^{-\beta}} + \alpha + (1-\alpha)(\alpha+(1-\alpha)\gamma) + \frac{e^{-\beta}}{1-e^{-\beta}} + \frac{2\alpha^2}{1-2\alpha} + 3\alpha - \alpha^2\right)p_1.$$

This can be further simplified to yield the bound provided in the paper.

$$\left(\frac{\beta e^{-\beta}(1-\alpha)(1-\alpha-(1-\alpha)\gamma)}{1-e^{-\beta}} + \alpha + (1-\alpha)(\alpha+(1-\alpha)\gamma) + \frac{e^{-\beta}}{1-e^{-\beta}} + \frac{2\alpha^2}{1-2\alpha} + 3\alpha - \alpha^2\right)p_1.$$
$$= \frac{1+\beta(1-\alpha)^2(1-\gamma)}{e^{\beta}-1} + 4\alpha + (1-\alpha)(\alpha+(1-\alpha)\gamma) + \frac{2\alpha^2}{1-2\alpha} - \alpha^2.$$