Agent-Based Simulations of Blockchain protocols illustrated via Kadena's *Chainweb*

Tarun Chitra[1], Monica Quaintance[2], Stuart Haber[3], and Will Martino[4]

***Abstract*—While many distributed consensus protocols provide robust liveness and consistency guarantees under the presence of malicious actors, quantitative estimates of how economic incentives affect security are few and far between. In this paper, we describe a system for simulating how adversarial agents, both economically rational and Byzantine, interact with a blockchain protocol. This system provides statistical estimates for the economic difficulty of an attack and how the presence of certain actors influences protocol-level statistics, such as the expected time to regain liveness. This simulation system is influenced by the design of algorithmic trading and reinforcement learning systems that use explicit modeling of an agent's reward mechanism to evaluate and optimize a fully autonomous agent. We implement and apply this simulation framework to Kadena's Chainweb, a parallelized Proof-of-Work system, that contains complexity in how miner incentive compliance affects security and censorship resistance. We provide the first formal description of Chainweb that is in the literature and use this formal description to motivate our simulation design. Our simulation results include a phase transition in block height growth rate as a function of shard connectivity and empirical evidence that censorship in Chainweb is too costly for rational miners to engage in. We conclude with an outlook on how simulation can guide and optimize protocol development in a variety of contexts, including Proof-of-Stake parameter optimization and peer-to-peer networking design.**

I. Introduction

Blockchain systems provide security via the economic disincentivization of malicious behaviors, such as double-spending or long-range attacks. Formally, this security is established by proving that the system achieves liveness, consistency, and persistence under suitable networking conditions (e.g. partial synchrony or asynchrony) and under the assumption that honest and rational agents are in the majority [1], [2], [3]. Many of these results give precise quantitative conditions that prescribe when a protocol will fail as a function of parameters, such as the difficulty adjustment period [1] or epoch length [2]. However, it becomes exceedingly difficult to provide similarly strong results when dealing with sharded or parallelized blockchains. There is a marked loss in quantitative strength in the existing literature of results, as accommodating for communication complexity and resource (e.g. energy, space, or stake) redistribution leads to significantly worse liveness performance [4], [5] and weaker tolerances for Byzantine actors. Mathematically, the main reason for this loss in statistical power comes from an inability to apply non-trivial concentration inequalities to all chains simultaneously, due to residual correlations that stem from cross-shard transactions and the overhead of repeated committee selection. Moreover, the increased communication cost for cross-shard or cross-chain transactions is only described asymptotically, which makes it hard to tune practical peer-to-peer networking algorithms for cross-shard transactions. Finally, as one increases the number of shards, there is often a dramatic increase in the number of parameters, such as timeouts and resource limits per shard, that a protocol designer must choose. The choice of these parameters can dramatically impact real-world performance and security of a sharded or parallelized blockchain. This paper will introduce agent-based simulation, which is used in a variety of other fields, as a way for protocol designers and practitioners to overcome some of these problems.

Agent-based simulation is used by practitioners and researchers in algorithmic trading [6], artificial intelligence [7], autonomous vehicles [8], cybersecurity [9], economics [10], energy allocation [11], and by the US Commodities and Futures Trading Commission to detect fraudulent market activity [12]. These systems define a set of agents $A_1, \ldots, A_n$ that each have a state space $\mathcal{S}_i$ and interact with each other via a prescribed set of actions $\mathcal{A}$. Each agent receives updates to their state, caused either by the choice of actions of other agents or due to exogenous signals (e.g. external market prices), and then computes a *policy function* π_i that selects an action for an agent to take. After running a number of simulations of agents interacting under

[1]T. Chitra is at Gauntlet Networks, Inc.
tarun@gauntlet.network
[2]M. Quaintance is at Kadena LLC
monica@kadena.io
[3]S. Haber is Stuart Haber Crypto LLC
stuart.haber@acm.org
[3]W. Martino is at Kadena LLC
will@kadena.io

different ensembles of initial conditions, exogenous data, and policies, a practitioner usually computes statistical averages and/or estimators of game theoretic quantities [13] to provide estimates of 'macroscopic' quantities that are emergent properties of the agent interaction. The statistical sampling of these ensembles is usually implemented via Monte Carlo (MC) and Markov chain Monte Carlo (MCMC) methods, while the designing of policies, state spaces, and action spaces is performed via analytic modeling and more recently, deep reinforcement learning [7].

While these techniques have been successfully deployed in practice within other financial disciplines, the use of agent-based simulations for blockchain development has been scant. As far as the authors can tell, agent-based blockchain simulation has focused on refining selfish mining rewards [14], reducing the energy usage of Proof-of-Work systems [15], and on disproving the claims of various block-free ledgers (e.g. IOTA [16]). However, agent-based simulation can be used as a production tool to help protocol designers optimize parameters, estimate latency and bandwidth usage, and to estimate the true cost of security for Proof-of-Stake systems. Moreover, the pervasive usage of agent-based simulations in production environments within algorithmic trading intimates that the technique is useful for monitoring and estimating risk within live blockchain systems.

We aim to illustrate the versatility of agent-based simulation via an analysis of Chainweb, a parallelized Proof-of-Work protocol that has eluded a closed-form security proof due to explicit correlation between chains that makes it difficult to use standard probabilistic tools. Our simulations, which are designed to handle arbitrary distributed consensus protocols, will show that we can get high-fidelity estimates for the latency-security trade-off in Chainweb and an estimate for how much miners lose when they try to censor particular chains.

Finally, we note that estimating the precise cost of security is a difficult challenge that involves a number of variables, some of which are exogenous to the underlying chain like rental markets [17], arbitrage opportunities on centralized and decentralized exchanges, and the difficulty of estimating the market impact of attacks. The simulations in this paper illustrate how *endogenous* design choices, such as difficulty adjustment periods or shard correlations, affect practical protocol performance. Our final section will conclude with a description of how simulations can interact with exogenous data (akin to trading strategies) and how we can use simulation to model the effects of volatile exchange prices and derivatives markets on chain security. These uses are increasingly important as the security of Proof-of-Stake protocols is inseparably tied to exogenous data and judicious parameter selection (e.g. slashing rates in live protocols such as Tezos [18]). Even alternative Sybil resistance mechanisms such as Proof-of-Replication (used in FileCoin [19]) and privacy data marketplaces [20] end up relying on market making and order matching functionality that is best optimized via agent-based simulation.

II. Simulation Methodology

We will discuss our simulation methodology in two parts: one that evinces the features of our generic simulation platform and another that describes how we modeled agents within our system.

A. Simulation Platform

Our `C++` simulation platform consists of a custom discrete event simulation that emulates the peer-to-peer network of a blockchain system and uses a probabilistic generative model to generate events that corresponds to actions that agents can take within a blockchain system. We use MCMC methods to sample from this model to generate the next event and to propagate this event via the network graph. Each protocol, whether it be Proof-of-Work, Proof-of-Stake, of Proof-of-Replication, has a standardized interface to define its generative model. This interface forces the protocol designer to specify a model for event arrival times, a method for selecting an agent or committee to produce an event, and actions that agents are supposed to take. The protocol designer also has to select a routing algorithm that represents how protocol users are gossiping blocks and off-chain packets with each other. We deliberately separate the routing algorithm from the underlying graph of miners and users, allowing us to test how the system behaves with different gossip protocols. This allows for a protocol designer to test their protocol under deterministic algorithms, such as Kademlia [21], or a randomized gossip method [22]. Moreover, the design of this platform is based on a combination of high-frequency financial backtesting simulations, which use highly optimized, low latency versions of probabilistic networking models, and computational physics simulations, which provide optimized methods for evolving systems of interacting objects.

The simulation also includes a domain-specific language for describing the statistical calculations and policies of individual agents. In order to mimic cryptographic simulation-based proofs [23] and ideal functionalities, we allow for agents to directly interact with the MCMC model. This allows for honest agents to

mutate stochastic components of the simulation related to their local state (e.g. hash power, bonded stake or space) and lets Byzantine agents mutate the state of other agents. Agent computations are not restricted and allowed to be generic; for instance, an agent can use a trained TensorFlow or PyTorch model as a policy. Given that there is a modicum, at best, of data stored in blockchains, all agent policies used in the sequel will be rules-based and closer to high precision, low recall trading strategies versus low precision, high recall reinforcement learning policies.

B. Agent Design Methodology

We use agent-based models to model rational, Byzantine, and adversarial miner strategies. We assume, without the loss of generality, that our agents satisfy the Byzantine-Altruistic-Rational assumption [24]. Agents are represented via a state and action space model, in which agents provide a utility function $U : \mathcal{S} \to \mathbb{R}$ that is used to adapt a policy, $\pi : \mathcal{S} \to \mathcal{A}$, where the state space is S and the action space is $\mathcal{A}$. In this paper, we define our state space to be $\mathrm{A}_t \times \Delta^{N_{\text{shards}}}$, where A_t is the miner's local arboretum at time t and $\Delta^{N_{\text{shards}}}$ represents the miner's hash power distribution. Note that we explicitly exclude block withholding from the action space, as the only choices that a miner can make involve changing their hash power distribution. In particular, this also means that our action space is simply $\Delta^{N_{\text{shards}}}$, as any action corresponds to a choice of hash power distribution. As this is preliminary work, we excluded selfish mining and withholding attacks to simplify the statistical analysis[1] in §4. We will expand upon these results and include selfish mining and withholding attacks in future work. Agents adjust their policies by attempting to optimize their expected reward under a prescribed utility function. For simplicity, we assume that the utility function is C^1, so that we can optimize it via gradient descent. Drawing inspiration from the reinforcement learning literature, we define the expected reward as the k-step exponential moving average ($\alpha < 1$) of the utility function:

$$\mathsf{E}_\alpha[R_t] = \sum_{\tau=0}^{k-1} \alpha^\tau U(S_{t-\tau})$$

where $S_t \in \mathcal{S}$ is the state observed by the agent at time t. In this paper, S_t is simply the agent's local copy of the Chainweb arboretum at time t and their current hash power distribution. From here on, we fix $\alpha = \frac{1}{2}$ and let $\mathsf{E}[R_t] = \mathsf{E}\left[R_t, \frac{1}{2}\right]$. Thus, given a utility function, an agent will update their state via the ordinary gradient descent iteration,

$$S_{t+1} = S_t - \eta \sum_{\tau=0}^{k-1} 2^{-\tau} \nabla_{S_{t-\tau}} U(S_{t-\tau}).$$

Using gradient descent also helps us enforce a locality constraint that ensures that miner strategies are purely local and not reliant on long-term historical results.

In order to program the agents within our simulation environment, we have developed a custom domain-specific language that aims to optimize the computation of utilities and policies for each simulated agent. This will be discussed in more detail in a subsequent paper by a subset of the authors. Using this language, we can specify what statistical signals trigger agents to take actions, allowing for us to describe complex agent strategies and policies via a simple script. Our language also allows for us to treat agents as a template, which lets us take a set of nodes and assign the same strategy to that set of nodes. This allows for easy specification of agent distributions, allowing for a user to specify policies $\pi_1, \ldots, \pi_n$ and their relative proportions $p \in \Delta^n$.

1) Utility Function: The authors of [25] show that in the continuous time and zero latency (e.g. no peer-to-peer network) setting, Markovian agents maximize their expected reward in Bitcoin by optimizing a utility function propotional to $\frac{R(t)}{H(t)}$, where $R(t)$ is the number of rewards that the agent has accrued at time t and $H(t)$ the agent's hash power expenditure at time t. Since this paper assumes that $H(t)$ is constant and uniform for all agents, we can simply define the single chain utility function to be:

$$U_{i,\alpha}(S_t) = U_{i,\alpha}(A_t, h_t) = \frac{\gamma(\text{height}(\mathcal{C}_{\alpha,t}))}{h_t}$$

where $h_t \in (0, 1)$ is the fraction of hash power placed on chain $\mathcal{C}_{\alpha,t}$ is the αth chain of the arboretum A_t and γ is the fraction of blocks in chain $\mathcal{C}$ that agent i produced (see equation 4). Since both values fall in $(0, 1)$, we've removed the issue of ensuring that the dynamic ranges of these quantities are compatible and will not under/overflow. In future work, we plan on allowing h_t to be an arbitrary positive real number, representing energy costs, which will force us to add in a scaling constant to this utility function.

In the multiple chain setting, we simply define the ith miner's utility function as:

$$U_i(S_t) = \sum_{\alpha \in [N_{\text{shards}}]} U_{i,\alpha}(S_t)$$

[1]Our future work will discuss using adaptive sampling methods to deal with the non-stationarity that occurs when adding block withholding

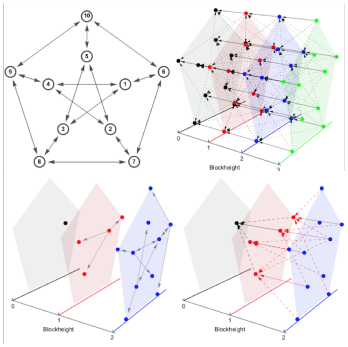

Figure 1. Visualization of Chainweb Base Graph using the Petersen graph as base protocol configuration

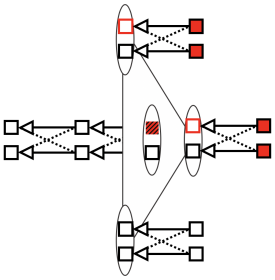

Figure 2. A visualization of the multiple local copies of Chainweb that are maintained by miners, here depicted as ovals, with blocks being generated from left to right and Merkle root references as arrows; this replication generates the arboretum structure. The red block represents two conflicting blocks in the same location, creating a fork which must be reconciled.

III. Chainweb Details

For demonstration of the proposed modeling techniques we will analyze Chainweb [26], a parallelized Proof-of-Work network architecture designed to provide high security and high transaction throughput. Chainweb serves as an ideal use case for simulation because designers have to choose $\Omega(N_{\text{shards}}^2)$ parameters and because it is difficult to formally prove that Chainweb is resistant to miner censorship. As there has not been a formal description of Chainweb in the literature, we will provide both an informal description (with visualization) and a formal description that proves that the parallelized system achieves liveness.

A. Informal Description

Chainweb is a parallel chain Proof of Work architecture that combines hundreds or thousands of individually mined peer chains into a single network. Figure 1 depicts ten chains that are connected via the Petersen graph. The three-dimensional figures show the dependencies of blocks at different heights on blocks of lower heights. Each chain in the Chainweb braid maintains its own ledger, and blocks on each chain are mined with Nakamoto-style Proof of Work hashing. Peer chains incorporate each other's Merkle roots to enforce a single super chain that offers an effective hash power that is the sum of the hash rate of each individual chain. Each chain in the network mines the same cryptocurrency which can be transferred cross-chain via a Proof-of-Burn verified with trustless Simple Payment Verification (SPV) at the smart contract level. The configuration of the Chainweb braid and the relationship between chains is fixed at launch and can be hard forked to larger configurations as throughput demands. The majority of miners are expected to mine the entire Chainweb braid, and as such each miner will maintain its own local version of Chainweb, the braid of which will be recombined with those of other miners as blocks are created.

B. Formal Description

Notation:

- For any $n \in \mathbb{N}, [n] = \{1, 2, \ldots, n\} \subset \mathbb{N}$
- $\mathcal{T}_{h,d}$ is the set of a directed trees with maximum height h and maximum degree d
- $\mathcal{T} = \lim_{h,d \uparrow \infty} \mathcal{T}_{h,d}$. We can take this limit as there is a natural lattice: $\mathcal{T}_{h,d} \subset \mathcal{T}_{h',d'}$ if $h < h'$ and $d < d'$
- $\mathcal{B} \subset \{0,1\}^*$: The space of admissible blocks
- $\mathcal{T}_\emptyset$: The tree with one node that represents the genesis block (e.g. the empty blockchain). We define $\mathsf{height}(\mathcal{T}_\emptyset) = 0$
- If we have a graph $G = (V, E)$ define the boundary operator $\partial : V \to 2^V$ by $\partial(v) = \{w : (v, w) \in E\}$

From single chain to multiple chains with constraints

In order to fully-specify a single chain PoW system, one needs to define the following [1]:

1) Network graph that describes how miners and users communicate
2) Merkle trees for each miner that represent each miner's copy of the blockchain.
3) A process for dynamically growing and updating the Merkle tree

When we refer to trees, we will refer to a block tree, where each vertex of the tree represents a block. In the multiple chain world, in order to balance security and throughput, we need to replace the single Merkle tree rooted at each miner with a set of Merkle trees and a set of constraints that do not let the different chains get too far out of sync with out other. To formalize this,

we will first need to define the terms network graph, base graph, and arboretum:

Definition 1. *A **network graph** is a weighted, undirected graph $G_{network} = (V_{network}, E_{network}, W_{network})$, where each vertex $v \in V_{network}$ represents a miner, each edge $e \in E_{network}, e = (v, w)$ represents a peer-to-peer network connection between miners v and w, and $W_{network} : E_{network} \to \mathbb{R}$ maps an edge to a latency. An N_{shards}-**base graph** is an unweighted, undirected graph $G_{base} = ([N_{shards}], E_{base})$, where each vertex represents a separate chain and each edge represents a constraint.*

In practice, the network graph is changing as users join and leave the network and the edges change in accordance with the peer-to-peer networking protocol.[2] In this work, we will consider the network graph static and unchanging during the remainder of our analysis. We next need to define how our constraint set, represented by G_{base}, affects the allowable Merkle trees.

Definition 2. *An* arboretum *A is a triple (V, E, T) where $V \subset \mathbb{N}$ is a vertex set, $E \subset 2^{\binom{V}{2}}$ is an edge set and $T : V \to \mathcal{T}$ is an operator that maps a vertex to a tree. If $\Lambda : E \times \sqcup_{v \in V} T(v) \to \{0, 1\}$ is a predicate, a Λ-**arboretum** is an arboretum that $\Lambda((v, w), T(v) \sqcup T(v)) = 1$ for all $(v, w) \in E$. Finally, if $G' = (W, F)$ is a subgraph of $G = (V, E)$, we define a **subarboretum** centered on (W, F) to be $(W, F, T|_W)$*

An arboretum is distinct from a forest, as a forest does not prescribe connectivity between trees, whereas the edge set in an arboretum provides a natural graph distance between any two trees. Figure 2 depicts an arboretum where G_{network} is the triangle graph and G_{base} is a length-1 path with two vertices and one edge. The use of an arboretum is a key feature that is important to Chainweb's performance claims, as the choice of graph can dramatically affect performance in a sharded PoW setting. More precisely, if an edge $e = (v, w) \in E_{\text{base}}$, this means that ith block of chain v can only be added if the $i-1$st block of w has been added. Thus, a valid Chainweb arboretum will need to satisfy the following predicate for all $(v, w) \in E$:

$$\begin{aligned}
&\Lambda_{\text{Chainweb}}((v, w), T(v) \cup T(w)) = \\
&(\mathsf{height}(T(w)) == \mathsf{height}(T(v)) - 1) \\
&\vee (\mathsf{height}(T(w)) - 1 == \mathsf{height}(T(v))) \\
&\vee (\mathsf{height}(T(w)) == \mathsf{height}(T(v))) \\
&\vee (\mathsf{height}(T(v) == 0) \\
&\vee (\mathsf{height}(T(v) == 0) \vee \mathsf{compatible}(T(v), T(w))
\end{aligned} \tag{1}$$

The last predicate, $\mathsf{compat}(T(v), T(w))$ ensures that the block at the tip of $T(v)$ at height h includes a header from a block at height $h - 1$ of $T(w)$. This condition is illustrated in figure 2, where we see the two blocks in red depending on both their predecessor on the same chain and the adjacent chain. By assumption, we have $\mathsf{compat}(T_\emptyset, T_\emptyset) = 1$ Finally, given a predicate Λ, we define a dynamic process on an arboretum that represents a single miner's mining activity:

Data: An initial arboretum
$A_0 \leftarrow ([N_{\text{shards}}], E, T_0)$, where $T_0(i) = \mathcal{T}_\emptyset$,
$t \leftarrow 0$
while *true* **do**
  $\tilde{A}_{t+1} = (\tilde{V}, \tilde{F}, \tilde{T}_{t+1}) \leftarrow$ largest Λ-subarboretum of A_t ;
  $(i, B) \in \tilde{V} \times \mathcal{B} \leftarrow$ Apply mining strategy and find chain i that is admissible and mine a valid block B on it (possibly causing a fork);
  $T_{t+1} \leftarrow$ Apply block B to $\tilde{T}_{t+1}(i)$
end

In order to prove that the system achieves liveness, assuming that there is a non-zero amount of hash power on every shard, we need to show that this process can continue in an infinite loop without halting for Chainweb. This can happen, for instance, if there are no $\Lambda_{\text{Chainweb}}$-subarboreta of A_t. We will show that this cannot happen:

Claim 1. *If $G = (V, E)$ is connected, then for all $t \in \mathbb{N}$, there exists a non-empty $\Lambda_{Chainweb}$-subarboretum of A_t*

This claim is proved in the appendix. Let $B_t = (C_t, D_t, \tilde{T}_t)$ be the non-empty $\Lambda_{\text{Chainweb}}$-subarboretum of A_t. We call C_t the *cut set* of the chain at time t. The above claim says that the cut set is never empty and we can always make progress and the liveness of the set of chains reduces to the liveness of the individual chains. Chainweb thus consists of a miner graph G and a series of arboreta, one for each vertex, denoted

[2]Formally, if we think of all possible node sets N as being a subset of $[N_{\max}]$, then we can view the networking protocol as a map $\rho : 2^{N_{\max}} \to 2^{\binom{N_{\max}}{2}}$, which maps a node set into an edge set.

$A_{t,v}$. We note for completeness that one can also frame Chainweb as an arboretum on the strong graph product of a network graph and base graph.

C. Implementation of Chainweb

We represent Chainweb via a graph of miners each of whom holds a copy of their local arboretum. We sample new blocks from a probabilistic model representing Chainweb that has the following parameters:

- λ_α: Block frequency for chain α
- $\zeta_i \in \Delta^{N_{\text{shards}}}$: Hash power distribution of miner i
- $H_i \in \mathbb{R}_{\geq 0}$: Hash power of miner i
- H_{total}: The total hash power of the system, defined as
$$H_{\text{total}} = \sum_{i=1}^{N_{\text{net}}} H_i$$
- Γ_α: The fraction of hash power on chain α, defined as:
$$\Gamma_\alpha = \frac{1}{H_{\text{total}}} \sum_{i=1}^{N_{\text{net}}} H_i \zeta_{i,\alpha}$$

We sample a new block on chain α at time t created by miner j, $B_{\alpha,j,t} \in \mathcal{B}$, via the following generative model:

$$\begin{aligned} \alpha &\sim \mathsf{Multinomial}(\Gamma_1, \ldots, \Gamma_\alpha) \\ j &\sim \mathsf{Multinomial}(\zeta_{1,\alpha}, \ldots, \zeta_{N_{\text{net}},\alpha}) \\ t &\sim \mathsf{Poisson}(\lambda_\alpha) \end{aligned}$$

Given this model and initial conditions (such as a choice of genesis block and random number seeds), our simulation platform can sample a trajectory of how Chainweb might evolve. Each miner's policy is represented via a function that adapts and optimizes their own hash power distribution, ζ_i, which allows for the miner to affect the sampled trajectory. We also note that this model shares a number of similarities with multivariate Hawkes processes [27]. In all simulations performed in this paper, we make the following assumptions about the above parameters (as these likely reflect the choices in Chainweb upon launch [26]):

- All chains have the same block production rate: $\exists \lambda \in \mathbb{R}_{\geq 0}\ \forall \alpha \in [N_{\text{shards}}]\ \lambda_\alpha = \lambda$
- We use uniform hash power, e.g. $\exists H \in \mathbb{R}_{\geq 0}\ \forall i \in [N_{\text{net}}]\ H_i = H$
- Our network graph will only contain miners (e.g. we are assuming that all transaction generating participants are also miners)
- All simulations in this paper route blocks using either k-nearest neighbor routing (like Bitcoin) or use randomized gossip [22]
- All latencies were sampled from the Bitcoin latency distribution in [28]

Finally, we note that we do not include an explicit difficult adjustment in our simulations, even though it is supported within the platform. While changing the difficulty adjustment time window can have dramatic effects on the profitability of selfish mining [29], we wanted to reduce the noise in our statistics and plan on describing selfish mining optimization in a subsequent work.

IV. Results

We performed two experiments to validate our simulation methodology and to estimate endogenous costs of security in Chainweb. Our goal is to illustrate how modeling adversarial and networking behavior can help choose design parameters such as λ_i and the choice of base graph that is used in the production client.

A. Network Analysis

Our first experiment aims to test how different network and base graph configurations lead to changes in system dynamics under different routing profiles. As Claim 1 only guarantees that we will eventually achieve liveness in Chainweb provided that all chains have non-trivial chain growth, our goal is to perform a numerical study of how realistic chain growth looks and to find an estimate for how long it takes for a block to reach the whole network. Statistically estimating chain growth and the time to achieve liveness is crucial for deciding how to set parameters the block production rates, λ_i, as there is a natural trade-off between liveness time and block production rate [30]. Since Chainweb's throughput is bounded by the diameter of the base graph [26], we aimed to use graphs that are the best known solutions to the degree-diameter problem. For these experiments, we used $N_{\text{net}} = 8192$ and $N_{\text{shards}} = 57$, with the choice of shards due to the existence of a Moore Graph, the Hoffman-Singleton graph, that solves the degree-diameter problem [31].

We will describe the statistics of interest. For a miner i and chain α, let $h_{i,\alpha}(t) \in \mathbb{N}$ be the height of the longest branch of chain α that miner i has seen. For notational convenience, we assume that $h_{i,\alpha}(t) = 0$ for $t < 0$. Define the *height function*, $H : \mathbb{R}_{\geq 0} \to \mathbb{R}$ as:

$$H(\tau) = \frac{1}{N_{\text{net}}} \frac{1}{N_{\text{shards}}} \sum_{i=1}^{N_{\text{net}}} \sum_{\alpha=1}^{N_{\text{shards}}} \mathsf{E}_t[h_{i,\alpha}(t)) - h_{i,\alpha}(t-\tau)] \tag{2}$$

where E_t is the expectation over all environments up to time[3] t. In practice, if we simulate k trajectories until time T, we can approximate this expectation as:

$$\mathsf{E}_t[h_{i,\alpha}(t) - h_{i,\alpha}(t-\tau)] \approx \frac{1}{kT} \sum_{t \leq T} (h_{i,\alpha}(t) - h_{i,\alpha}(t-\tau))$$

Intuitively, this tells us what the expected height change of the blockchain is within a window of size τ. By averaging over windows that start at different times t, we can de-noise the variations of this measurement and observe behaviors that are persistent across different initial conditions. Finally, let $\tau_{i,\alpha,k}$ be the time that the kth miner receives block i on chain α. Let $\hat{\tau}_{i,\alpha,k}$ be the random variable that is $\tau_{i,\alpha,k}$ conditional on block k being on the main branch of chain α. Define the *liveness time* $\overline{\tau}$ to be:

$$\overline{\tau} = \frac{1}{N_{\text{shards}}} \sum_{i=1}^{N_{\text{shards}}} \mathsf{E}_k \left[\max_{k \in [N_{\text{net}}]} \hat{\tau}_{i,j,k} - \min_{k \in [N_{\text{net}}]} \hat{\tau}_{i,\alpha,k} \right] \tag{3}$$

This measures the average time that it takes for a block that reaches the main chain to reach all miners. If we use randomized gossip, one expects this time to simply be the covering time of random walk on the miner graph, which is controlled by the spectral gap of the miner graph [32]. However, since we are conditioning on blocks that make up the main chain, this number can be significantly greater than the covering time.

1) Verification Runs: We performed a few verification runs to show that our simulation is replicating the behavior of Chainweb. For these runs, we let set the base graph be equal to the complete graph, $G_{\text{base}} = K_{N_{\text{shards}}}$. In this situation, since every shard depends on every other shard, we expect the time between successive blocks on the same chain to have super-linear scaling in the size of the base graph. We ran 100 simulations with 16,384 miners and a base graph with $|V_{\text{base}}| \in \{2^k : 1 \leq k \leq 13\}$ and calculated this time. In figure 3, we see super-linear scaling (note that both scales are logarithmic) and decreasing variance as we increase the base graph size. We also note that one expected some finite-size effects as $|V_{\text{base}}| \approx |V_{\text{network}}|$, which is the likely cause of the decay in the rate of growth of this curve.

2) Liveness Time: To assess liveness time (3) under realistic conditions, we constructed a realistic internet graph, the Barabási small world graph [33], used a randomized gossip protocol [22], and sampled latencies the Bitcoin latency distribution [28]. In order to stress the system, we assumed a block production rate

[3]Our Chainweb model is a composed of Markov models and inherits a natural filtration that E_t is defined on (regardless of initial environment).

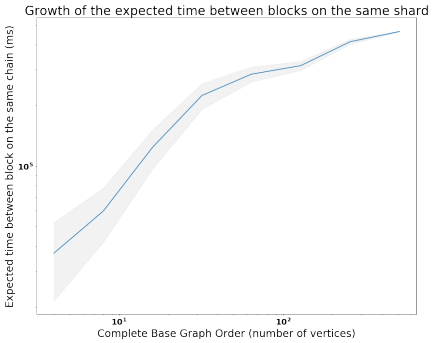

Figure 3. This figure shows the average inter-arrival time (e.g. time between successive blocks on the same chain) as a function of $n = |V_{\text{base}}|$ with $G_{\text{base}} = K_n$, the complete graph on n vertices. We expect super-linear growth in this time, as each chain has to wait on all other chains before it can make progress. This continues to be true even up to $|V_{\text{base}}| = |V_{\text{network}}|/2$.

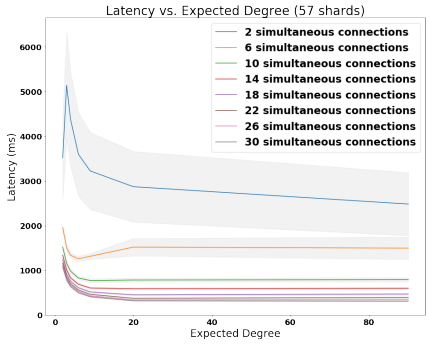

Figure 4. In this figure, which plots expected latency (and confidence intervals, represented via error bands) against expected degree of a Barabśi graph, we see that by increasing the bandwidth used, we quickly saturate the graph.

of $\lambda = 1$Hz. In figure 4, we see a plot of the expected degree of a Barabási graph versus liveness time. We used a miner graph of size 32,768 and used the Hoffman-Singleton graph as a base graph to generate these figures. The different curves correspond to a different number of simultaneous connections, which is how we quantify bandwidth. This corresponds to the upper bound on the number of neighbors forwarded to by a miner. We can see that by the time we get to 10 simultaneous connections, we are close to saturating the high-bandwidth limit, even for low-connectivity Barabási graphs. Data of this form helps protocol developers assess design decisions and choose peer-to-peer networking capabilities whose bandwidth-latency tradeoff matches the expectations of the consensus mechanism.

3) Random Miner Graph: In order to assess how Chainweb performs under more extreme network graphs, we looked at a modified version of Erdös-

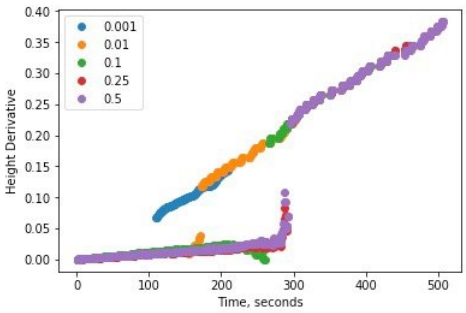

Figure 5. Computation of $H(\tau)$ (denoted height derivative) for times within one block interval ($\lambda = \frac{1}{600}$Hz.) The legend indicates the Erdös-Renyi probability p that was used to generate G_p.

Renyi random graphs G_p such that if $i \in [N_{\text{net}}-1], j \in [N_{\text{net}}] - \{i+1\}$, then the edge (i,j) is included with probability p and such that the edge $(i, i+1)$ is always included. We chose this ensemble because it ensures that our graph is connected (via the inclusion of the line edges $(i, i+1)$), but still inherits the Erdös-Renyi phase transition. In these experiments, we set the block production rate to be that of Bitcoin ($\lambda = \frac{1}{600}$Hz) so that these results could be interpreted in terms of real world data. Moreover, we averaged over 10 instances of each random graph G_p and used the Hoffman-Singleton graph as a base graph.

In Figure 5, we see computations of $H(\tau)$ within one block interval. We see that for high values of p (e.g. the miner graph is more connected), there is a sharp uptick in expected height increase around a half block time (300s). As we decrease connectivity, we expect there to be a more gradual height increase as some of the graph will have received the latest block, whereas other are still waiting to receive it and/or are on other forks. This is the expected behavior: when height increases propagate uniformly and rapidly throughout the network graph, we expect a sharp uptick in expected height change, as most participants receive the block (on average) at the same time (since miners have the same hash power). However, it is curious to note that the Erdös-Renyi phase transition [34] takes place around $p = 0.001$ for our graphs and yet we only observe the signs of the transition (e.g. the sharp derivative for $p \in \{0.1, 0.25, 0.5\}$) far away from it. This suggests that the presence of forks can dramatically slow down block propagation and adds in a non-linear latency effect. This effect can likely be measured by miners, who measure their deviation from these expected curves, who can potentially use this information to boost the rewards from selfish mining, akin to latency arbitrage in high-frequency trading.

B. Adversarial Censorship

In order to test how resilient Chainweb is to miners attempting to censor a chain α by not allocating any hash power to α, we used our domain specific language to construct two agents that do the following:

1) Honest Agent: Computes utility and gradient on all chains and updates its hash power distribution via gradient descent
2) Adversarial Agent: Computes utility and gradient on all chains, updates its hash power via gradient descent, sets the hash power allocated to the censored chain to zero, and then redistributes the excess hash power equally.

We chose a sequence of twenty five evenly-spaced adversary fractions $f_i \in (0,1)$ and for each of these, ran 256 simulations (with the Hoffman-Singleton base graph) of 30,000 blocks produced with a $(1-f_i)$ fraction of honest agents and f_i censoring agents. For each miner i and chain α, we computed the fraction $\gamma_{i,\alpha}(h)$ of blocks on the main branch of chain α that were created by miner i at block height h, where $\forall \alpha \in [N_{\text{shards}}] \forall h \in \mathbb{N} \sum_{i \in [N_{\text{net}}]} \gamma_{i,\alpha}(h) = 1$. For simplicity, we assume that the block reward is constant and is equal to one coin per block throughout our simulations (as all of our plots are about relative rewards, this choice of units does not affect our results). We define the following two quantities:

$$\gamma_{\text{honest}}(h) = \frac{1}{|S_{\text{honest}}|} \frac{1}{N_{\text{shards}}} \sum_{i \in S_{\text{honest}}} \gamma_{i,\alpha}(h) \quad (4)$$

$$\gamma_{\text{adversary}}(h) = \frac{1}{|S_{\text{adversary}}|} \frac{1}{N_{\text{shards}}} \sum_{i \in S_{\text{adversary}}} \gamma_{i,\alpha}(h) \quad (5)$$

where $S_{\text{honest}} \subset [N_{\text{net}}]$ is the set of honest miners and $S_{\text{adversary}} = [N_{\text{net}}] - S_{\text{honest}}$. In Figure 6, we see the plot for a single trajectory with $f_i = 0.1$, where we see that $\gamma_{\text{honest}} > \gamma_{\text{adversary}}$ and that the honest curve is much smoother than the noisy adversary curve.

In Figure 7, we see how the expected difference in rewards and risk-adjusted rewards vary as a function of $f \in (0,1)$. We measure risk-adjusted rewards using the Sharpe Ratio, a common measure of trading strategy performance, which is measured via the mean reward (over all trajectories sampled) divided by the standard deviation of the reward. As the plots illustrate, honest, profit-maximizing agents beat out the censoring agents until 50-60%, at which point the censoring adversary is increasingly favored.

From this data, we can conclude that it appears to be hard for miners that are aiming to censor a chain to successfully do it as long as there are enough rational, profit-maximizing miners in the system.

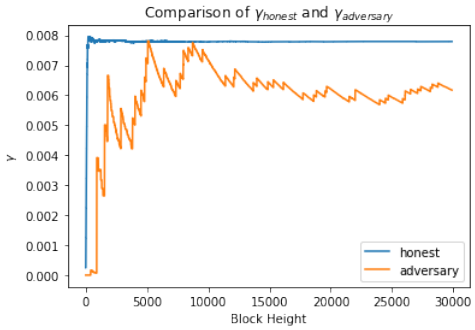

Figure 6. In this image, we see that the expected fraction of honest blocks, $\gamma_{\text{honest}}(h)$ stabilizes rapidly, whereas $\gamma_{\text{adversary}}(h)$ fluctuates wildly, with excursions to the stable point that is achieved by honest agents. This data is taken from a single simulation.

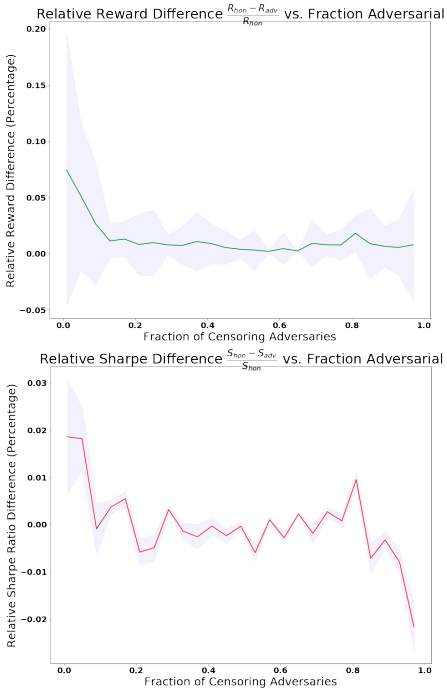

Figure 7. In the top figure, we see the relative expected reward, $\frac{\mathsf{E}[R_{hon}]-\mathsf{E}[R_{adv}]}{\mathsf{E}[R_{adv}]}$ between expected adversary rewards and honest rewards. We see that the honest miner advantage decays quickly from 7% to 1%, although honesty appears to be the dominant strategy. On the other hand, the risk adjusted returns, measured via the Sharpe Ratio $S_\alpha = \frac{\mathsf{E}[R_\alpha]}{\sqrt{\mathsf{Var}[R_\alpha}}$, $\alpha \in \{hon, adv\}$, has a stronger transition around 80.

V. Conclusions and Further Work

We explored how adversarial, agent-based simulation can be used to assess claims and measure networking behavior for a blockchain protocol. Our techniques helped us assess different design choices, such as how the choice of base graph affects liveness time, and we were able to statistically estimate the rewards sacrificed by an adversarial, censoring agent. These techniques helped us evaluate the safety of Chainweb and help us statistically justify that degree-diameter minimizing base graphs prevent censorship attacks. As our experiments with liveness time show, one can use these techniques to optimize peer-to-peer networking performance based on various metrics, such as the number of simultaneous connection in 4. Finally, we note that more complicated consensus protocols, such as Proof-of-Stake and Proof-of-Space tend to have a variety of other parameters (slashing, market making or auction parameters, etc.) and these can be optimized in a similar fashion, given a certain mixture of honest and adversarial agents. Our future work will include an analysis of block withholding and selfish mining and analyzing Chainweb (once it is live) by incorporating *exogenous* data to estimate the true cost of security and selfish mining, conditional on the existence of active fiat and derivatives markets.

VI. Acknowledgments

The authors would like to thank Joseph Bonneau, Yi Sun, Emily Pillmore, and the anonymous reviewers for insightful and constructive comments.

References

[1] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

[2] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[3] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[4] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948. ACM, 2018.

[5] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[6] Sandrine Jacob Leal, Mauro Napoletano, Andrea Roventini, and Giorgio Fagiolo. Rock around the clock: An agent-based model of low-and high-frequency trading. *Journal of Evolutionary Economics*, 26(1):49–76, 2016.

[7] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[9] Igor Kotenko, Alexey Konovalov, and Andrey Shorov. Agent-based modeling and simulation of botnets and botnet defense. In *Conference on Cyber Conflict. CCD COE Publications. Tallinn, Estonia*, pages 21–44, 2010.

[10] J Doyne Farmer and Duncan Foley. The economy needs agent-based modelling. *Nature*, 460(7256):685, 2009.

[11] George Grozev, David Batten, Miles Anderson, Geoff Lewis, John Mo, and Jack Katzfey. Nemsim: Agent-based simulator for australia's national electricity market. In *SimTecT 2005 Conference Proceedings*. Citeseer, 2005.

[12] Steve Yang, Mark Paddrik, Roy Hayes, Andrew Todd, Andrei Kirilenko, Peter Beling, and William Scherer. Behavior based learning in identifying high frequency trading strategies. In *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on*, pages 1–8. IEEE, 2012.

[13] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[14] Johannes Göbel, Holger Paul Keeler, Anthony E Krzesinski, and Peter G Taylor. Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23–41, 2016.

[15] Kei-Leo Brousmichc, Andra Anoaica, Omar Dib, Tesnim Abdellatif, and Gilles Deleuze. Blockchain energy market place evaluation: an agent-based approach. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 321–327. IEEE, 2018.

[16] Michele Bottone, Franco Raimondi, and Giuseppe Primiero. Multi-agent based simulations of block-free distributed ledgers. 2018.

[17] Joseph Bonneau. Why buy when you can rent? In *International Conference on Financial Cryptography and Data Security*, pages 19–26. Springer, 2016.

[18] LM Goodman. Tezos—a self-amending crypto-ledger white paper. *URL: https://www. tezos. com/static/papers/white_paper. pdf*, 2014.

[19] Ben Fisch. Poreps: Proofs of space on useful data. Technical report, Cryptology ePrint Archive, Report 2018/678, 2018. https://eprint. iacr. org . . . , 2018.

[20] Nick Hynes, David Dao, David Yan, Raymond Cheng, and Dawn Song. A demonstration of sterling: a privacy-preserving data marketplace. *Proceedings of the VLDB Endowment*, 11(12):2086–2089, 2018.

[21] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[22] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530, 2006.

[23] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[24] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *ACM SIGOPS operating systems review*, volume 39, pages 45–58. ACM, 2005.

[25] Swapnil Dhamal, Tijani Chahed, Walid Ben-Ameur, Eitan Altman, Albert Sunny, and Sudheer Poojary. A stochastic game framework for analyzing computational investment strategies in distributed computing with application to blockchain mining. *arXiv preprint arXiv:1809.03143*, 2018.

[26] Will Martino, Monica Quaintance, and Stuart Popejoy. Chainweb: A proof-of-work parallel-chain architecture for massive throughput. http://kadena.io/docs/chainweb-v15.pdf, January 2018.

[27] Paul Embrechts, Thomas Liniger, and Lu Lin. Multivariate hawkes processes: an application to financial data. *Journal of Applied Probability*, 48(A):367–378, 2011.

[28] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. *arXiv preprint arXiv:1801.03998*, 2018.

[29] Cyril Grunspan and Ricardo Pérez-Marco. On profitability of selfish mining. *arXiv preprint arXiv:1805.08281*, 2018.

[30] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[31] Mirka Miller and Jozef Sirán. Moore graphs and beyond: A survey of the degree/diameter problem. *The electronic journal of combinatorics*, 1000:DS14–Dec, 2005.

[32] László Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46, 1993.

[33] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Internet: Diameter of the world-wide web. *nature*, 401(6749):130, 1999.

[34] Remco Van Der Hofstad. Random graphs and complex networks. *Available on http://www. win. tue. nl/rhofstad/NotesRGCN. pdf*, 11, 2009.

VII. Appendix: Proof of Claim 1

Proof. We will prove this by induction. By construction, A_0 is a $\Lambda_{\text{Chainweb}}$-arboretum since $\mathsf{height}(\mathcal{T}_\emptyset) = 0$ and $\mathsf{compat}(\mathcal{T}_\emptyset, \mathcal{T}_\emptyset) = 1$. Now assume the induction hypothesis that A_{t-1} has a $\Lambda_{\text{Chainweb}}$-subarboretum and suppose for a contradiction that A_t does not have a $\Lambda_{\text{Chainweb}}$-arboretum. We first note that if $\mathsf{compat}(T_t(v), T_t(w))$ is not satisfied, but all of the height conditions are satisfied, then one can start a fork at $T_{t-1}(v)$ or $T_{t-1}(w)$ that forces compatibility. By assumption $T_{t-1}(v)$ and $T_{t-1}(w)$ are admissible, so this is possible. Thus, we can assume that the compatibility condition is satisfied. Since A_{t-1} has a $\Lambda_{\text{Chainweb}}$-subarboretum, $\tilde{A}_t$ was non-empty. Let $i \in [N_{\text{shards}}]$ be the index of the block mined in the second step of the loop. From the definition of $\Lambda_{\text{Chainweb}}$, this means that $\mathsf{height}(T_{t-1}(i)) \in \{\mathsf{height}(T_{t-1}(j)) + \eta : j \in \partial(i_t), \eta \in \{0, 1\}\}$ and this set is non-empty since G is connected. We have two cases:

1) $\mathsf{height}(T_{t-1}(i)) = \mathsf{height}(T_{t-1}(j))$: This implies that $\mathsf{height}(T_t(i)) = \mathsf{height}(T_t(j)) + 1$, so j is admissible in the next round
2) $\mathsf{height}(T_{t-1}(i)) = \mathsf{height}(T_{t-1}(j)) + 1$: This implies that $\mathsf{height}(T_t(i)) = \mathsf{height}(T_t(j)$ in the next round, so both i, j are admissible

Thus j is admissible and there must be a non-empty $\Lambda_{\text{Chainweb}}$-arboretum □